AD-A258 504

Reinforced Concrete Wall Form Design Program

DTIC
ELECTE
DEC 8 1992
S C D

Submitted to

Professor Charles T. Jahren
Professor George Turkiyyah
Professor Fred L. Mannering

Department of Civil Engineering
University of Washington

92-31021

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

By

Leo D. McKinley
Lieutenant, Civil Engineer Corps, U. S. Navy

August 1992

Reinforced Concrete Wall Form Design Program

Submitted to

Professor Charles T. Jahren
Professor George Turkiyyah
Professor Fred L. Mannering

Department of Civil Engineering
University of Washington

Accession For
NTIS GRA&I ☐
DTIC TAB ☐
Unannounced ☐
Justification
By Per Form 50
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

By

Leo D. McKinley
Lieutenant, Civil Engineer Corps, U. S. Navy

August 1992

TABLE OF CONTENTS

INTRODUCTION 1
EXISTING DESIGN METHODS 2
DESCRIPTION OF PROGRAM ARCHITECTURE 2
Global Architecture 3
Detailed Architecture 4
User Interface 4
Design Calculations 5
WOOD ADT 6
Data Tables 6
TABLE ADT 9
GUIDE TO PROGRAM USE 9
Running the Wall Form Design Program Inside the Scheme Interpreter 9
Compiling Scheme Files 10
Wall Form Design Program Interface 11
Loading the Default Wall Form System 11
Changing the Plyform Elements 11
Changing the Stud Elements 12
Changing the Wale Elements 12
Changing the Form Tie Elements 13
Changing the Deflection Criteria 13
Changing the Concrete Criteria 13
Displaying Stud, Wale and Form Tie Spacing 13
CONCLUSION 14
RECOMMENDATIONS 15
APPENDIX A - Scheme Code for the Wall Form Design Program 16
APPENDIX B - Glossary to the *USER INTERFACE* Module 17
APPENDIX C - Glossary to the *DESIGN CALCULATIONS* Module 28
APPENDIX D - Glossary to the *WOOD ADT* Module 37
APPENDIX E - Glossary to the *DATA TABLES* Module 42
APPENDIX F - Glossary to the TABLE ADT Module 46
APPENDIX G - Global Architecture Flow Chart 48
APPENDIX H - Defining the Input Vectors 49
APPENDIX I - Default Wall Form System 52
APPENDIX J - Defining the Element Vectors 54
APPENDIX K - Graphical Illustration of Example Wall Form System 58
APPENDIX L - Hand Calculations for Example Wall Form System 59

INTRODUCTION

Formwork[1] is the container into which plastic concrete is placed; it may be arranged in virtually any manner to achieve the architect's or engineer's desired shape for the structure being constructed. This formwork is a temporary structure which is many times erected on site by experienced construction workers requiring little or no engineering assistance. However, the design of the formwork system lies in fundamental structural engineering calculations to limit the deflection of the members, and to limit the stresses on the members in bending, shear, tension and compression. It is imperative that these calculations be performed to provide a necessary degree of safety and to ensure that the formwork system will completely function to provide the structure intended by the architect or engineer.

The process of performing the design calculations necessary to complete the formwork system is tedious and may require a great deal of an engineer's time. This process is potentially mistake-prone and time consuming. The objective of this paper is to introduce a quick and reliable object-oriented computer program that will assist engineers during the design of formwork systems for reinforced concrete walls.

This paper will outline the global and detailed architecture of the Wall Form Design Program (WFDP), provide a guide for WFDP's implementation, describe the program's five modules and their internal interface, describe WFDP's user interface, and illustrate the lengthy calculations required to design an example wall form system. The programming language code on which WFDP was developed is provided in Appendix A as a separate attachment. However, glossaries for each of the five program modules are provided which define the function of each of the procedures of the code. These glossaries are included as Appendix B (*USER INTERFACE*),

---

[1]The term "formwork" will be used throughout this paper to describe the entire system of support for the freshly placed concrete, to include sheathing plus all supporting members, hardware and accessories.

Appendix C (*DESIGN CALCULATIONS*), Appendix D (*WOOD ADT*), Appendix E (*DATA TABLES*) and Appendix F (*TABLE ADT*).

EXISTING DESIGN METHODS

The predominant method described in text books and manuals for designing wall forms has been by traditional hand calculations[2,3,4]. In an effort to decrease the time required to design formwork, some more recent methods have been developed utilizing charts[5] and tables[6].

DESCRIPTION OF PROGRAM ARCHITECTURE

WFDP was developed with the object-oriented programming language Scheme[7] which operates within the Scheme Interpreter (SCI), or as a stand-alone program in the Unix operating environment after being compiled in the Scheme-to-C (Scheme->C) compiler. An object-oriented programming language has aspects of data abstraction (organizing data of a domain of values with associated operators), data type completeness (all data types are fully manipulable within a specified domain) and polymorphism (references to data types can assume many forms), and data type inheritance (relations among data types are linked)[8]. Data abstraction separates the behavior

---

[2]M. K. Hurd, *Formwork for Concrete*, 5th Ed., American Concrete Institute, Detroit, MI. 1989.

[3]J. G. Richardson, *Formwork Construction and Practice*, Whitefriards Press Ltd, London, England, 1977.

[4]C. J. Wilshire, *Formwork*, Thomas Telford Ltd, London, England, 1989.

[5]P. H. Sommers, "Charts Simplify Design of Vertical Formwork", *Concrete Construction*, April 1984, pp. 392-394.

[6]The Burke Company, *The Concrete Supermarket*, Catalog No. 910, Standard Edition 4/91.

[7]Jonathan Rees and William Clinger (Editors), Revised3 Report on the Algorithmic Language Scheme, SIGPLAN Notices, V21, #12, December 1986.

[8]John W. Baugh, and Daniel R. Rehak, "Data Abstraction in Engineering Software Development", *American Society of Civil Engineering, Journal of Computing in Civil Engineering*, Vol. 6, No. 3, July 1992.

of a data type from its implementation. This separation is obtained by defining the functional behavior of procedures that operate on the values, or instances, of a data type. Users of an abstract data type (ADT) see only its external view (i.e., the set of operators), allowing its implementation to be modified without adverse effects. A data type may be defined as a set of values (i.e., the domain) and a set of operators with both syntactic and semantic descriptions. The code for the entire program is shown in Appendix A. WFDP was developed with Scheme so that it can be compiled to operate as a stand-alone program.

Global Architecture

WFDP was developed in five modules.

1. *USER INTERFACE*
2. *DESIGN CALCULATIONS*
3. *WOOD ADT*
4. *DATA TABLES*
5. *TABLE ADT*

These modules perform specialized tasks and pass information back and forth. The interaction between the program user and all WFDP modules is summarized here and is described in more detail in the next section. The relationships between the WFDP user, the five modules (shown in all capital letters and italics) and the input and Element Vectors are graphically represented in Appendix G.

The user interacts directly with the *USER INTERFACE* module by responding to questions regarding specifics of each of the required wall form elements. The *USER INTERFACE* module takes the information from the user and stores it in the six previously created Input Vectors. The *USER INTERFACE* module then passes the six Input Vectors to the *DESIGN CALCULATIONS* module. By its interaction with the *WOOD ADT* module, the *DESIGN*

CALCULATIONS module will use the information stored in the six Input Vectors to create the four Element Vectors. The *WOOD ADT* module calls the *DATA TABLES* module to retrieve information which will be stored in the Element Vectors. The *DESIGN CALCULATIONS* module then computes the spacing of the elements of the wall form system from the four Element Vectors. This information is then passed back to the *USER INTERFACE* module. The role of the *TABLE ADT* module is to create the format and means of storage and retrieval of information to and from the various data tables used within WFDP.

Detailed Architecture

The five WFDP modules are described in further detail in this section.

USER INTERFACE

The *USER INTERFACE* module executes a series of information collection procedures designed to draw from the user all the data needed to completely design the spacing of the elements of a wall form system, and then display this information back to the WFDP user. This module contains the procedure START-DESIGN which invokes WFDP. Appendix B is a glossary of all the procedures in the *USER INTERFACE* module.

START-DESIGN greets the user and asks if they would like to begin designing a wall form. If the response is "No", then WFDP exits to the SCI prompt. If the response is "Yes", then six Input Vectors (see Appendix H) are created which contain all the input information required to compute the spacing of the wall form elements. These six vectors are initially loaded with the values for the default wall form system listed in Appendix I. Next, the *WOOD ADT* module is invoked to create the four wall form Element Vectors PLYFORM, STUD, WALE and TIE. These four vectors contain all the information for each element that may be required to design the wall form. Appendix J describes the contents of each of the vector's indices for each of the four elements of the wall form. These four Element Vectors are used by the *DESIGN*

CALCULATIONS module to compute the applicable maximum spacing for each element. Appendix K is provided to graphically illustrate a typical wall form system and the positioning of its major components

One of the more useful characteristic of WFDP is the manner by which stylistic and design issues have been separated. As stated above, a default wall form system is loaded to begin the design process. From this point, only minor changes to the wall form system need to be made to customize the form system to the site specific application. This default wall form system can be changed to suit the style of each of WFDP's primary users. Once the default wall form system is loaded, no other stylistic issues remain. All design calculations are based on universal engineering principles.

DESIGN CALCULATIONS

The *DESIGN CALCULATIONS* module computes the spacing of each of the elements of the wall form system (studs, wales and form ties), and is the workhorse of WFDP. The maximum stud spacing is determined by the smallest computed spacing as governed by either bending stresses, shear stresses or deflection of the plyform, based upon the concrete pressure loading of the plyform. The maximum wale spacing is determined by the smallest computed spacing as governed by either bending stresses, shear stresses, deflection of the studs, or compressive stresses between the studs and wales. The loading to be used to determine the wale spacing is based on the previously computed stud spacing and tributary concrete loading of the studs based on that spacing. The maximum form tie spacing is determined by the smallest computed spacing as governed by bending stresses, shear stresses, deflection of the wales, compression between the wales and form tie bearing plates, or maximum bending moment or shear computed from point loads (studs) on the wales. The loading used to determine the tie spacing is based on the computed stud and wale spacings, and the tributary concrete loading of the wales based on that spacing. Appendix C is a glossary of all the procedures in the *DESIGN CALCULATIONS* module.

WOOD ADT

The *WOOD ADT* transforms the information stored in the six Input Vectors from terms of species, grade, nominal dimensions, etc., into terms of allowable stresses, section properties, dressed dimensions, etc., in the four Element Vectors. Additionally, this module provides a means of storing and retrieving specific information for each of the wall form elements for use by the *DESIGN CALCULATIONS* module. Appendix D is a glossary of all the procedures in the *WOOD ADT* module.

The programming code written for WFDP is efficient and significantly contributes to its reusability attribute since the indices of the Element Vectors contain similar information. For instance, index location for every Element Vector holds the allowable bending stress of that element, index location 17 holds the uniform loading of that element, and index location 23 holds the element spacing based on its uniform loading. In fact, new Element Vectors for changing form system requirements can be created with the *WOOD ADT* module, without significant modification to the programming code. There are exceptions which are accommodated in the *WOOD ADT* for loading of each individual Element Vector. For example, index location 21 contains the maximum capacity of the form tie. Of course, this is only applicable to the TIE Element Vector and is taken into consideration in the LOAD-WOOD procedure.

DATA TABLES

The data tables used by WFDP store all the information which may be required to design a wall form system. This information includes lumber and plyform section properties and allowable stresses, nominal lumber dimensions, as well as a table which stores the spacing equations used in the *DESIGN CALCULATIONS* module. The format of the data tables and the means by which information is stored and retrieved is defined using the *TABLE ADT* described below. Appendix E is a glossary of all the procedures in the *DATA TABLES* module.

The values stored in the data table PLY-PROP were reproduced from Table 9 - Section Properties for Plyform Class I and Class II, and Structural I Plyform of the American Plywood Association Design/Construction Guide for Concrete Forming[9]. All plyform section properties used here have been adjusted to account for the reduced effectiveness of plies when their grain is perpendicular to the applied stress. The section properties stored in this table are the exact ones for the three plyform grades (Class I, Class II and Structural I) taking into consideration their special layup restrictions. The various species of wood used in manufacturing plywood have different stiffness and strength properties. Those species having similar properties are assigned to a species group. In order to simplify plywood design, the effects of using different species groups in a panel, as well as the effects of cross-banded construction, have been accounted for in this table. In calculating these section properties, all plies were transformed to properties of the face ply. Consequently, designers need not concern themselves with the actual panel layup, but only with the allowable stresses for the face ply and the given section properties.

The values stored in the data table PLY-STRESS were reproduced from the unnumbered table on page 16 of the Concrete Forming Guide[10]. As concrete forming is a special application, wet stresses have been used and then adjusted for forming conditions such as duration of load (1.25 for bending and rolling shear stresses), and other experience factors (1.30 for bending and rolling shear stresses). Because shear deflection is computed separately from bending deflection, the modulus of elasticity has been increased by 10 percent.

The values stored in the data table LUM-PROP were reproduced from Table 7.1 of the Timber Construction Manual, Third Edition[11]. For practicality, only lumber with nominal widths

---

[9]American Plywood Association Design/Construction Guide, *Concrete Forming*, Form No. V345K, Revised January 1984.

[10]Ibid.

[11]American Institute of Timber Construction, *Timber Construction Manual*, Third Edition, John

of 2, 3 and 4 inches, and nominal depths from 2 to 12 inches have been included in this program. The actual widths and depths returned from this table are the standard dressed sizes obtained from sanding all four sides of the sawn lumber.

The data table NOMINAL is necessary to reduce the size of the data table LUM-PROP by one half. This table takes the nominal width and depth dimensions of the required lumber and returns a single nominal size entry for access to LUM-PROP. For example, if the nominal dimensions of the studs are to be 2 inches by 4 inches, then NOMINAL would be called with the first entry "2" and second entry "4", thereby returning the symbol "s2x4" (size 2 inches by 4 inches). This symbol will be used to access LUM-PROP for all the section properties (actual width, actual depth, area, section modulus and moment of inertia) of nominal 2 by 4 inch lumber.

The values stored in the data table LUM-STRESS were reproduced from Table 7.3 of the Timber Construction Manual, Third Edition[12]. The design values stored in this table are for normal loading conditions. All values are for lumber which is surfaced dry or surfaced green and used at 19 percent maximum moisture content.

The data table CFDE stores spacing equations in its records instead of values. These equations are based on the principles presented in Chapter 6 of *Formwork for Construction*[13]. This table is a convenient way of reducing the amount of Scheme code required to call a particular spacing equation. To return the appropriate equation for use in determining maximum spacing, CFDE requires as entry keys the governing stress being checked (i.e., bending, shear or deflection) and the support conditions (i.e., continuous over 1, 2 or 3 or more spans). The spacing equations have been previously defined, as shown in Appendix A, and take as arguments

---

Wiley & Sons, New York, 1985.

[12]Ibid.

[13]Supra

the lumber or plyform allowable stress (bending, shear or modulus of elasticity), section property (section modulus, rolling shear constant, moment of inertia or area) and uniform member loading.

TABLE ADT

The *TABLE ADT* was designed to provide a means of readily storing and retrieving the massive amount of information for each type and size of lumber and plyform regarding section properties and allowable stresses. However, the *TABLE ADT* module can create one-, two-, three- or four-deminsional tables capable of storing and retrieving literally any type of information, not just numbers as used in this program,since this data structure is virtually independent of its external application. For instance, the information in a telephone directory can be stored and retrieved in one of these two-deminsional tables using last names as the first key, first names as the second key, and telephone numbers as the value stored and retrieved based on these keys. Appendix F is a glossary of all the procedures in the *TABLE ADT* module.

GUIDE TO PROGRAM USE

Running the Wall Form Design Program Inside the Scheme Interpreter

WFDP consists of eight files written in the Scheme programming language.

1. cfde.sc: This file contains the code for the creation of the equations for spacing as governed by bending, shear and deflection of the wall form elements and the table to contain these equations.
2. default.sc: This file contains the code for the creation of the six Input Vectors and fills their indices with the default wall form system.elements.
3. design.sc: This file contains the code for the *DESIGN CALCULATIONS* module
4. iface.sc: This file contains the code for the *USER INTERFACE* Module
5. prop.sc: This file contains the code for four of the data tables, PLY-PROP, PLY-STRESS, LUM-PROP and NOMINAL

6. stress.sc: This file contains the code for the data table LUM-STRESS.

7. table.sc: This file contains the code for the *TABLE ADT*.

8. wood.sc: This file contains the code for the *WOOD ADT*.

To initiate WFDP from within SCI, first load the eight program files by typing the following at the SCI prompt.

(load "wallform.sc")

This invokes a procedure that calls a series of modified file-loading procedures that loads the eight files into SCI. After the files have been loaded, type the following procedure at the SCI prompt.

(start-design)

This invokes WFDP. Type Control C simultaneously to exit WFDP at any time.

Compiling Scheme Files

WFDP may be compiled to improve its operation. There are three distinct advantages for utilizing the compiled version of WFDP. First, when the executable file "wallform" is called at the Unix command prompt, it takes only a matter of seconds for WFDP to load. In order load WFDP into SCI, a separate procedure must be called (i.e., load "wallform.sc") and the loading time is between 5 and 9 minutes, depending on overall system demand at the time. Second, execution time for each procedure, or the time to respond to questions asked of the user, is an order of magnitude shorter for the compiled version of WFDP. Finally, the entire content of the user's session within WFDP can be viewed by placing the cursor in the current window pad that is open and running WFDP.

To compile the eight Scheme files into a stand-alone, executable program, type the following at the command prompt in the Unix operating environment.

```
>scc -o wallform iface.sc cfde.sc default.sc design.sc prop.sc stress.sc table.sc wood.sc
```

Where scc[14] is the Scheme->C compiler and "-o wallform" is a command option that links the eight Scheme code files into one executable file called "wallform". After compilation, typing "wallform" at the Unix command prompt will invoke WFDP. Note that "module" procedures are required at the beginning of each file, as shown in Appendix A. The file with the initiating procedure START-DESIGN must have included in the "module" procedure a "with" and "main" statement. The "with" statement must list each "module" name in the same order they would be loaded into SCI for proper operation and the "main" statement must identify START-DESIGN as the initiating procedure for WFDP.

Wall Form Design Program Interface

Loading the Default Wall Form System

After loading WFDP, a short introduction will appear and you will be asked whether you are ready to begin designing a wall form system. If you respond "No", then WFDP will return to the command prompt. If you respond "Yes", then the default wall form system is loaded and displayed to you with the information shown in Appendix I. WFDP then asks if you would like to change the current wall form system.

Changing the Plyform Elements

If you would like to change the plyform element of the wall form system, then you have the option of either inputting values for the plyform's allowable bending stress, allowable rolling shear stress and modulus of elasticity, or having WFDP retrieve these values based on the selected plyform's grade.

---

[14]This program first converts the code developed in the programming language Scheme to code in the programming language C. After this transformation, SCC calls the C compiler, CC, which compiles that code into a stand-alone, executable program.

Regardless of the option selected above, the user will required to select options for plyform grade, thickness, support conditions, and direction of the plyform's face grain relative to the stud span. Notice the user is required to input the plyform's grade regardless of options selected above. This is necessary since the section properties are also selected based upon the plyform's grade.

Changing the Stud Elements

If the stud element of the wall form system is to be changed, you have the option of either inputting values for the stud's allowable bending stress, allowable compressive stress, allowable shear stress and modulus of elasticity, or having WFDP retrieve these values based on the species, grade, size classification, and type of bending stress (single or repetitive) of the selected studs.

Regardless of the option selected above, the user will be required to select options for the studs' nominal width, nominal depth, and support conditions. Only three size classifications of lumber are available to be selected in WFDP

1. 2 to 4 inches wide and 2 to 4 inches deep.
2. 2 to 4 inches wide and 4 inches deep.
3. 2 to 4 inches wide and 5 or more inches deep.

Accordingly, only three nominal widths are available: 2, 3 or 4 inches. The nominal depth options available to the user differ with each selection of nominal width since, for each nominal width, only certain nominal depths of lumber are manufactured. Additionally, the lumber grades "construction", "standard" and "utility" are only manufactured at a nominal depth of 4 inches so no options, except four inches nominal depth, are available if one of these three grades are selected.

Changing the Wale Elements

Exactly the same options as changing stud elements are available to the user if the wales are to be changed, except that the user may select also single or double wales for the wall form system.

Changing the Form Tie Elements

If you would like to change the form tie element of the wall form system, then you have the option of changing the maximum capacity of the form ties, the shape of the form ties, and the dimensions of the bearing plates used with the form ties. If the bearing plates are rectangular, then you will be asked for the width and length. The width is input first and is assumed to be the bearing plate dimension running across the width of the wales. The length is then input as the bearing plate dimension running along the length of the wales. If the bearing plates are circular, then only the diameter of the bearing plates is required.

Changing the Deflection Criteria

If you would like to change the deflection criteria of the wall form system, then you have the option of using either one or two deflection criteria. In both cases, the first deflection criteria must be used, a second deflection criteria is optional. The first deflection criteria is the span-dependent deflection limit. You have the choice of limiting the deflection to either 1/180, 1/240 or 1/360 of the span length. The second deflection criteria is an absolute limit. You have the choice of 1/8 or 1/16 of an inch total deflection in a span. Once these limits are set here, then they are used for all elements of the wall form system.

Changing the Concrete Criteria

If you would like to change the concrete criteria for the wall form system, then you have the option of either inputting the value for concrete pressure, or having WFDP compute this value based on the temperature of the concrete at the time of placement, rate of placement of concrete, and the height of the wall forms.

Displaying Stud, Wale and Form Tie Spacing

If you choose not to change any of the elements of the default wall form system, then the

maximum stud, wale and form tie spacing will be calculated and displayed for the default system in order (stud spacing, wale spacing, then form tie spacing). If changes to any of the elements are required, then WFDP will lead you through the changes and recompute the spacing of each element after all the changes to that element are complete. As each spacing value is computed and displayed, you are given the opportunity of using a smaller value for the spacing of each element, to accommodate a modular wall form system, or for any other reason. If a smaller value is input, then the spacing of each of the subsequent elements will be computed based on this smaller spacing. After the form tie spacing is displayed, then a summary of the entire wall form system is displayed, including the spacing of the stud, wale and form tie elements. At the end of this screen, you are once again given the opportunity to change any of the elements of the wall form system. If you choose not to make any changes, then WFDP exits to the command prompt. If more changes are required, then WFDP leads you through those changes and once again displays the wall form design summary.

CONCLUSION

Scheme is a very powerful programming language that can be used by engineers to solve problems of virtually any magnitude or complexity. In this instance, it has been possible to develop an object-oriented program for the purpose of designing wall forms. WFDP was created with the desirable attributes of modularity, reusability, and programming code efficiency. Although specifically applicable to wooden wall form systems, WFDP is reusable in the sense that it can easily be adapted to handle any other form design challenge such as column forms, slab forms, or form systems utilizing other materials such as aluminum and steel. WFDP is modular since the data structures are virtually independent of their external application. Additionally, WFDP can be customized to accommodate the stylistic preferences of its primary user by loading that user's preferred default wall form system. This had the distinct advantage of requiring fewer changes to the design as site adaptations are necessary.

Under normal circumstance, an experienced engineer might spend as little as 2 to 3 hours designing a system of wall forms on a construction project. On the other hand, a civil engineering student with little or no experience might spend as much as a week pouring over the myriad of details necessary to safely and yet adequately design the same wall form system. Appendix K is a illustration of the complexity of calculations required to design a typical wall form system. WFDP's quickness and reliability can easily be demonstrated when given the same design problem as that presented in Appendix K. WFDP is a tool, placed at the disposal of engineers with any level of experience, that can perform the same task in a matter of minutes, with reliable and safe results.

RECOMMENDATIONS

1. WFDP interacts with the user by screen dialog only. Much more versatility, ease of use and aesthetics could be added if a graphical interface is developed.

2. The user does not now have the capability to print the summary of each wall form system designed with WFDP. A print option would be very useful to compare several different design options developed by the user.

3. WFDP specifically designs form systems for reinforced concrete walls. This program could be easily modified and expanded to encompass all other form system applications such as slab forms, column forms and form systems made of other materials such as aluminum and steel.

4. With a little more difficulty, another module may be added to WFDP to make economic cost comparisons of alternatives such as species and grades of wood and plyform that have been selected in the design.

APPENDIX A

Scheme Code for

Wall Form Design Program

(See Attachment)

APPENDIX B

Glossary to the *USER INTERFACE* Module

This appendix is a glossary of the procedures defined in the *USER INTERFACE* module. Each procedure identifies all other procedures they invoke, as well as providing a brief description of the purpose of the procedure, any assumptions that have been made, and other clarifying remarks to aid the user in understanding the process.

START-DESIGN

- Calls WRITELN and displays an introduction to WFDP.
- Calls ASK-TO-START
- This is the main procedure to invoke the Wall Form Design Program at the Scheme Interpreter.

WRITELN

- Takes any number of arguments and displays each argument consecutively, after executing the argument (if it contains a procedure).

ASK-TO-START

- Calls WRITELN and displays a question asking the user if they would like to begin designing a wall form.
- Calls YES-NO
- Calls LOAD-INPUT

YES-NO

- Calls WRITELN and displays possible responses "Yes" or "No" to a given question.

LOAD-INPUT

- If the response to ASK-TO-START is "Yes", then the following procedures are executed.
- LOAD-DEFAULT
- LOAD-WOOD (from *WOOD ADT* module)
- GET-DEFAULT
- DESIGN (from *DESIGN CALCULATIONS* module)
- If the response to ASK-TO-START is "No", then WFDP terminates.

LOAD-DEFAULT

- Calls DEFAULT-PLY
- Calls DEFAULT-STUD
- Calls DEFAULT-WALE
- Calls DEFAULT-TIE
- Calls DEFAULT-DEF
- Calls DEFAULT-CONC

DEFAULT-PLY

- Creates the vector PLY-VEC
- Loads PLY-VEC with default plyform settings

DEFAULT-STUD

- Creates the vector STUD-VEC
- Loads STUD-VEC with default stud settings

DEFAULT-WALE

- Creates the vector WALE-VEC
- Loads WALE-VEC with default wale settings

DEFAULT-TIE

- Creates the vector TIE-VEC
- Loads TIE-VEC with default form tie settings

DEFAULT-DEF

- Creates the vector DEF-VEC
- Loads DEF-VEC with default deflection settings

DEFAULT-CONC

- Creates the vector CONC-VEC
- Loads CONC-VEC with default concrete settings

GET-DEFAULT

- Displays a summary of the default settings for the default wall form system by calling SUMMARY-TEXT
- Asks the user if they would like to change any of the default settings.
- Calls YES-NO
- If the response to GET-DEFAULT is "Yes", then the following procedure is executed.
- Calls CHANGE-DEFAULT.
- If the response to GET-DEFAULT is "No", then WFDP continues by calling DESIGN from inside the procedure LOAD-INPUT.

SUMMARY-TEXT

- For the plyform the following procedures are called
- DISP-PLY-CLASS
- DISP-PLY-DEPTH
- DISP-PLY-DIR
- DISP-F-B
- E
- F-V
- ROLLING
- For the studs the following procedures are called
- DISP-WIDTH
- DISP-DEPTH
- DISP-SPECIES
- DISP-GRADE
- DISP-F-B
- DISP-F-B-TYPE
- E
- F-C-PERP
- F-V
- For the wales the following procedures are called
- DISP-WIDTH
- DISP-DEPTH
- DISP-SPECIES
- DISP-GRADE
- DISP-SOLDIER
- DISP-F-B
- DISP-F-B-TYPE
- E
- F-C-PERP
- F-V
- For the form ties the following procedures are called
- DISP-TIE-CAP
- TIE-CAP
- DISP-TIE-SHAPE
- For the concrete the following procedures are called
- UNIF-LOAD

- For the deflection criteria the following procedures are called.
- DISP-DEF
- This is a template which inserts various values of the current wall form system in a summary format on the screen form the user to readily see.

CHANGE-DEFAULT

- Asks the user which element of the wall form system they would like to change.
- If the response is "Plyform", then GET-PLY-INFO is called.
- If the response is "Studs", then GET-STUD-INFO is called.
- If the response is "Wales", then GET-WALE-INFO is called.
- If the response is "Form Ties", then GET-TIE-INFO is called.
- If the response is "Deflection Criteria", then GET-DEF-INFO is called.
- If the response is "Concrete Criteria", then GET-CONC-INFO is called.

GET-PLY-INFO

- Calls GET-INFO-TEXT
- Calls ZERO-PLY
- If the response to GET-INFO-TEXT is "Values retrieved by computer", then the following procedures are executed.
- GET-PLY-CLASS
- GET-PLY-DIR
- GET-PLY-DEPTH
- GET-SUPCOND
- If the response to GET-INFO-TEXT is "User input values", then the following procedures are executed.
- GET-USER-PLY

GET-INFO-TEXT

- Asks the user if they would prefer that WFDP retrieve values for allowable stresses, or if they would like to enter their own values.

ZERO-PLY

- Re-initializes all PLY-VEC values to "?".

GET-PLY-CLASS

- Retrieves from the user the grade of plyform to be used in the wall form design.

GET-PLY-DIR

- Retrieves from the user the direction of the face grain of the plyform relative to the span of the studs, either "Parallel" (strong direction) or "Perpendicular" (weak direction).

GET-PLY-DEPTH

- Retrieves from the user the thickness of the plyform to be used in the wall form design.

GET-SUPCOND

- Retrieves from the user the support conditions under which the plyform will be used.
- Calls IF-PLY

IF-PLY

- If the type of wall form element is plyform, then the response to GET-SUPCOND is stored in index 3 of the PLY-VEC vector. Otherwise the response is stored in index 6 of either the STUD-VEC or WALE-VEC vector.

GET-USER-PLY

- Calls GET-PLY-CLASS
- Retrieves from the user the allowable stresses in bending, rolling shear and modulus of elasticity.

GET-STUD-INFO

- Calls GET-INFO-TEXT
- Calls ZERO
- If the response to GET-INFO-TEXT is "Values retrieved by computer", then the following procedures are executed.
- GET-SPECIES
- GET-GRADE
- GET-WIDTH
- GET-DEPTH
- GET-SIZE
- GET-FB
- GET-SUPCOND
- If the response to GET-INFO-TEXT is "User input values", then the following procedures are executed.
- GET-WIDTH
- GET-DEPTH
- GET-SUPCOND
- GET-USER-STRESS

ZERO

- Re-initializes all STUD-VEC values to "?".

GET-SPECIES

- Retrieves from the user the species of either stud or wale lumber to be used in the wall form design.

GET-GRADE

- Retrieves from the user the grade of either stud or wale lumber to be used in the wall form design.

GET-WIDTH

- Retrieves from the user the nominal width of either stud or wale lumber to be used in the wall form design.

GET-DEPTH

- Calls CHOOSE-GRADE
- If the response to CHOOSE-GRADE is "Yes", then the following procedures are executed.
- GET-GRADE
- GET-DEPTH
- If the response to CHOOSE-GRADE is "No", then only one of the following procedures are executed, depending on the nominal lumber width.
- If the lumber width is 2 inches nominally, then calls GET-DEPTH2
- If the lumber width is 3 inches nominally, then calls GET-DEPTH3
- If the lumber width is 4 inches nominally, then calls GET-DEPTH4
- If the lumber width is 5 inches nominally, then calls GET-DEPTH5
- If the lumber width is six or more inches nominally, then calls GET-DEPTH6+

CHOOSE-GRADE

- If the response to GET-GRADE is "Construction", "Standard" or "Utility" grade lumber, then this procedure either sets the lumber automatically to 4 inches nominal depth, or asks the user if they would like to change the lumber grade. These three lumber grades only come in 4 inch nominal depth.

GET-DEPTH2

- Retrieves from the user the depth of either stud or wale lumber to be used in the wall form design. Lumber with a nominal width of 2 inches only comes in depths of 3, 4, 5, 6, 8, 10 and 12 inches nominal.

GET-DEPTH3

- Retrieves from the user the depth of either stud or wale lumber to be used in the wall form design. Lumber with a nominal width of 3 inches only comes in depths of 2, 4, 5, 6, 8, 10 and 12 inches nominal.

GET-DEPTH4

- Retrieves from the user the depth of either stud or wale lumber to be used in the wall form design. Lumber with a nominal width of 4 inches only comes in depths of 2, 3, 4, 5, 6, 8, 10 and 12 inches nominal.

GET-SIZE

- Calls CHOOSE-GRADE
- Automatically sets the size classification of either the stud or wale lumber depending on the width, depth and grade of lumber being used in the wall form design.

GET-FB

- Retrieves from the user the type of bending stress, single or repetitive, to be used in the wall form design.

GET-USER-STRESS

- Retrieves from the user the allowable stresses in bending, shear, compression (perpendicular) and modulus of elasticity.

GET-WALE-INFO

- Calls GET-INFO-TEXT
- Calls ZERO
- If the response to GET-INFO-TEXT is "Values retrieved by computer", then the following procedures are executed.
- GET-SPECIES
- GET-GRADE
- GET-WIDTH
- GET-DEPTH
- GET-SIZE
- GET-WALE-SOLDIER
- GET-FB
- GET-SUPCOND
- If the response to GET-INFO-TEXT is "User input values", then the following procedures are executed.
- GET-WIDTH
- GET-DEPTH
- GET-SUPCOND
- GET-WALE-SOLDIER
- GET-USER-STRESS

GET-WALE-SOLDIER

- Retrieves from the user the wall form system uses single or double wales.

GET-TIE-INFO

- Calls ZERO-TIE
- Calls GET-TIE-CAP
- Calls GET-TIE-PSHAPE

ZERO-TIE

- Re-initializes all TIE-VEC values to "?".

GET-TIE-CAP

- Retrieves from the user the capacity of the form ties to be used in the wall form design.

GET-TIE-PSHAPE

- Retrieves from the user the shape of the form ties' bearing plates to be used in the wall form design.
- If the shape of the form ties' bearing plates are circular, then GET-TIE-PDIAM is called.
- If the shape of the form ties' bearing plates are rectangular, then GET-TIE-PWIDTH and GET-TIE-PLENGTH are called.

GET-TIE-PWIDTH

- Retrieves from the user the capacity of the form ties to be used in the wall form design.

GET-TIE-PLENGTH

- Retrieves from the user the capacity of the form ties to be used in the wall form design.

GET-TIE-PDIAM

- Retrieves from the user the capacity of the form ties to be used in the wall form design.

GET-DEF-INFO

- Re-initializes all DEF-VEC values to "?".
- Calls GET-D1
- Calls GET-D2

GET-D1

- Retrieves from the user the first deflection limit to be used in the wall form design.

GET-D2

- Retrieves from the user the second deflection limit to be used in the wall form design.

GET-CONC-INFO

- Calls ZERO-CONC
- Asks the user if they would prefer that WFDP retrieve information to compute concrete pressure, or if they would like to enter their own value.
- If the response is "Value computed by computer", then the following procedures are executed.
- GET-RATE
- GET-TEMP
- GET-HEIGHT
- If the response is "User input value", then the following procedures are executed.
- GET-USER-CONC

ZERO-CONC

- Re-initializes all DEF-VEC values to "?".

GET-RATE

- Retrieves from the user the rate of placement of concrete to be used in the wall form design.

GET-TEMP

- Retrieves from the user the temperature of the concrete to be used in the wall form design.

GET-HEIGHT

- Retrieves from the user the height of the wall forms to be used in the wall form design.

GET-USER-CONC

- Retrieves from the user the concrete pressure to be used in the wall form design.

USER-SPACE

- Displays the maximum spacing for each element of the wall form design, then asks if the user would like to use a smaller value for the spacing.
- Calls YES-NO
- If the response is "Yes", then the following procedures are executed.
- GET-REVISED SPACE
- CHECK-MAX
- If the response is "No", then WFDP continues inside the procedure START-INPUT.

GET-REVISED-SPACE

- Retrieves from the user the new value for the spacing of the particular element of the wall form system.

CHECK-MAX

- Verifies that the new spacing for the particular element of the wall form system does not exceed the maximum computed spacing.

DISP-PLY-CLASS

- Displays the plyform grade in a form the user can understand.

DISP-PLY-DEPTH

- Displays the plyform thickness in a form the user can understand.

DISP-SPECIES

- Displays the lumber species in a form the user can understand.

DISP-GRADE

- Displays the lumber grade in a form the user can understand.

DISP-WIDTH

- Displays the lumber nominal width in a form the user can understand.

DISP-DEPTH

- Displays the lumber nominal depth in a form the user can understand.

DISP-SOLDIER

- Displays whether the wale is single or double in a form the user can understand.

DISP-PLATE-DIAM

- Displays the form ties' bearing plate diameter, if circular, in a form the user can understand.

DISP-PLATE-WIDTH

- Displays the form ties' bearing plate width, if rectangular, in a form the user can understand.

DISP-PLATE-LENGTH

- Displays the form ties' bearing plate length, if rectangular, in a form the user can understand.

DISP-F-B

- Displays the allowable bending stress in a form the user can understand.

DISP-F-B-TYPE

- Displays the type of allowable bending stress (single or repetitive) in a form the user can understand.

DISP-PLY-DIR

- Displays the direction of plyform face grain relative to the stud span in a form the user can understand.

DISP-TIE-SHAPE

- Displays the form ties' bearing plate shape in a form the user can understand.

DISP-PLATE-SIZE

- Displays the dimensions of the form ties' bearing plates in a form the user can understand.

ROLLING

- Displays the word "Rolling" if the element type is plyform.

DISP-DEF

- Displays the deflection limits (first and second, if any) used the design of the wall form system in a form the user can understand.

SUMMARY

- Calls SUMMARY-TEXT
- Displays a summary of the spacing of each of the elements of the wall form system.
- Calls CHOOSE-SPACE

CHOOSE-SPACE

- If the user inputs a smaller value for the spacing of one of the elements, then that value will be displayed instead of the maximum spacing computed by WFDP.
- If the user does not input a smaller value for the spacing, then the maximum value will be displayed.

APPENDIX C

Glossary to the *DESIGN CALCULATIONS* Module

This appendix is a glossary of the procedures defined in the *DESIGN CALCULATIONS* module. Each procedure identifies all other procedures they invoke, as well as providing a brief description of the purpose of the procedure, any assumptions that have been made, and other clarifying remarks to aid the user in understanding the process

DESIGN

- Calls LOAD-WOOD
- Calls START-INPUT
- Calls SUMMARY
- This procedure uses the six Input Vectors retrieved from the *USER INTERFACE* module and creates the four wall form Element Vectors (PLYFORM, STUD, WALE and TIE). These four Element Vectors are then used to compute values for element loading and spacing, then stores these values back in the Element Vectors. Finally, all the information relative to the current design, as well as the spacing of the elements, is displayed to the user on the screen.

START-INPUT

- Calls ROUND-2-PLACES
- Calls USER-SPACE
- Calls WOOD-LOAD
- Calls WOOD-SPACING
- This procedure performs three steps for each element of the wall form system. First, the loading on the element is determined from WOOD-LOAD, then the support spacing for that element is determined from WOOD-SPACING. Finally, the user is asked if a smaller spacing is required to make the spacing modular.

ROUND-2-PLACES

- Calls ROUND-N-PLACES
- Rounds any given number to two decimal places.

ROUND-N-PLACES

- Rounds any given number to a specified number of decimal places.

WOOD-LOAD

- Calls TYPE
- Calls S-W-LOAD
- Calls CHOOSE-SPACE
- Returns the uniform loading placed on each element of the wall form system.

S-W-LOAD

- Calls CHOOSE-SPACE
- Returns the uniform loading placed on either the studs or wales.

WOOD-SPACING

- Calls TYPE
- Calls MIN-B-S-D
- Calls COMPRESSION
- Calls M-MAX
- Calls S-MAX
- Calls CAPACITY
- Returns the maximum allowable spacing of each of the elements of the wall form system.

MIN-B-S-D

- Calls BENDING
- Calls SHEAR
- Calls DEFLECTION
- Returns the smallest element spacing as governed by either bending stress, shear stress, or the deflection limit(s).

BENDING

- Calls LOOKUP2D
- Calls SUP-COND
- Calls CFDE
- Calls CHOOSE-STRESS
- Calls SM
- Calls UNIF-LOAD
- Returns the maximum allowable element spacing as governed by bending stresses in the element.

CHOOSE-STRESS

- Calls TYPE
- Calls PSI-INDEX
- In the *USER INTERFACE* module, the user has the choice of entering their own values for allowable stress for each element of the wall form system, or let WFDP retrieve these values from the tables. This procedure determines which method was used, then returns the appropriate allowable stress (Fb, E, Fc or Fv).

PSI-INDEX

- Returns the vector index location where each of the four allowable stresses (Fb, E, Fc or Fv) are stored (9, 10, 11 or 12) in the Input Vectors.

SHEAR

- Calls TYPE
- Calls LOOKUP2D
- Calls SUP-COND
- Calls CFDE
- Calls CHOOSE-STRESS
- Calls IB/Q
- Calls UNIF-LOAD
- Calls DEPTH
- Calls AREA
- Returns the maximum allowable element spacing as governed by shear stresses in the element.

DEFLECTION

- Calls DEFLECTION-LIM2
- Calls DEF-EQ1
- Calls DEF-EQ2
- Returns the maximum allowable element spacing as governed by all deflection limits of the element.

DEF1-EQ

- Calls LOOKUP2D
- Calls DEFLECTION-LIM1
- Calls SUP-COND
- Calls CFDE
- Calls CHOOSE-STRESS
- Calls MOI
- Calls UNIF-LOAD
- Returns the maximum element spacing as governed by the first deflection limit.

DEF2-EQ

- Calls LOOKUP2D
- Calls DEFLECTION-LIM2
- Calls SUP-COND
- Calls CFDE
- Calls CHOOSE-STRESS
- Calls MOI
- Calls UNIF-LOAD
- Returns the maximum allowable element spacing as governed by the second deflection limit of the element.

COMPRESSION

- Calls MODIFY-FC
- Calls CHOOSE-STRESS
- Calls BEAR-AREA
- Calls UNIF-LOAD
- Calls TYPE
- Calls CHOOSE-SPACE
- Returns the maximum allowable element spacing as governed by compressive stress between two of the elements. This procedure applies only in determining wale spacing and tie spacing.

MODIFY-FC

- Calls SOLDIER
- Calls NEW-FC
- Calls WIDTH
-

NEW-FC

- Calls DEPTH
- Calls WIDTH
- Calls FC
-

BEAR-AREA

- Calls STUD-WALE-AREA
- Calls WALE-TIE-AREA
- This procedure returns the effective bearing area between elements in the wall form system for use in the calculation of allowable compressive stresses.

STUD-WALE-AREA

- Calls WIDTH
- Calls SOLDIER
- This procedure returns the effective bearing area between stud and wale elements in the wall form system for use in the calculation of allowable compressive stresses.

WALE-TIE-AREA

- Calls SOLDIER
- Calls SPECIES
- Calls DBL-REC-AREA
- Calls DBL-CIR-AREA
- Calls SGL-REC-AREA
- Calls SGL-CIR-AREA
- This procedure returns the effective bearing area between wale and form tie elements in the wall form system for use in the calculation of allowable compressive stresses.

DBL-REC-AREA

- Calls EFF-WALE-WIDTH
- Calls WIDTH
- Calls DEPTH
- Returns the effective wale bearing area by the form ties if a double wale system and rectangular form tie bearing plates are used in the wall form design.

DBL-CIR-AREA

- Calls EFF-WALE-WIDTH
- Calls DIAMETER
- Calls SQUARE
- OVERLAP-AREA
- Returns the effective wale bearing area by the form ties if a double wale system and circular form tie bearing plates are used in the wall form design.

SQUARE

- Returns the square of a given number

SGL-REC-AREA

- Calls WIDTH
- Calls DEPTH
- Returns the effective wale bearing area by the form ties if a single wale system and rectangular form tie bearing plates are used in the wall form design.

SGL-CIR-AREA

- Calls WIDTH
- Calls DIAMETER
- Calls SQUARE
- Calls Calls OVERLAP-AREA
- Returns the effective wale bearing area by the form ties if a single wale system and circular form tie bearing plates are used in the wall form design.

EFF-WALE-WIDTH

- Calls SOLDIER
- Calls WIDTH
- Returns the effective distance from outer edge to outer edge of the wales, whether single or double wales are used in the design.

OVERLAP-AREA

- Calls C
- Calls OVERLAP-LENGTH
- Returns the area of the circular form tie bearing plate that overhangs beyond the edge of the wale.

OVERLAP-LENGTH

- Calls DIAMETER
- Calls EFF-WALE-WIDTH
- Returns the distance from the outer edge of the wale to the outer edge of the circular form tie bearing plate. This distance is used in the calculation of the overhang area.

C

- Calls DIAMETER
- Calls OVERLAP-LENGTH
- Returns the area of the circular form tie bearing plate the overhangs beyond the outer edge of the wale.

M-MAX

- Calls MOMENT
- Calls M-CLOSE-ENOUGH?
- Calls IMPROVE
- Returns the maximum allowable form tie spacing that will produce the maximum allowable bending moment in the wale element of the wall form system.

MOMENT

- Calls N-LOAD
- Calls ODD-LOAD-MOM
- Calls EVEN-LOAD-MOM
- Returns the maximum bending moment produced in the wale at a given form tie spacing.

N-LOAD

- Calls CHOOSE-SPACE
- Calls FLOAT->INTEGER
- Returns the number of point loads (studs) on the wale system between form tie supports. This number is required since the calculation of moments on the wale is different for an odd or even number of point loads.

FLOAT->INTEGER

- Returns an integer for a given floating point decimal number.

ODD-LOAD-MOM

- Calls N-LOAD
- Calls UNIF-LOAD
- Calls CHOOSE-SPACE
- Calls COEF1
- Calls SQUARE
- Returns the maximum bending moment produced in the wale at a given form tie spacing, if there are an odd number of point loads.

COEF1

- Calls SQUARE
- Calls N-LOAD
- Returns the appropriate coefficient to be used in the ODD-LOAD-MOM procedure for a given number of point loads on the wale.

EVEN-LOAD-MOM

- Calls N-LOAD
- Calls UNIF-LOAD
- Calls CHOOSE-SPACE
- Calls SQUARE
- Returns the maximum bending moment produced in the wale at a given form tie spacing, if there are an even number of point loads.

M-CLOSE-ENOUGH?

- Calls F-B
- Calls SM
- To initiate the procedure M-MAX, an initial guess for the form tie spacing of 6 inches is assumed. The moment is calculated at this spacing and is compared to the maximum allowable bending stress created at this spacing. If the allowable bending stress is not exceeded, then the form tie spacing is increased incrementally until the bending moment reaches the maximum allowable moment.

IMPROVE

- Returns an incrementally larger form tie spacing.

S-MAX

- Calls S-CLOSE-ENOUGH?
- Calls S-ACTUAL
- Calls IMPROVE
- Returns the maximum allowable form tie spacing that will produce the maximum allowable shear stress in the wale element of the wall form system.

S-CLOSE-ENOUGH?

- Calls F-V
- Calls S-ACTUAL
- To initiate the procedure S-MAX, an initial guess for the form tie spacing of 6 inches is assumed. The shear is calculated at this spacing and is compared to the maximum allowable shear stress created at this spacing. If the allowable shear stress is not exceeded, then the form tie spacing is increased incrementally until the shear reaches the maximum allowable shear.

S-ACTUAL

- Calls POINT-SHEAR
- Calls DEPTH
- Calls SOLDIER
- Calls WIDTH
- Returns the actual shear created in the wale at a given form tie spacing.

POINT-SHEAR

- Calls UNIF-LOAD
- Calls CHOOSE-SPACE
- Calls N-LOAD-S
- Calls SUM
- Calls DEPTH
- Calculates the maximum shear in the wale at a given form tie spacing.

N-LOAD-S

- Calls FLOAT->INTEGER
- Calls DEPTH
- Calls CHOOSE-SPACE
- Returns the number of point loads (studs) on the wale system between form tie supports. This number is required since the calculation of shear in the wale is different for an odd or even number of point loads.

SUM

- Returns the summation of all the integers between zero an a given number

CAPACITY

- Returns the maximum allowable tie spacing as governed by the maximum capacity of the form ties.

APPENDIX D

Glossary to the *WOOD ADT* Module

This appendix is a glossary of the procedures defined in the *WOOD ADT* module. Each procedure identifies all other procedures they invoke, as well as providing a brief description of the purpose of the procedure, any assumptions that have been made, and other clarifying remarks to aid the user in understanding the process

LOAD-WOOD

- Calls LOAD-PLYFORM
- Calls LOAD-STUD
- Calls LOAD-WALE
- Calls LOAD-TIE
- Uses the information stored in the six Input Vectors to create the four Element Vectors

LOAD-PLYFORM

- Calls MAKE-WOOD
- Calls LOOKUP4D
- Calls PLY-PROP
- Calls LOOKUP2D
- Calls PLY-STRESS
- Calls PLY-PRESS
- Uses the information stored in the Input Vector PLY-VEC, DEF-VEC and CONC-VEC to create the Element Vector PLYFORM.

MAKE-WOOD

- Creates an uninitiated Element Vector of length 24.

PLY-PRESS

- Calls W
- Calculates the uniform loading created by the concrete across a 12 inch strip of the plyform between the studs. The plyform is assumed to be acting as a simply supported beam.

W

- Calculates the lateral concrete pressure exerted on the wall forms for a given temperature of concrete, rate of placement of the concrete, and height of the wall forms.

LOAD-STUD

- Calls MAKE-WOOD
- Calls PROP-TABLE
- STRESS-TABLE
- Uses the information stored in the Input Vectors STUD-VEC, DEF-VEC and CONC-VEC to create the Element Vector STUD.

PROP-TABLE

- Calls LOOKUP2D
- Calls NOMINAL
- Calls LUM-PROP
- Returns the value of the specified lumber section property, from the data table LUM-PROP, based on the nominal dimensions of the lumber.

STRESS-TABLE

- Calls EMPTY?
- Calls LOOKUP4D
- Calls LUM-STRESS
- Returns the value of the specified lumber allowable stress, from the data table LUM-STRESS, based on the nominal dimensions of the lumber.

EMPTY?

- The user has the option of inputting his own values for allowable stress for the wall form elements, or having WFDP retrieve these values from the data tables. This procedure determines which method has been used, then allows the loading procedure to store the appropriate value in the Element Vector.

LOAD-WALE

- Calls MAKE-WOOD
- Calls PROP-TABLE
- Calls STRESS-TABLE
- Uses the information stored in the Input Vectors WALE-VEC, DEF-VEC and CONC-VEC to create the Element Vector WALE

LOAD-TIE

- Calls MAKE-WOOD

  Uses the information stored in the Input Vectors TIE-VEC, DEF-VEC and CONC-VEC to create the Element Vector TIE

TYPE

- Retrieves the wall form element type from index location 0 of the Element Vector.

WIDTH

- Not applicable for the PLYFORM vector. Retrieves the actual width of the wall form element from index location 1 of the STUD and WALE vectors. Retrieves the shortest dimension of the form tie bearing plate, if rectangular, from the TIE vector.

DEPTH

- Retrieves the actual depth of the wall form element from index location 2 of the PLYFORM, STUD and WALE vectors. Retrieves the longest dimension of the form tie bearing plate, if rectangular, from the TIE vector.

AREA

- Not applicable for the PLYFORM vector. Retrieves the cross-sectional area of the wall form element from index location 3 of the STUD and WALE vectors. Not applicable for the TIE vector.

DIAMETER

- Not applicable for the PLYFORM, STUD or WALE vectors. Retrieves the form tie bearing plate diameter, if circular, from index location 3 of the TIE vector.

MOI

- Retrieves the moment of inertia index location 4 of the Element Vector. Not applicable for the TIE vector.

SM

- Retrieves the effective section modulus from index location 5 of the Element Vector. Not applicable for the TIE vector.

SPECIES

- Not applicable for the PLYFORM vector. Retrieves the species of lumber from index location 6 of the Element Vector. For the TIE vector, the shape of the form tie bearing plate is retrieved.

GRADE

- Retrieves the grade of plyform or lumber from index location 7 of the Element Vector. Not applicable for the TIE vector.

F-B

- Calls TYPE
- Calls F-B-SING
- Calls F-B-REP
- Returns the appropriate allowable bending stress for the lumber.

F-B-SING

- Not applicable for the PLYFORM vector. Retrieves the allowable bending stress (singular) from index location 8 of the Element Vector. Not applicable for the TIE vector.

F-B-REP

- Retrieves the allowable bending stress (repetitive) from index location 9 of the Element Vector. Not applicable for the TIE vector.

F-T

- This index is not used by any of the four Element Vectors.

F-V

- Retrieves the allowable shear stress (rolling for plyform) from index location 11 of the Element Vector. Not applicable for the TIE vector.

F-C-PERP

- Not applicable for the PLYFORM vector. Retrieves the allowable compressive stress (perpendicular) from index location 12 of the Element Vector. Not applicable for the TIE vector.

E

- Retrieves the modulus of elasticity from index location 13 of the Element Vector. Not applicable for the TIE vector.

SIZE-CLASS

- Not applicable for the PLYFORM vector. Retrieves the lumber size classification from index location 14 of the Element Vector. Not applicable for the TIE vector.

CLASS

- Retrieves the plyform grade from index location 15 of the Element Vector. Not applicable for the STUD, WALE or TIE vectors.

IB/Q

- Retrieves the plyform rolling shear constant from index location 16 of the Element Vector. Not applicable for the STUD, WALE or TIE vectors.

UNIF-LOAD

- Retrieves the uniform loading of the element from index location 17 of the Element Vector.

SUP-COND

- Retrieves the support conditions of the element from index location 18 of the Element Vector. Not applicable for the TIE vector.

DEFLECTION-LIM1

- Retrieves the first deflection limit from index location 19 of the Element Vector. Not applicable for the TIE vector.

DEFLECTION-LIM2

- Retrieves the second deflection limit from index location 20 of the Element Vector. Not applicable for the TIE vector.

TIE-CAP

- Not applicable for the PLYFORM, STUD or WALE vectors. Retrieves the maximum form tie capacity from index location 21 of the Element Vector.

FACE-GRAIN

- Retrieves the direction of the face grain of the plyform relative to the stud span from index location 22 of the Element Vector. Not applicable for the STUD, WALE or TIE vectors.

SPACING

- Not applicable for the PLYFORM vector. Retrieves the element spacing from index location 23 of the Element Vector.

SOLDIER

- Not applicable for the PLYFORM, STUD or TIE vector. Retrieves whether the wale is single or double from index location 24 of the Element Vector.

APPENDIX E

Glossary to the *DATA TABLES* Module

This appendix is a glossary of the procedures defined in the *DATA TABLES* module. Each procedure identifies all other procedures they invoke, as well as providing a brief description of the purpose of the procedure, any assumptions that have been made, and other clarifying remarks to aid the user in understanding the process

LUM-PROP

- Calls MAKE-TABLE2D
- Takes the nominal lumber dimensions from the two-dimensional table NOMINAL and the description of the required lumber section property (area, moment of inertia, section modulus, width and depth) as two keys, and returns the value of the appropriate section property.

NOMINAL

- Calls MAKE-TABLE2D
- Takes the nominal lumber width and depth as two keys and returns a symbol for the combined nominal dimensions of the lumber (i.e., "2" and "4" as keys returns "s2x4"). This symbol will be used by the two-dimensional table LUM-PROP to retrieve various properties of that particular size of lumber.

LUM-STRESS

- Calls MAKE-TABLE4D
- Takes the lumber species, grade, size classification, and description of the required allowable stress (bending, shear, compression and modulus of elasticity) as four keys, and returns the value of the allowable stress.

PLY-PROP

- Calls MAKE-TABLE4D
- Takes the plyform grade, direction of face grain, thickness, and description of the required section property (moment of inertia, section modulus and rolling shear constant) as four keys, and returns the value of the section property

PLY-STRESS

- Calls MAKE-TABLE2D
- Takes the plyform grade and description of the required allowable stress (bending, rolling shear and modulus of elasticity) as two keys, and returns the value of the allowable stress.

CFDE

- Calls MAKE-TABLE2D
- Takes the design condition (bending, shear, or deflection limits) and support condition (continuous over one, two, or three or more spans) as two keys, and returns the appropriate spacing equation coinciding with those conditions.

B1

- An equation that takes as arguments the allowable bending stress, section modulus and uniform loading of a lumber section, and produces the maximum spacing as governed by bending if the member is continuous over only one span.

B2

- An equation that takes as arguments the allowable bending stress, section modulus and uniform loading of a lumber section, and produces the maximum spacing as governed by bending if the member is continuous over two spans.

B3

- An equation that takes as arguments the allowable bending stress, section modulus and uniform loading of a lumber section, and produces the maximum spacing as governed by bending if the member is continuous over three or more spans.

S1

- An equation that takes as arguments the allowable shear stress, rolling shear constant, uniform loading and member depth, and produces the maximum spacing for plyform as governed by shear if the member is continuous over only one span.

S2

- An equation that takes as arguments the allowable shear stress, rolling shear constant, uniform loading and member depth, and produces the maximum spacing for plyform as governed by shear if the member is continuous over two spans.

S3

- An equation that takes as arguments the allowable shear stress, rolling shear constant, uniform loading and member depth, and produces the maximum spacing for plyform as governed by shear if the member is continuous over three or more spans.

S4

- An equation that takes as arguments the allowable shear stress, area, uniform loading and member depth, and produces the maximum spacing for lumber as governed by shear if the member is continuous over only one span.

S5

- An equation that takes as arguments the allowable shear stress, area, uniform loading and member depth, and produces the maximum spacing for lumber as governed by shear if the member is continuous over only one span.

S6

- An equation that takes as arguments the allowable shear stress, area, uniform loading and member depth, and produces the maximum spacing for lumber as governed by shear if the member is continuous over only one span.

D1/180

- An equation that takes as arguments the modulus of elasticity, moment of inertia and uniform loading, and produces the maximum spacing as governed by deflection of no more than 1/180 of the span length if the member is continuous over only one span.

D2/180

- An equation that takes as arguments the modulus of elasticity, moment of inertia and uniform loading, and produces the maximum spacing as governed by deflection of no more than 1/180 of the span length if the member is continuous over two spans.

D3/180

- An equation that takes as arguments the modulus of elasticity, moment of inertia and uniform loading, and produces the maximum spacing as governed by deflection of no more than 1/180 of the span length if the member is continuous over three or more spans.

D1/240

- An equation that takes as arguments the modulus of elasticity, moment of inertia and uniform loading, and produces the maximum spacing as governed by deflection of no more than 1/240 of the span length if the member is continuous over only one span.

D2/240

- An equation that takes as arguments the modulus of elasticity, moment of inertia and uniform loading, and produces the maximum spacing as governed by deflection of no more than 1/240 of the span length if the member is continuous over two spans.

D3/240

- An equation that takes as arguments the modulus of elasticity, moment of inertia and uniform loading, and produces the maximum spacing as governed by deflection of no more than 1/240 of the span length if the member is continuous over three or more spans.

D1/360

- An equation that takes as arguments the modulus of elasticity, moment of inertia and uniform loading, and produces the maximum spacing as governed by deflection of no more than 1/360 of the span length if the member is continuous over only one span.

D2/360

- An equation that takes as arguments the modulus of elasticity, moment of inertia and uniform loading, and produces the maximum spacing as governed by deflection of no more than 1/360 of the span length if the member is continuous over two spans.

D3/360

- An equation that takes as arguments the modulus of elasticity, moment of inertia and uniform loading, and produces the maximum spacing as governed by deflection of no more than 1/360 of the span length if the member is continuous over three or more spans.

D125MIL

- An equation that takes as arguments the modulus of elasticity, moment of inertia and uniform loading, and produces the maximum spacing as governed by deflection of no more than 1/8 of an inch.

D0625MIL

- An equation that takes as arguments the modulus of elasticity, moment of inertia and uniform loading, and produces the maximum spacing as governed by deflection of no more than 1/16 of an inch.

APPENDIX F

Glossary to the *TABLE ADT* Module

This appendix is a glossary of the procedures defined in the *TABLE ADT* module. Each procedure identifies all other procedures they invoke, as well as providing a brief description of the purpose of the procedure, any assumptions that have been made, and other clarifying remarks to aid the user in understanding the process

MAKE-TABLE

- Creates a one dimensional table.

LOOKUP

- Calls FIND-IN-LIST
- Retrieves a value from a one dimensional table corresponding to a single input key.

FIND-IN-LIST

- Returns a record in the table containing the value associated with the applicable input key.

INSERT!

- Calls FIND-IN-LIST
- Stores a value in a one dimensional table corresponding to the input key.

MAKE-TABLE2D

- Creates a two dimensional table.

LOOKUP2D

- Calls LOOKUP
- Retrieves a value from a two dimensional table corresponding to two input keys.

INSERT2D!

- Calls LOOKUP
- Calls MAKE-TABLE
- Calls INSERT!
- Stores a value in a two dimensional table corresponding to two input keys.

MAKE-TABLE3D

- Creates a three dimensional table.

LOOKUP3D

- Calls LOOKUP
- Calls LOOKUP2D
- Retrieves a value from a three dimensional table corresponding to three input keys.

INSERT3D!

- Calls LOOKUP
- Calls MAKE-TABLE
- Calls INSERT!
- Calls INSERT2D!
- Stores a value in a three dimensional table corresponding to three input keys.

MAKE-TABLE4D

- Creates a four dimensional table.

LOOKUP4D

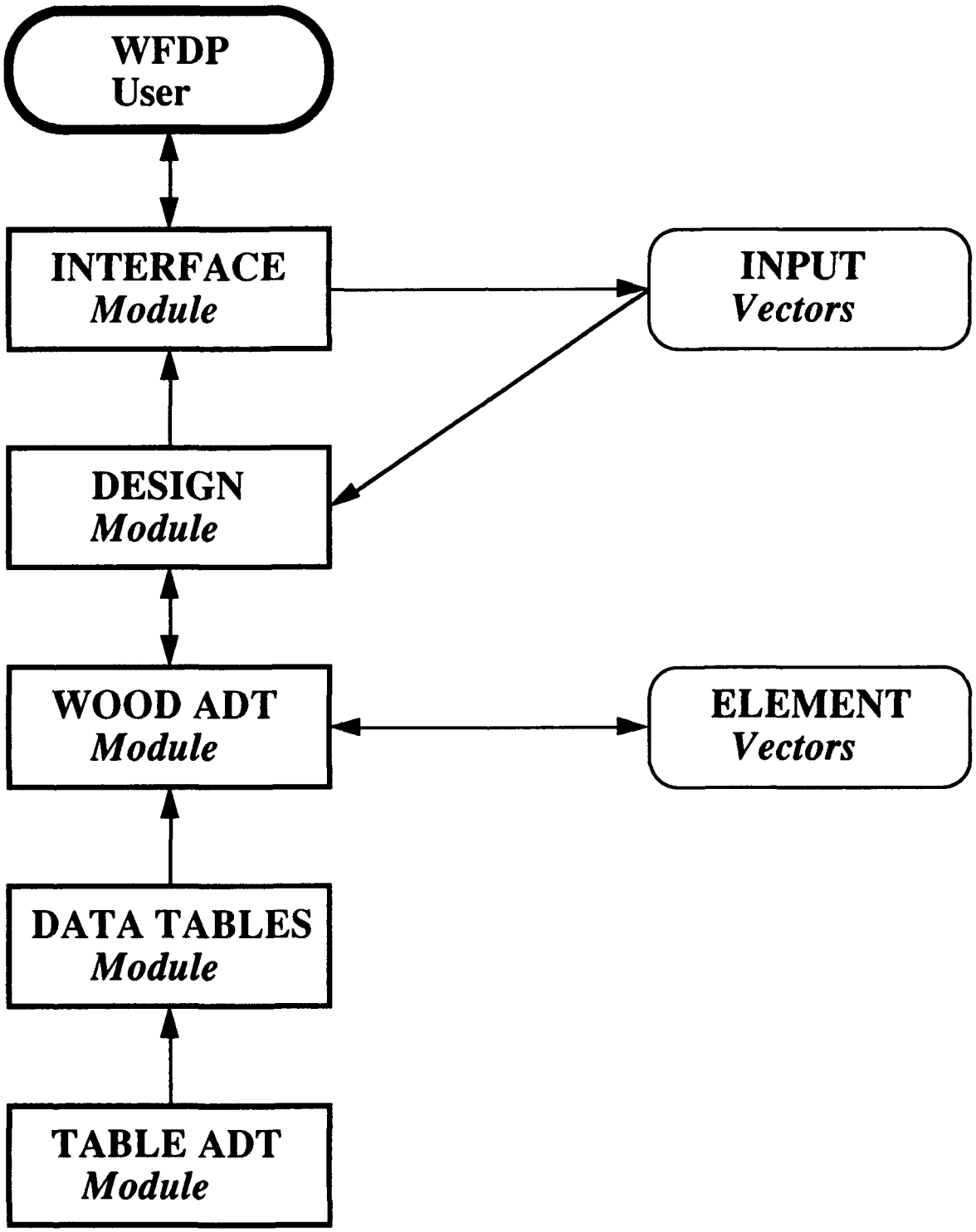
- Calls LOOKUP
- Calls LOOKUP2D
- Calls LOOKUP3D
- Retrieves a value from a four dimensional table corresponding to four input keys.

INSERT4D!

- Calls LOOKUP
- Calls LOOKUP2D
- Calls LOOKUP3D
- Calls MAKE-TABLE
- Calls INSERT!
- Calls INSERT2D!
- Calls INSERT3D!
- Stores a value in a four dimensional table corresponding to four input keys.

APPENDIX G

Flow chart of the global architecture showing the relationship between the WFDP user, five program modules, six Input Vectors, and four Element Vectors.



APPENDIX H

This appendix defines the contents of the indices of each of the six Input Vectors. The Input Vectors are created when the default wall form system is loaded, then, as changes to the system are required, the new information is inserted into the appropriate Input Vector indices by the *USER INTERFACE* Module. These vectors are then used by the *WOOD ADT* Module to create the four Element Vectors.

PLY-VEC vector

Index	Description of Contents
0	Plyform grade
1	Direction of face grain relative to span
2	Thickness
3	Support conditions
4	User specified allowable bending stress
5	User specified modulus of elasticity
6	User specified rolling shear stress
7	User specified rolling shear constant

STUD-VEC vector

Index	Description of Contents
0	User specified stud spacing
1	Species of lumber
2	Grade of lumber
3	Size classification
4	Nominal lumber width
5	Nominal lumber depth
6	Support conditions
7	Allowable bending stress type (single or repetitive)
8	Unused
9	User specified allowable bending stress
10	User specified modulus of elasticity
11	User specified allowable compressive stress (perp.)
12	User specified allowable shear stress

WALE-VEC vector

Index	Description of Contents
0	User specified wale spacing
1	Species of lumber
2	Grade of lumber
3	Size classification
4	Nominal lumber width
5	Nominal lumber depth
6	Support conditions
7	Allowable bending stress type (single or repetitive)
8	Type of wale system (single or double)
9	User specified allowable bending stress
10	User specified modulus of elasticity
11	User specified allowable compressive stress (perp.)
12	User specified allowable shear stress

TIE-VEC vector

Index	Description of Contents
0	User specified form tie spacing
1	Tie capacity
2	Shape of form tie bearing plate (circular or rectangular)
3	Bearing plate width, if rectangular (shortest dimension)
4	Bearing plate length, if rectangular (longest dimension)
5	Bearing plate diameter, if circular

DEF-VEC vector

Index	Description of Contents
0	First deflection limit (span-dependent)
1	Second deflection limit (absolute)

CONC-VEC vector

Index	Description of Contents
0	Rate of placement of concrete
1	Temperature of concrete
2	Height of wall form
3	User specified concrete pressure

APPENDIX I

This appendix describes all the criteria required to design the default wall form system. This information is loaded into the Input Vectors when WFDP is invoked.

Plyforms

Class II grade
3/4 inch thick
1330 psi allowable bending stress (repetitive)
72 psi allowable rolling shear stress
1,430,000 psi modulus of elasticity
Continuous over 3 or more supports
Direction of face grain parallel (strong direction) to span

Studs

2 x 6 inch lumber
Lumber species not specified
Lumber grade not specified
1450 psi allowable bending stress (repetitive)
185 psi allowable shear stress
385 psi allowable compressive stress (perpendicular)
1,700,000 psi modulus of elasticity
Continuous over 3 or more supports

Wales

2 x 6 inch lumber
Lumber species not specified
Lumber grade not specified
1450 psi allowable bending stress (repetitive)
185 psi allowable shear stress
185 psi allowable compressive stress (perpendicular)
1,700,000 psi modulus of elasticity
Continuous over 3 or more supports

Form Ties

4000 pounds maximum capacity
4 x 4 inch bearing plates

Deflection Criteria

Not to exceed 1/360 of the member spacing

Not to exceed 1/8 inch total deflection

Concrete Criteria

1000 pounds per square foot of pressure on plyform (maximum)

APPENDIX J

This appendix defines the contents of the indices of each of the four Element Vectors. The Element Vectors are created by the *WOOD ADT* Module, after the default wall form system has been loaded into the Input Vectors, in order that the *DESIGN CALCULATIONS* Module can determine the appropriate element spacing.

PLYFORM vector

Index	Description of Contents
0	Type of wall form element (plyform)
1	N/A
2	Thickness
3	N/A
4	Moment of inertia
5	Effective section modulus
6	N/A
7	Grade of plyform
8	N/A
9	Allowable bending stress
10	N/A
11	Allowable rolling shear stress
12	N/A
13	Modulus of elasticity
14	N/A
15	Plyform grade
16	Rolling shear constant
17	Plyform loading
18	Support conditions
19	First deflection limit (span-dependent)
20	Second deflection limit (absolute)
21	N/A
22	Direction of face grain relative to span
23	N/A
24	N/A

STUD vector

Index	Description of Contents
0	Type of wall form element (stud)
1	Actual width
2	Actual depth
3	Cross-sectional area
4	Moment of inertia
5	Effective section modulus
6	Species of lumber
7	Grade of lumber
8	Allowable bending stress (single)
9	Allowable bending stress (repetitive)
10	N/A
11	Allowable shear stress
12	Allowable compressive stress (perpendicular)
13	Modulus of elasticity
14	Lumber size classification
15	N/A
16	N/A
17	Stud loading
18	Support conditions
19	First deflection limit (span-dependent)
20	Second deflection limit (absolute)
21	N/A
22	N/A
23	Stud spacing
24	N/A

WALE vector

Index	Description of Contents
0	Type of wall form element (wale)
1	Actual width
2	Actual depth
3	Cross-sectional area
4	Moment of inertia
5	Effective section modulus
6	Species of lumber
7	Grade of lumber
8	Allowable bending stress (single)
9	Allowable bending stress (repetitive)
10	N/A
11	Allowable shear stress
12	Allowable compressive stress (perpendicular)
13	Modulus of elasticity
14	Lumber size classification
15	N/A
16	N/A
17	Wale loading
18	Support conditions
19	First deflection limit (span-dependent)
20	Second deflection limit (absolute)
21	N/A
22	N/A
23	Wale spacing
24	Single or double wale

TIE vector

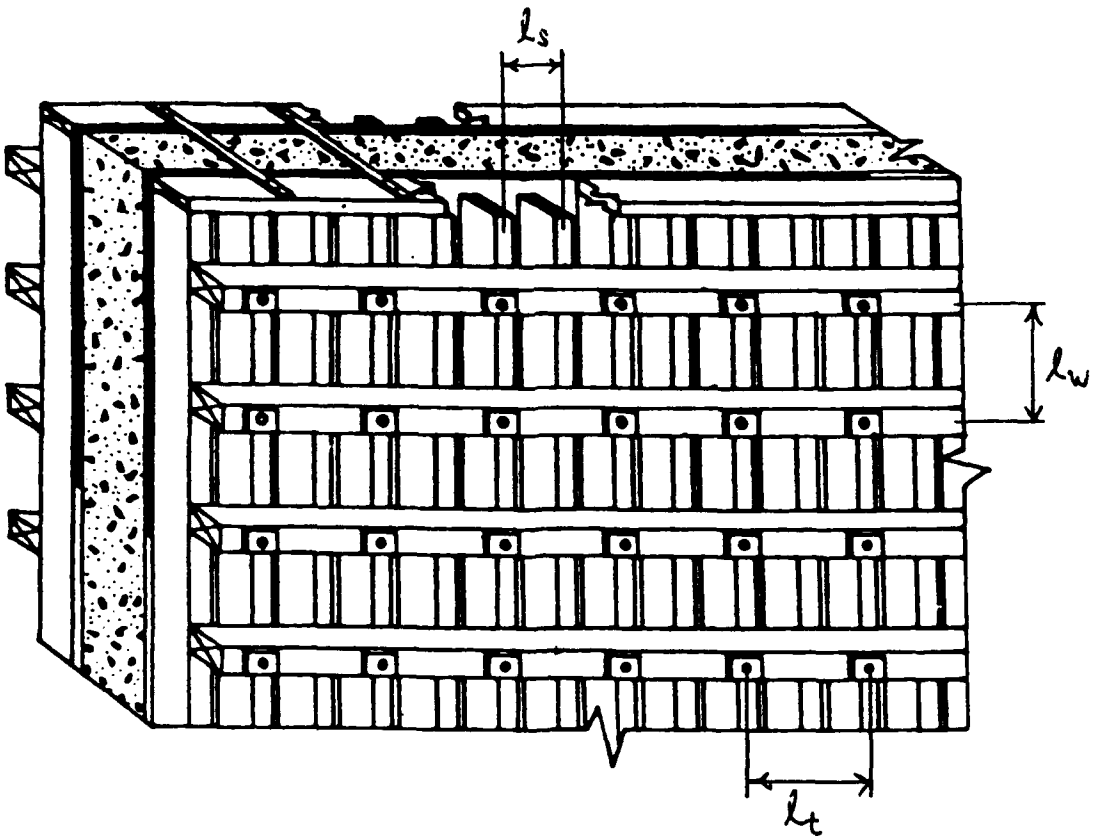
Index	Description of Contents
0	Type of wall form element (tie)
1	Shortest dimension of tie plate, if rectangular
2	Longest dimension of tie plate, if rectangular
3	Bearing plate diameter, if circular
4	N/A
5	N/A
6	Shape of tie plate or washer
7	N/A
8	N/A
9	N/A
10	N/A
11	N/A
12	N/A
13	N/A
14	N/A
15	N/A
16	N/A
17	Tie loading
18	N/A
19	N/A
20	N/A
21	Tie capacity
22	N/A
23	Spacing
24	N/A

APPENDIX K

Graphical Illustration of an Example Wall Form Design

This appendix show a typical Wall Form System including the positioning of the plyform, studs, wales and form ties. Also shown is the appropriate spacing dimension for each element as computed by WFDP. The following variables are defined and shown here for use in Appendix L.

l_s = spacing of the stud element
l_w = spacing of the wale element
l_t = spacing of the form tie element



APPENDIX L

Example Wall Form Design Problem

Design a wall form under the following conditions:

Concrete:

Temperature at time of placement:	80 degrees Fahrenheit
Rate of placement of concrete in forms:	6 feet per hour
Height of wall forms:	10 feet

Deflection limits:

First deflection limit:	1/360 of the span length
Second deflection limit:	1/8 inch total

Plyform:

Grade:	Class I
Thickness:	7/8 inch
Allowable bending stress:	1930 psi
Allowable rolling shear stress:	72 psi
Modulus of elasticity:	1,650,000 psi

Continuous over 3 or more spans
Direction of face grain placed parallel (strong direction) to stud spans

Stud lumber:

Species and grade not specified

Allowable bending stress (repetitive):	1,500 psi
Allowable shear stress:	220 psi
Allowable compressive stress (perp.):	340 psi
Modulus of elasticity:	1,550,000 psi

Continuous over 3 or more spans

Wales:

Species and grade not specified

Allowable bending stress (repetitive):	1,500 psi
Allowable shear stress:	220 psi
Allowable compressive stress (perp.):	340 psi
Modulus of elasticity:	1,550,000 psi

Continuous over 3 or more spans
Double wales

Form Ties:

Maximum capacity:	3000 lbs.

Burke form ties with 2 1/2 inch by 4 inch bearing plates

I. Determine the stud spacing from the plyform properties:

a. Plyform section properties

Section modulus:	0.553 in^3/ft
Moment of inertia:	0.280 in^4/ft
Rolling shear constant:	8.668 in^2/ft

b. Concrete pressure

$$p = 150 + 9000 \frac{R}{T}$$

$$p = 150 + 9000 \left(\frac{6}{80}\right)$$

$$p = 825 \text{ lb/ft}^2$$

c. Loading on plyform

$$w_p = p \left(\frac{12 \text{ in}}{12 \text{ in/ft}}\right)$$

$$w_p = 825 \text{ lb/ft}^2 \left(\frac{12 \text{ in}}{12 \text{ in/ft}}\right)$$

$$w_p = 825 \text{ lb/lft}$$

d. Stud spacing as governed by plyform bending moment

$$l_s = 10.95 \sqrt{\frac{f_b S}{w}}$$

$$l_s = 10.95 \sqrt{\frac{1930 \times 0.553}{825}}$$

$$l_s = 12.45 \text{ in.}$$

e. Stud spacing as governed by plyform deflection

1. Deflection limited to 1/360 of stud span

$$l_s = 1.69 \sqrt[3]{\frac{EI}{w}}$$

$$l_s = 1.69 \sqrt[3]{\frac{1.64 \times 10^6 \times 0.280}{825}}$$

$$l_s = 13.93 \text{ in.}$$

2. Deflection limited to 1/8 inch

$$l_s = 3.84 \sqrt[4]{\frac{EI}{w}}$$

$$l_s = 3.84 \sqrt[4]{\frac{1.65 \times 10^6 \times 0.280}{825}}$$

$$l_s = 18.68 \text{ in.}$$

f. Stud spacing as governed by plyform shear

$$l_s = \frac{20 f_v (\frac{Ib}{Q})}{w} + 2h$$

$$l_s = \frac{(20 \times 72 \times 8.668)}{825} + (2 \times 0.875)$$

$$l_s = 16.88 \text{ in.}$$

g. Bending governs and $l_s = 12.45$ in., but use $l_s = 12$ in.

II. Determine wale spacing from the stud properties:

a. Stud section properties

Section modulus:	3.06 in^3
Moment of inertia:	5.36 in^4
Area:	5.25 in^2

b. Loading on studs

$$w_s = p \left(\frac{l_s}{12 \text{ in/ft}}\right)$$

$$w_s = 825 \text{ lb/ft}^2 \left(\frac{12 \text{ in}}{12 \text{ in/ft}}\right)$$

$$w_s = 825 \text{ lb/lft}$$

c. Wale spacing as governed by stud bending moment

$$l_w = 10.95 \sqrt{\frac{f_b S}{w}}$$

$$l_w = 10.95 \sqrt{\frac{1500 \times 3.06}{825}}$$

$$l_w = 25.83 \text{ in.}$$

d. Wale spacing as governed by stud deflection

1. Deflection limited to 1/360 of wale span

$$l_w = 1.69 \sqrt[3]{\frac{EI}{w}}$$

$$l_w = 1.69 \sqrt[3]{\frac{1.55 \times 10^6 \times 5.36}{825}}$$

$$l_w = 36.50 \text{ in.}$$

2. Deflection limited to 1/8 inch

$$l_w = 3.84 \sqrt[4]{\frac{EI}{w}}$$

$$l_w = 3.84 \sqrt[4]{\frac{1.55 \times 10^6 \times 5.36}{825}}$$

$$l_w = 38.47 \text{ in.}$$

e. Wale spacing as governed by stud shear

$$l_w = \frac{13.33\ Af_v}{w} + 2\ h$$

$$l_w = \frac{(13.33 \times 5.25 \times 220)}{825} + (2 \times 3.5)$$

$$l_w = 25.66 \text{ in.}$$

f. Wale spacing as governed by stud-wale compressive stresses.

The allowable compressive stress of 340 psi can be modified by a factor of $\frac{l_b + 0.375}{l_b}$ since,

1. The bearing length (l_b) of 2 (1.5) = 3 inches is less than 6 inches, and
2. The point of bearing is more than 3 inches from the end of the member.

So, $$f'_c = 340 \frac{(3 + 0.375)}{3}$$

$$f'_c = 382.50 \text{ psi}$$

$$l_w = \frac{f'_c A_c}{w_s(1ft/12in)}$$

$$l_w = \frac{382.50 \times (2 \times 1.5 \times 1.5)}{68.75}$$

$$l_w = 25.04 \text{ in.}$$

g. Compression governs and $l_w = 25.04$ in., but use $l_w = 24$ in.

III. Determine form tie spacing from the wale properties:

a. Wale section properties

Section modulus:	6.12 in^3
Moment of inertia:	10.72 in^4
Area:	10.50 in^2

b. Loading on wales

$$w_w = p\left(\frac{l_w}{12in/ft}\right)$$

$$w_w = 825\ lb/ft^2\left(\frac{24\ in}{12in/ft}\right)$$

$$w_w = 1650\ lb/lft$$

c. Form tie spacing as governed by wale bending moment

$$l_t = 10.95\sqrt{\frac{f_b S}{w}}$$

$$l_t = 10.95\sqrt{\frac{1500 \times 6.12}{1650}}$$

$$l_t = 25.83\ in.$$

d. Form tie spacing as governed by wale deflection

1. Deflection limited to 1/360 of form tie span

$$l_t = 1.69\sqrt[3]{\frac{EI}{w}}$$

$$l_t = 1.69\sqrt[3]{\frac{1.55 \times 10^6 \times 10.72}{1650}}$$

$$l_t = 36.50\ in.$$

2. Deflection limited to 1/8 inch

$$l_t = 3.84\sqrt[4]{\frac{EI}{w}}$$

$$l_t = 3.84\sqrt[4]{\frac{1.55 \times 10^6 \times 10.72}{1650}}$$

$$l_t = 38.47\ in.$$

e. Form tie spacing as governed by wale shear

$$l_t = \frac{13.33\ A\ f_v}{w} + 2\ h$$

$$l_t = \frac{(13.33 \times (2 \times 5.25) \times 220)}{1650} + (2 \times 3.5)$$

$$l_t = 25.67 \text{ in.}$$

f. Form tie spacing as governed by wale-tie compressive stresses.

$$l_t = \frac{f'_c A_c}{\frac{w_p}{(12 \text{ in/ft})^2} l_w}$$

$$l_t = \frac{382.50 \times (2 \times 2.5 \times 1.5)}{\frac{825}{144} \times 24}$$

$$l_t = 20.86 \text{ in.}$$

g. Form tie spacing as governed by maximum moment from point loads (studs) on the wales.

From the previous calculations, the governing spacing is from compression at 20.86 inches. Use 16 inch spacing to check for maximum moment by point loads. At a form tie spacing of 16 inches, the maximum moment occurs with three studs placed symmetrically across the wale between form tie supports. The maximum allowable spacing can be computed based on an assumed spacing of 16 inches. If the maximum spacing is greater than 16 inches, then the assumption is valid, if not, then a smaller value of spacing will have to be tried.

$$l_t = \frac{4 f_b S (12 \text{ in/ft})^2}{w_p l_s l_w}$$

$$l_t = \frac{4 \times 1500 \times 6.126 \times (12 \text{ in/ft})^2}{825 \times 12 \times 24}$$

$$l_t = 22.28 \text{ in.} > 16 \text{ in.} \quad \text{OK!}$$

h. Form tie spacing as governed by maximum shear from point loads (studs) on the wales.

The maximum shear on the wales when a stud is positioned 3.5 inches from one of the form tie supports. One again the maximum allowable spacing can be computed based on an assumed spacing of 16 inches. If the maximum spacing is greater than 16 inches, then the assumption is valid, if not, then a smaller value of spacing will have to be tried.

$$l_t = \frac{(l_s + 2h)}{(2 - (\frac{4Af_v(12\ in/ft)^2}{3w_p l_s l_w}))}$$

$$l_t = \frac{(12 + (2 \times 3.5))}{(2 - (\frac{4 \times (1.5 \times 3.5) \times 220 \times (12\ in/ft)^2}{3 \times 825 \times 12 \times 24}))}$$

$$l_t = 17.81\ in.$$

i. Form tie spacing as governed by maximum capacity of the form tie.

$$l_t = \frac{T_{tie}(12\ in/ft)^2}{w_p l_w}$$

$$l_t = \frac{3000 \times (12\ in/ft)^2}{825 \times 24}$$

$$l_t = 21.82\ in.$$

j. Maximum shear forces created by point loads (studs) governs the form tie spacing and $l_t = 17.81$ in., but use $l_t = 16$ in.

IV. Final Design Summary

a. 3/4 inch Class I plyform.

b. 2 by 4 inch studs spaced 12 inches on center.

c. Double 2 by 4 inch wales spaced 24 inches on center.

d. 3000 lb capacity form ties spaced 16 inches apart along the wales.

```
(module iface (with table wood stress cfde prop default design) (main start-design))

;;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;;  User Interface Procedures Defined
;;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

(define start-design
  (lambda ()
    (newline)
    (writeln "********************")
    (newline)
    (writeln "You have called the Wall Form Design Program.  This program will")
    (writeln "assist you in designing formwork for cast-in-place reinforced")
    (writeln "concrete wall systems.")
    (newline)
    (ask-to-start)))

(define yes-no
  (lambda ()
    (newline)
    (writeln "a.  Yes")
    (writeln "b.  No")
    (newline)))

(define ask-to-start
  (lambda ()
    (writeln "Would you like to begin designing a wall form?")
    (yes-no)
    (let ((answer (read)))
      (load-input answer))))

(define summary
  (lambda (ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
    (newline)
    (writeln "********************")
    (writeln "********************")
    (newline)
    (writeln "FINAL DESIGN SUMMARY")
    (newline)
    (summary-text ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
    (writeln "  The STUD spacing is:  " (choose-space stud stud-vec) " inches.")
    (writeln "  The WALE spacing is:  " (choose-space wale wale-vec) " inches.")
    (writeln "  The  TIE spacing is:  " (choose-space tie tie-vec) " inches.")
    (newline)
    (writeln "********************")
    (writeln "********************")
    (newline)
    (writeln "Would you like to change any of the elements in the current wall form")
    (writeln "system?")
    (yes-no)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! stud-vec 0 '?)
             (vector-set! wale-vec 0 '?)
             (vector-set! tie-vec 0 '?)
             (change-default ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
             (design ply-vec stud-vec wale-vec tie-vec def-vec conc-vec))
            ((equal? answer 'b)
             (writeln "Thank you for using the Wall Form Design Program.  Good bye."))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-default ply-vec stud-vec wale-vec tie-vec def-vec conc-vec))))))

(define summary-text
  (lambda (ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
    (writeln "  PLYFORM")
```

```
    (writeln "    - " (disp-ply-class ply-vec)) " grade"
    (writeln "    - " (disp-ply-depth ply-vec) " thick")
    (writeln "    - Face grain running " (disp-ply-dir ply-vec) " to stud span")
    (writeln "    - " (disp-f-b ply-vec plyform) " psi allowable bending stress")
    (writeln "    - " (e plyform) " psi modulus of elasticity")
    (writeln "    - " (f-v plyform) " psi allowable " (rolling plyform) "shear stress")
    (writeln "  STUDS")
    (writeln "    - " (disp-width stud-vec) " x " (disp-depth stud-vec) " inch lumber")
    (writeln "    - " (disp-species stud-vec))
    (writeln "    - " (disp-grade stud-vec))
    (writeln "    - " (disp-f-b stud-vec stud) " psi allowable bending stress" " (" (disp-
f-b-type stud-vec) ")")
    (writeln "    - " (e stud) " psi modulus of elasticity")
    (writeln "    - " (f-c-perp stud) " psi allowable compressive stress (Perpendicular)")
    (writeln "    - " (f-v stud) " psi allowable shear stress")
    (writeln "  WALES")
    (writeln "    - " (disp-width wale-vec) " x " (disp-depth wale-vec) " inch lumber")
    (writeln "    - " (disp-species wale-vec))
    (writeln "    - " (disp-grade wale-vec))
    (writeln "    - " (disp-soldier wale-vec) " waler")
    (writeln "    - " (disp-f-b wale-vec wale) " psi allowable bending stress" " (" (disp-
f-b-type stud-vec) ")")
    (writeln "    - " (e wale) " psi modulus of elasticity")
    (writeln "    - " (f-c-perp wale) " psi allowable compressive stress (Perpendicular)")
    (writeln "    - " (f-v wale) " psi allowable shear stress")
    (writeln "  TIES")
    (writeln "    - " (tie-cap tie) " pounds maximum capacity")
    (writeln "    - " (disp-tie-shape tie-vec) " bearing plates")
    (disp-plate-size tie-vec)
    (writeln "  CONCRETE pressure")
    (writeln "    - " (unif-load plyform) " pounds per square feet.")
    (writeln "  DEFLECTION limits")
    (disp-def def-vec)
    (newline)))

(define choose-space
  (lambda (wood input)
    (if (equal? (vector-ref input 0) '?)
        (spacing wood)
        (vector-ref input 0))))

(define load-input
  (lambda (answer)
    (cond ((equal? answer 'a)
           (load-default)
           (load-wood ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
           (get-default ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
           (design ply-vec stud-vec wale-vec tie-vec def-vec conc-vec))
          ((equal? answer 'b)
           (newline)
           (writeln "Thank you for using the Wall Form Design Program.  Good bye.")
           (newline))
          (else
           (newline)
           (writeln "Your entry is not defined, please try again.")
           (newline)
           (writeln "********************")
           (newline)
           (ask-to-start)))))

(define get-default
  (lambda (ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
    (newline)
    (writeln "********************")
    (writeln "********************")
    (newline)
```

```
    (writeln "The default Wall Form System has been loaded with its elements")
    (writeln "described as follows:")
    (newline)
    (summary-text ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
    (writeln "********************")
    (newline)
    (writeln "Would you like to change any of this information?")
    (yes-no)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (change-default ply-vec stud-vec wale-vec tie-vec def-vec conc-vec))
            ((equal? answer 'b))
            (else
             (writeln "Your entry is not defined, please try again.")
             (newline)
             (get-default ply-vec stud-vec wale-vec tie-vec def-vec conc-vec))))))

(define change-default
  (lambda (ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "Which element of the Wall Form System would you like to change?")
    (newline)
    (writeln "a.  Plyform")
    (writeln "b.  Studs")
    (writeln "c.  Wales")
    (writeln "d.  Form Ties")
    (writeln "e.  Deflection Criteria")
    (writeln "f.  Concrete Criteria")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (get-ply-info plyform ply-vec))
            ((equal? answer 'b)
             (get-stud-info stud stud-vec))
            ((equal? answer 'c)
             (get-wale-info wale wale-vec))
            ((equal? answer 'd)
             (get-tie-info tie-vec))
            ((equal? answer 'e)
             (get-def-info def-vec))
            (else
             (get-conc-info conc-vec))))))

(define get-info-text
  (lambda (wood input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "Would you like for the Wall Form Design Program to retrieve values")
    (writeln "for allowable " (type wood) " stresses, or would you like to enter your")
    (writeln "own values?")
    (newline)
    (writeln "a.  Values retrieved by computer.")
    (writeln "b.  User input values.")
    (newline)))

(define get-ply-info
  (lambda (wood input)
    (get-info-text wood input)
    (zero-ply input)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (get-ply-class input)
             (get-ply-dir input)
```

```
              (get-ply-depth input)
              (get-supcond wood input))
             ((equal? answer 'b)
              (get-user-ply input)
              (get-ply-dir input)
              (get-ply-depth input)
              (get-supcond wood input))
             (else
              (writeln "Your entry is not defined, please try again.")
              (get-ply-info wood input))))))

(define zero-ply
  (lambda (input)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)))

(define get-user-ply
  (lambda (input)
    (newline)
    (get-ply-class input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "Plyform allowable bending stress, in psi:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 4 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-user-ply input))))
    (newline)
    (writeln "Plyform modulus of elasticity, in psi:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 5 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-user-ply input))))
    (newline)
    (writeln "Plyform allowable rolling shear stress, in psi:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 6 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-user-ply input))))))

(define get-ply-class
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What grade of plyform is being used?")
    (newline)
    (writeln "Note:  All plyform properties retrieved from tables in this program have bee
n")
    (writeln "adjusted to account for the reduced effectiveness of plies with their grain"
)
    (writeln "perpendicular to the applied stress.")
    (newline)
```

```
    (writeln "a.  Class I")
    (writeln "b.  Class II")
    (writeln "c.  Structural I")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 0 'c1))
            ((equal? answer 'b)
             (vector-set! input 0 'c2))
            ((equal? answer 'c)
             (vector-set! input 0 's1))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-ply-class input))))))

(define get-ply-dir
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "Is the face grain of the plyform running parallel or perpendicular")
    (writeln "to the span of the studs?")
    (newline)
    (writeln "a.  Parallel (strong direction)")
    (writeln "b.  Perpendicular")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 1 'para))
            ((equal? answer 'b)
             (vector-set! input 1 'perp))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-ply-dir input))))))

(define get-ply-depth
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the thickness of the plyform?")
    (newline)
    (writeln "a.  15/32 inch")
    (writeln "b.  1/2 inch")
    (writeln "c.  19/32 inch")
    (writeln "d.  5/8 inch")
    (writeln "e.  23/32 inch")
    (writeln "f.  3/4 inch")
    (writeln "g.  7/8 inch")
    (writeln "h.  1 inch")
    (writeln "i.  1 1/4 inch")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 2 0.46875))
            ((equal? answer 'b)
             (vector-set! input 2 0.5))
            ((equal? answer 'c)
             (vector-set! input 2 0.59375))
            ((equal? answer 'd)
             (vector-set! input 2 0.625))
            ((equal? answer 'e)
             (vector-set! input 2 0.71875))
            ((equal? answer 'f)
             (vector-set! input 2 0.75))
            ((equal? answer 'g)
```

```
                (vector-set! input 2 0.875))
               ((equal? answer 'h)
                (vector-set! input 2 1.0))
               ((equal? answer 'i)
                (vector-set! input 2 1.125))
               (else
                (writeln "Your entry is not defined, please try again.")
                (get-ply-depth input))))))

(define get-stud-info
  (lambda (wood input)
    (get-info-text wood input)
    (zero input)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (get-species wood input)
             (get-grade wood input)
             (get-width wood input)
             (get-depth wood input)
             (get-size input)
             (get-fb wood input)
             (get-supcond wood input))
            ((equal? answer 'b)
             (get-width wood input)
             (get-depth wood input)
             (get-supcond wood input)
             (get-user-stress wood input))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-stud-info wood input))))))

(define zero
  (lambda (input)
    (vector-set! input 0 '?)
    (vector-set! input 1 '?)
    (vector-set! input 2 '?)
    (vector-set! input 3 '?)
    (vector-set! input 4 '?)
    (vector-set! input 5 '?)
    (vector-set! input 6 '?)
    (vector-set! input 7 '?)
    (vector-set! input 8 '?)
    (vector-set! input 9 '?)
    (vector-set! input 10 '?)
    (vector-set! input 11 '?)
    (vector-set! input 12 '?)))

(define get-user-stress
  (lambda (wood input)
    (newline)
    (writeln (type wood) " lumber allowable bending stress, in psi:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 9 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-user-stress wood input))))
    (newline)
    (writeln (type wood) " lumber modulus of elasticity, in psi:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 10 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-user-stress wood input))))
    (newline)
```

```
    (writeln (type wood) " lumber allowable compressive stress (perpendicular), in psi:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 11 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-user-stress wood input))))
    (newline)
    (writeln (type wood) " lumber allowable shear stress, in psi:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 12 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-user-stress wood input))))))

(define get-species
  (lambda (wood input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the species of lumber for the " (type wood) "s?")
    (newline)
    (writeln "a.  Aspen")
    (writeln "b.  Balsam Fir")
    (writeln "c.  Douglas Fir-Larch")
    (writeln "d.  Douglas Fir South")
    (writeln "e.  Eastern Hemlock")
    (writeln "f.  Eastern Hemlock-Tamarack")
    (writeln "g.  Eastern Spruce")
    (writeln "h.  Eastern White Pine")
    (writeln "i.  Eastern Woods")
    (writeln "j.  Engelmann Spruce-Alpine Fir (Engelmann Spruce-Lodgepole Pine)")
    (writeln "k.  Hem-Fir")
    (writeln "l.  Idaho White Pine")
    (writeln "m.  Lodgepole Pine")
    (writeln "n.  Mountain Hemlock")
    (writeln "o.  Northern Aspen")
    (writeln "p.  Northern Pine")
    (writeln "q.  Northern White Cedar")
    (writeln "r.  Ponderosa Pine-Sugar Pine (Ponderosa Pine-Lodgepole Pine)")
    (writeln "s.  Sitka Spruce")
    (writeln "t.  Southern Pine")
    (writeln "u.  Western Cedars")
    (writeln "v.  Western Hemlock")
    (writeln "w.  White Woods (Western Woods)")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 1 'a))
            ((equal? answer 'b)
             (vector-set! input 1 'bf))
            ((equal? answer 'c)
             (vector-set! input 1 'dfl))
            ((equal? answer 'd)
             (vector-set! input 1 'dfs))
            ((equal? answer 'e)
             (vector-set! input 1 'eh))
            ((equal? answer 'f)
             (vector-set! input 1 'eht))
            ((equal? answer 'g)
             (vector-set! input 1 'es))
            ((equal? answer 'h)
             (vector-set! input 1 'ewp))
            ((equal? answer 'i)
             (vector-set! input 1 'ew))
```

```
            ((equal? answer 'j)
             (vector-set! input 1 'esaf))
            ((equal? answer 'k)
             (vector-set! input 1 'hf))
            ((equal? answer 'l)
             (vector-set! input 1 'iwp))
            ((equal? answer 'm)
             (vector-set! input 1 'lp))
            ((equal? answer 'n)
             (vector-set! input 1 'mh))
            ((equal? answer 'o)
             (vector-set! input 1 'na))
            ((equal? answer 'p)
             (vector-set! input 1 'np))
            ((equal? answer 'q)
             (vector-set! input 1 'nwc))
            ((equal? answer 'r)
             (vector-set! input 1 'ppsp))
            ((equal? answer 's)
             (vector-set! input 1 'ss))
            ((equal? answer 't)
             (vector-set! input 1 'sp))
            ((equal? answer 'u)
             (vector-set! input 1 'wc))
            ((equal? answer 'v)
             (vector-set! input 1 'wh))
            ((equal? answer 'w)
             (vector-set! input 1 'ww))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-species wood input))))))

(define get-grade
  (lambda (wood input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the grade of lumber for the " (type wood) "s?")
    (newline)
    (writeln "a.  Select Structural")
    (writeln "b.  No. 1")
    (writeln "c.  No. 2")
    (writeln "d.  No. 3")
    (writeln "e.  Appearance")
    (writeln "f.  Stud")
    (writeln "g.  Construction")
    (writeln "h.  Standard")
    (writeln "i.  Utility")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 2 'ss))
            ((equal? answer 'b)
             (vector-set! input 2 'n1))
            ((equal? answer 'c)
             (vector-set! input 2 'n2))
            ((equal? answer 'd)
             (vector-set! input 2 'n3))
            ((equal? answer 'e)
             (vector-set! input 2 'app))
            ((equal? answer 'f)
             (vector-set! input 2 'stud))
            ((equal? answer 'g)
             (vector-set! input 2 'con))
            ((equal? answer 'h)
             (vector-set! input 2 'std))
```

```
            ((equal? answer 'i)
             (vector-set! input 2 'util))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-grade wood input))))))

(define get-width
  (lambda (wood input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the nominal width of the " (type wood) "s?")
    (newline)
    (writeln "a.  2 inches")
    (writeln "b.  3 inches")
    (writeln "c.  4 inches")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 4 2))
            ((equal? answer 'b)
             (vector-set! input 4 3))
            ((equal? answer 'c)
             (vector-set! input 4 4))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-width wood input))))))

(define choose-grade
  (lambda (input)
    (or (equal? (vector-ref input 2) 'con)
        (equal? (vector-ref input 2) 'std)
        (equal? (vector-ref input 2) 'util))))

(define get-depth
  (lambda (wood input)
    (cond ((choose-grade input)
           (newline)
           (writeln "********************")
           (newline)
           (writeln "This grade of lumber only comes in a nominal depth of 4 inches.")
           (writeln "Would you like to change the grade of " (type wood) " lumber?")
           (yes-no)
           (let ((answer (read)))
             (cond ((equal? answer 'a)
                    (get-grade wood input)
                    (get-depth wood input))
                   ((equal? answer 'b)
                    (vector-set! input 5 4))
                   (else
                    (writeln "Your entry is not defined, please try again.")
                    (get-depth wood input)))))
          (else
           (cond ((equal? (vector-ref input 4) 2)
                  (get-depth2 wood input))
                 ((equal? (vector-ref input 4) 3)
                  (get-depth3 wood input))
                 (else
                  (get-depth4 wood input)))))))

(define get-depth2
  (lambda (wood input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the nominal depth of the " (type wood) "s?")
```

```
    (newline)
    (writeln "a.  3 inches")
    (writeln "b.  4 inches")
    (writeln "c.  5 inches")
    (writeln "d.  6 inches")
    (writeln "e.  8 inches")
    (writeln "f.  10 inches")
    (writeln "f.  12 inches")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 5 3))
            ((equal? answer 'b)
             (vector-set! input 5 4))
            ((equal? answer 'c)
             (vector-set! input 5 5))
            ((equal? answer 'd)
             (vector-set! input 5 6))
            ((equal? answer 'e)
             (vector-set! input 5 8))
            ((equal? answer 'f)
             (vector-set! input 5 10))
            ((equal? answer 'g)
             (vector-set! input 5 12))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-depth wood input))))))

(define get-depth3
  (lambda (wood input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the nominal depth of the " (type wood) "s?")
    (newline)
    (writeln "a.  2 inches")
    (writeln "b.  4 inches")
    (writeln "c.  5 inches")
    (writeln "d.  6 inches")
    (writeln "e.  8 inches")
    (writeln "f.  10 inches")
    (writeln "g.  12 inches")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 5 2))
            ((equal? answer 'b)
             (vector-set! input 5 4))
            ((equal? answer 'c)
             (vector-set! input 5 5))
            ((equal? answer 'd)
             (vector-set! input 5 6))
            ((equal? answer 'e)
             (vector-set! input 5 8))
            ((equal? answer 'f)
             (vector-set! input 5 10))
            ((equal? answer 'g)
             (vector-set! input 5 12))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-depth wood input))))))

(define get-depth4
  (lambda (wood input)
    (newline)
    (writeln "********************")
```

```
    (newline)
    (writeln "What is the nominal depth of the " (type wood) "s?")
    (newline)
    (writeln "a.  2 inches")
    (writeln "b.  3 inches")
    (writeln "c.  4 inches")
    (writeln "d.  5 inches")
    (writeln "e.  6 inches")
    (writeln "f.  8 inches")
    (writeln "g.  10 inches")
    (writeln "h.  12 inches")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 5 2))
            ((equal? answer 'b)
             (vector-set! input 5 3))
            ((equal? answer 'c)
             (vector-set! input 5 4))
            ((equal? answer 'd)
             (vector-set! input 5 5))
            ((equal? answer 'e)
             (vector-set! input 5 6))
            ((equal? answer 'f)
             (vector-set! input 5 8))
            ((equal? answer 'g)
             (vector-set! input 5 10))
            ((equal? answer 'h)
             (vector-set! input 5 12))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-depth wood input))))))

(define get-size
  (lambda (input)
    (cond ((choose-grade input)
           (vector-set! input 3 's2to4/4))
          ((and (equal? (vector-ref input 2) (or 'ss 'n1 'n2 'n3 'app 'stud))
                (>= (vector-ref input 5) 5))
           (vector-set! input 3 's2to4/>=5))
          (else
           (vector-set! input 3 's2to4/2to4)))))

(define get-supcond
  (lambda (wood input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "Is the " (type wood) " continuous over 1, 2, or 3 or more spans?")
    (newline)
    (writeln "a.  Continuous over 1 span only.")
    (writeln "b.  Continuous over 2 spans.")
    (writeln "c.  Continuous over 3 or more spans.")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input (if-ply wood) 'span-1))
            ((equal? answer 'b)
             (vector-set! input (if-ply wood) 'span-2))
            ((equal? answer 'c)
             (vector-set! input (if-ply wood) 'span-3))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-supcond wood input))))))

(define if-ply
```

```
  (lambda (wood)
    (if (equal? (type wood) 'plyform)
        3
        6)))

(define get-fb
  (lambda (wood input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "Which allowable bending stress should be used for the " (type wood) "s?")
    (newline)
    (writeln "a.  Single.")
    (writeln "b.  Repetitive.")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 7 'single))
            ((equal? answer 'b)
             (vector-set! input 7 'repetitive))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-fb wood input))))))

(define get-wale-info
  (lambda (wood input)
    (get-info-text wood input)
    (zero input)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (get-species wood input)
             (get-grade wood input)
             (get-width wood input)
             (get-depth wood input)
             (get-size input)
             (get-wale-soldier input)
             (get-fb wood input)
             (get-supcond wood input))
            ((equal? answer 'b)
             (get-width wood input)
             (get-depth wood input)
             (get-supcond wood input)
             (get-wale-soldier input)
             (get-user-stress wood input))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-wale-info wood input))))))

(define get-wale-soldier
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "Are the wales single or double soldiers?")
    (newline)
    (writeln "a.  Single.")
    (writeln "b.  Double.")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 8 'single))
            ((equal? answer 'b)
             (vector-set! input 8 'double))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-wale-soldier input))))))
```

```
(define get-tie-info
  (lambda(input)
    (zero-tie input)
    (get-tie-cap input)
    (get-tie-pshape input)))

(define zero-tie
  (lambda (input)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)
    (vector-set! input 0 '?)))

(define get-tie-cap
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the maximum capacity of the form ties, in pounds?")
    (newline)
    (writeln "Tie Capacity:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 1 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-tie-cap input))))))

(define get-tie-pshape
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the shape of the form tie's bearing plate?")
    (newline)
    (writeln "a.  Circular.")
    (writeln "b.  Rectangular.")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 2 'circular)
             (get-tie-pdiam input))
            ((equal? answer 'b)
             (vector-set! input 2 'rectangular)
             (get-tie-pwidth input)
             (get-tie-plength input))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-tie-pshape input))))))

(define get-tie-pwidth
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the width of the tie plate, in inches?")
    (writeln "It is assumed that this dimension of the tie plate runs PERPENDICULAR")
    (writeln "to the wale.")
    (newline)
    (writeln "Tie Plate Width:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 3 answer))
```

```
            (else
             (writeln "Your response is not a number, please try again.")
             (get-tie-pwidth input))))))

(define get-tie-plength
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the length of the tie plate, in inches?")
    (writeln "It is assumed that this dimension of the tie plate runs PARALLEL to")
    (writeln "the wale.")
    (newline)
    (writeln "Tie Plate Length:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 4 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-tie-plength input))))))

(define get-tie-pdiam
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the diameter of the tie plate, in inches?")
    (newline)
    (writeln "Tie Plate Diameter:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 5 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-tie-pdiam input))))))

(define get-def-info
 (lambda (input)
   (vector-set! input 0 '?)
   (vector-set! input 1 '?)
   (get-d1 input)
   (get-d2 input)))

(define get-d1
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the first deflection limit for the forms?")
    (newline)
    (writeln "a.  Deflection not greater than 1/180 of the span.")
    (writeln "b.  Deflection not greater than 1/240 of the span.")
    (writeln "c.  Deflection not greater than 1/360 of the span.")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 0 'span/180))
            ((equal? answer 'b)
             (vector-set! input 0 'span/240))
            ((equal? answer 'c)
             (vector-set! input 0 'span/360))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-d1 input))))))

(define get-d2
```

```
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the second deflection limit for the forms?")
    (newline)
    (writeln "a.  1/8 inch")
    (writeln "b.  1/16 inch")
    (writeln "c.  None")
    (newline)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (vector-set! input 1 'span1/8))
            ((equal? answer 'b)
             (vector-set! input 1 'span1/16))
            ((equal? answer 'c)
             (vector-set! input 1 'none))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-d2 input))))))

(define get-conc-info
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "Would you like for the Wall Form Design Program to compute the")
    (writeln "value for the concrete pressure, or would you like to enter your")
    (writeln "own value?")
    (newline)
    (writeln "a.  Value computed by computer.")
    (writeln "b.  User input value.")
    (newline)
    (zero-conc input)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (get-rate input)
             (get-temp input)
             (get-height input))
            ((equal? answer 'b)
             (get-user-conc input))
            (else
             (writeln "Your entry is not defined, please try again.")
             (get-conc-info input))))))

(define zero-conc
  (lambda (input)
    (vector-set! input 0 '?)
    (vector-set! input 1 '?)
    (vector-set! input 2 '?)
    (vector-set! input 3 '?)))

(define get-user-conc
  (lambda (input)
    (newline)
    (writeln "Concrete pressure inside wall forms, in psf:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 3 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-user-conc input))))))

(define get-rate
  (lambda (input)
    (newline)
```

```
    (writeln "********************")
    (newline)
    (writeln "What is the rate of placement of concrete in the wall forms,")
    (writeln "in feet per hour (1 to 10)?")
    (newline)
    (writeln "Rate of placement:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 0 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-rate input))))))

(define get-temp
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the temperature of the concrete during placement,")
    (writeln "in degrees farenheit (40 to 90)?")
    (newline)
    (writeln "Concrete temperature:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 1 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-temp input))))))

(define get-height
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "What is the depth of the wall form that will be accepting the concrete,")
    (writeln "in feet ?")
    (newline)
    (writeln "Wall form depth:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 2 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-height input))))))

(define user-space
  (lambda (wood input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "The maximum " (type wood) " spacing has been computed to be:  " (round-2-pla
ces (vector-ref wood 23)) " inches.")
    (newline)
    (writeln "Would you like to use a smaller value for the " (type wood) " spacing?")
    (yes-no)
    (let ((answer (read)))
      (cond ((equal? answer 'a)
             (get-revised-space input)
             (check-max wood input))
            ((equal? answer 'b))
            (else
             (writeln "Your entry is not defined, please try again.")
             (user-space wood input))))))

(define check-max
  (lambda (wood input)
```

```
    (cond ((> (vector-ref input 0) (spacing wood))
           (vector-set! input 0 '?)
           (newline)
           (writeln "!!!Your entry is larger than the maximum allowable " (type wood) " sp
acing!!!")
           (writeln "Please try a smaller value.")
           (user-space wood input)))))

(define get-revised-space
  (lambda (input)
    (newline)
    (writeln "********************")
    (newline)
    (writeln "Enter your value for the spacing now, in inches.")
    (writeln "This value must not exceed the maximum stud spacing already computed.")
    (newline)
    (writeln "Spacing:")
    (let ((answer (read)))
      (cond ((number? answer)
             (vector-set! input 0 answer))
            (else
             (writeln "Your response is not a number, please try again.")
             (get-revised-space input))))))

(define disp-ply-class
  (lambda (input)
    (cond ((equal? (vector-ref input 0) 'c1)
           "Class I")
          ((equal? (vector-ref input 0) 'c2)
           "Class II")
          ((equal? (vector-ref input 0) 's1)
           "Structural I")
          (else
           "Plyform class not specified"))))

(define disp-ply-depth
  (lambda (wood)
    (cond ((equal? (depth wood) 0.46875)
           "15/32 inch")
          ((equal? (depth wood) 0.5)
           "1/2 inch")
          ((equal? (depth wood) 0.59375)
           "19/32 inch")
          ((equal? (depth wood) 0.625)
           "5/8 inch")
          ((equal? (depth wood) 0.71875)
           "23/32 inch")
          ((equal? (depth wood) 0.75)
           "3/4 inch")
          ((equal? (depth wood) 0.875)
           "7/8 inch")
          ((equal? (depth wood) 1.0)
           "1 inch")
          ((equal? (depth wood) 1.125)
           "1 1/4 inches"))))

(define disp-species
  (lambda (input)
    (cond ((equal? (vector-ref input 1) 'a)
           "Aspen")
          ((equal? (vector-ref input 1) 'bf)
           "Balsam Fir")
          ((equal? (vector-ref input 1) 'dfl)
           "Douglas Fir-Larch")
          ((equal? (vector-ref input 1) 'dfs)
           "Douglas Fir South")
```

```
          ((equal? (vector-ref input 1) 'eh)
           "Eastern Hemlock")
          ((equal? (vector-ref input 1) 'eht)
           "Eastern Hemlock-Tamarack")
          ((equal? (vector-ref input 1) 'es)
           "Eastern Spruce")
          ((equal? (vector-ref input 1) 'ewp)
           "Eastern White Pine")
          ((equal? (vector-ref input 1) 'ew)
           "Eastern Woods")
          ((equal? (vector-ref input 1) 'esaf)
           "Engelmann Spruce-Alpine Fir (Engelmann Spruce-Lodgepole Pine")
          ((equal? (vector-ref input 1) 'hf)
           "Hem-Fir")
          ((equal? (vector-ref input 1) 'iwp)
           "Idaho White Pine")
          ((equal? (vector-ref input 1) 'lp)
           "Lodgepole Pine")
          ((equal? (vector-ref input 1) 'mh)
           "Mountain Hemlock")
          ((equal? (vector-ref input 1) 'na)
           "Northern Aspen")
          ((equal? (vector-ref input 1) 'np)
           "Northern Pine")
          ((equal? (vector-ref input 1) 'nwc)
           "Northern White Cedar")
          ((equal? (vector-ref input 1) 'ppsp)
           "Ponderosa Pine-Sugar Pine (Ponderosa Pine-Lodgepole Pine)")
          ((equal? (vector-ref input 1) 'ss)
           "Sitka Spruce")
          ((equal? (vector-ref input 1) 'sp)
           "Southern Pine")
          ((equal? (vector-ref input 1) 'wc)
           "Western Cedars")
          ((equal? (vector-ref input 1) 'wh)
           "Western Hemlock")
          ((equal? (vector-ref input 1) 'ww)
           "White Woods (Western Woods)")
          (else
           "Lumber species not specified"))))

(define disp-grade
  (lambda (input)
    (cond ((equal? (vector-ref input 2) 'ss)
           "Select Structural grade")
          ((equal? (vector-ref input 2) 'n1)
           "No. 1 grade")
          ((equal? (vector-ref input 2) 'n2)
           "No. 2 grade")
          ((equal? (vector-ref input 2) 'n3)
           "No. 3 grade")
          ((equal? (vector-ref input 2) 'app)
           "Appearance grade")
          ((equal? (vector-ref input 2) 'stud)
           "Stud grade")
          ((equal? (vector-ref input 2) 'con)
           "Construction grade")
          ((equal? (vector-ref input 2) 'util)
           "Utility grade")
          ((equal? (vector-ref input 2) 'std)
           "Standard grade")
          (else
           "Lumber grade not specified"))))

(define disp-width
  (lambda (input)
```

```
    (vector-ref input 4)))

(define disp-depth
  (lambda (input)
    (vector-ref input 5)))

(define disp-soldier
  (lambda (input)
    (cond ((equal? (vector-ref input 8) 'single)
           "Single")
          (else
           "Double"))))

(define disp-plate-diam
  (lambda (input)
    (vector-ref input 5)))

(define disp-plate-width
  (lambda (input)
    (vector-ref input 3)))

(define disp-plate-length
  (lambda (input)
    (vector-ref input 4)))

(define disp-f-b
  (lambda (input wood)
    (cond ((equal? (vector-ref input 7) 'single)
           (f-b-sing wood))
          (else
           (f-b-rep wood)))))

(define disp-f-b-type
  (lambda (input)
    (cond ((equal? (vector-ref input 7) 'single)
           "Single")
          (else
           "Repetitive"))))

(define disp-ply-dir
  (lambda (input)
    (cond ((equal? (vector-ref input 1) 'para)
           "parallel")
          (else
           "perpendicular"))))

(define disp-tie-shape
  (lambda (input)
    (cond ((equal? (vector-ref input 2) 'rectangular)
           "Rectangular")
          (else
           "Circular"))))

(define disp-plate-size
  (lambda (input)
    (cond ((equal? (vector-ref input 2) 'rectangular)
           (writeln "    - " (vector-ref input 3) " x " (vector-ref input 4) " inch bearin
g area."))
          (else
           (writeln "    - " (vector-ref input 5) " inch diameter bearing area."))))

(define rolling
  (lambda (wood)
    (cond ((equal? (type wood) 'plyform)
           "rolling "))))
```

```
(define disp-def
  (lambda (input)
    (cond ((equal? (vector-ref input 0) 'span1/180)
           (writeln "    - Not to exceed 1/180 of any span length."))
          ((equal? (vector-ref input 0) 'span1/240)
           (writeln "    - Not to exceed 1/240 of any span length."))
          (else
           (writeln "    - Not to exceed 1/360 of any span length.")))
    (cond ((equal? (vector-ref input 1) 'span1/8)
           (writeln "    - Not to exceed 1/8 inch total."))
          ((equal? (vector-ref input 1) 'span1/16)
           (writeln "    - Not to exceed 1/16 inch total.")))))
```

```
(module default)

;;-----------------------------------------------------------------------
;; Default wall form design criteria
;;-----------------------------------------------------------------------

;;-----------------------------------------------------------------------
;; Plyform
;;-----------------------------------------------------------------------

;;  0 index - Plyform class
;;  1 index - Direction of face grain relative to stud span
;;  2 index - Plyform depth
;;  3 index - Plyform support conditions
;;  4 index - User input allowable bending stress
;;  5 index - User input modulus of elasticity
;;  6 index - User input rolling shear stress
;;  7 index - User input rolling shear constant

(define default-ply
  (lambda ()
    (set! ply-vec (make-vector 8 '?))
    (vector-set! ply-vec 0 'c2)
    (vector-set! ply-vec 1 'para)
    (vector-set! ply-vec 2 0.75)
    (vector-set! ply-vec 3 'span-3)
    (vector-set! ply-vec 4 '?)
    (vector-set! ply-vec 5 '?)
    (vector-set! ply-vec 6 '?)
    (vector-set! ply-vec 7 '?)))

;;-----------------------------------------------------------------------
;; Studs
;;-----------------------------------------------------------------------

;;  0 index - User input stud spacing
;;  1 index - Species of lumber
;;  2 index - Lumber grade
;;  3 index - Size Classification
;;  4 index - Nominal lumber width
;;  5 index - Nominal lumber depth
;;  6 index - Stud support conditions
;;  7 index - Single or Repetitive allowable bending stress?
;;  8 index - Unused
;;  9 index - User input allowable bending stress (Fb)
;; 10 index - User input modulus of elasticity (E)
;; 11 index - User input allowable compressive stress (perpendicular) (Fc)
;; 12 index - User input allowable rolling shear stress (Fv)

(define default-stud
  (lambda ()
    (set! stud-vec (make-vector 13 '?))
    (vector-set! stud-vec 0 '?)
    (vector-set! stud-vec 1 '?)
    (vector-set! stud-vec 2 '?)
    (vector-set! stud-vec 3 '?)
    (vector-set! stud-vec 4 2)
    (vector-set! stud-vec 5 6)
    (vector-set! stud-vec 6 'span-3)
    (vector-set! stud-vec 7 'repetitive)
    (vector-set! stud-vec 8 '?)
    (vector-set! stud-vec 9 1450)
    (vector-set! stud-vec 10 1700000)
    (vector-set! stud-vec 11 385)
    (vector-set! stud-vec 12 185)))
```

```
;;-----------------------------------------------------------------------
;; Wales
;;-----------------------------------------------------------------------

;;  0 index - User input wale spacing
;;  1 index - Species of lumber
;;  2 index - Lumber grade
;;  3 index - Size Classification
;;  4 index - Nominall lumber width
;;  5 index - Nominal lumber depth
;;  6 index - Wale support conditions
;;  7 index - Single or repetitive allowable bending stress?
;;  8 index - Single or double wale system?
;;  9 index - User input allowable bending stress (Fb)
;; 10 index - User input modulus of elasticity (E)
;; 11 index - User input allowable compressive stress (perpendicular) (Fc)
;; 12 index - User input allowable rolling shear stress (Fv)


(define default-wale
  (lambda ()
    (set! wale-vec (make-vector 13 '?))
    (vector-set! wale-vec 0 '?)
    (vector-set! wale-vec 1 '?)
    (vector-set! wale-vec 2 '?)
    (vector-set! wale-vec 3 '?)
    (vector-set! wale-vec 4 2)
    (vector-set! wale-vec 5 6)
    (vector-set! wale-vec 6 'span-3)
    (vector-set! wale-vec 7 'repetitive)
    (vector-set! wale-vec 8 'double)
    (vector-set! wale-vec 9 1450)
    (vector-set! wale-vec 10 1700000)
    (vector-set! wale-vec 11 385)
    (vector-set! wale-vec 12 185)))

;;-----------------------------------------------------------------------
;; Ties
;;-----------------------------------------------------------------------

;;  0 index - User input tie spacing
;;  1 index - Tie capacity
;;  2 index - Tie plate shape (if circular, index 4 and 5 leave blank)
;;  3 index - Tie plate width (shortest dimension)
;;  4 index - Tie plate length (longest dimension)
;;  5 index - Tie plate diameter

(define default-tie
  (lambda ()
    (set! tie-vec (make-vector 6 '?))
    (vector-set! tie-vec 0 '?)
    (vector-set! tie-vec 1 4000)
    (vector-set! tie-vec 2 'rectangular)
    (vector-set! tie-vec 3 4)
    (vector-set! tie-vec 4 4)
    (vector-set! tie-vec 5 '?)))

;;-----------------------------------------------------------------------
;; Deflection Limits
;;-----------------------------------------------------------------------

;;  0 index - First deflection limit (span-dependent)
;;  1 index - Second deflection limit (absolute)

(define default-def
  (lambda ()
```

```
    (set! def-vec (make-vector 2 '?))
    (vector-set! def-vec 0 'span/360)
    (vector-set! def-vec 1 'span1/8)))

;;-----------------------------------------------------------------------
;; Concrete
;;-----------------------------------------------------------------------

;;  0 index - Rate of placement of concrete
;;  1 index - Temperature of concrete
;;  2 index - Height of wall form
;;  3 index - User input maximum concrete pressure

(define default-conc
  (lambda ()
    (set! conc-vec (make-vector 4 '?))
    (vector-set! conc-vec 0 '?)
    (vector-set! conc-vec 1 '?)
    (vector-set! conc-vec 2 '?)
    (vector-set! conc-vec 3 1000)))

(define load-default
  (lambda ()
    (default-ply)
    (default-stud)
    (default-wale)
    (default-tie)
    (default-def)
    (default-conc)))
```

```
(module design)

;;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
;;                          Concrete Form Design Program
;;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

;;-----------------------------------------------------------------------------
;;  Miscellaneous procedures
;;-----------------------------------------------------------------------------

(define writeln
  (lambda args
    (for-each display args)
    (newline)))

;;The procedure ROUND-N-PLACES takes any number as an orgument and rounds it
;;to two decimal places.

(define round-n-places
  (lambda (n dec-num)
    (let ((scale-factor (expt 10 n)))
      (/ (round (* dec-num scale-factor)) scale-factor))))

(define round-2-places
  (lambda (dec-num)
    (round-n-places 2 dec-num)))

;;The procedure SQUARE takes a number as argument and returns its square.

(define square
  (lambda (x)
    (* x x)))

;;-----------------------------------------------------------------------------
;;  Design a wall form
;;-----------------------------------------------------------------------------

;;The procedure DESIGN is the primary procedure in the wall form design program.
;;DESIGN takes as arguments six vectors of information input from the program user.
;;PLY-VEC provides information regarding the type of plyform to be used in the
;;wall form system.  STUD-VEC provides information about the studs, WALE-VEC
;;about the walers, and TIE-VEC about the form ties.  DEF-VEC provides the
;;allowable deflection criteria for the different members, and CONC-VEC provides
;;information about the concrete to be used.
;;
;;First, the procedure LOAD-WOOD is called taking the six user defined vectors
;;as arguments.  This procedure creates four additional vectors called PLYFORM,
;;STUD, WALE and TIE.  These are the workhorses of the program.  They carry
;;all the information required for execution of the program from start to finish.
;;
;;Next, a series of steps determines the minimum spacing for each member of the
;;wall form system and stores this minimum spacing value in its vector.  For
;;example, the first spacing is determined by calling the procedure WOOD-SPACING
;;and applying it to the vector PLYFORM.  This takes the newly loaded PLYFORM
;;vector information and determines the minimum stud spacing.  The stud spacing
;;is then stored in Index 23 of the STUD vector.
;;
;;Immediately after each spacing step, the concrete pressure has to be determined
;;before the next member's spacing can be determined.  For instance, we have just
;;computed the stud spacing, so DESIGN next computes the uniform load from the
;;concrete on the studs at that spacing.  This concrete pressure initially used
;;to determine the stud spacing is computed by taking the pressure W from below
;;and applying it along a 12 inch wide strip of the plyform, assuming it acts
;;as a simply supported beam.  The computed concrete pressure for each member is
;;stored in Index 17 of that member.
```

```
(define design
  (lambda (ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
    (load-wood ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
    (start-input ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
    (summary ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)))

(define start-input
  (lambda (ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
    (vector-set! stud 23 (round-2-places (wood-spacing plyform)))
    (user-space stud stud-vec)
    (vector-set! stud 17 (wood-load stud stud-vec))
    (vector-set! wale 23 (round-2-places (wood-spacing stud)))
    (user-space wale wale-vec)
    (vector-set! wale 17 (wood-load wale wale-vec))
    (vector-set! tie 23 (round-2-places (wood-spacing wale)))
    (user-space tie tie-vec)
    (vector-set! tie 17 (wood-load tie tie-vec))))

;;The procedure W computes the lateral concrete pressure exerted on the forms.
;;The variable H is the height of the wall forms in feet.
;;The variable R is the rate of vertical placement of concrete in the wall
;;form in feet per hour.
;;The variable T is the temperature of the concrete in degrees farenheit.
;;The value of W will be placed in Index 17 of each WOOD (i.e., PLYWOOD, STUD,
;;WALE and TIE) vector.

(define w
  (lambda (h r t)
    (cond ((< r 7)
           (min (+ 150 (/ (* 9000 r) t))
                (* 150 h)
                2000))
          ((and (>= r 7) (<= r 10))
           (min (+ 150 (/ 43400 t) (/ (* 2800 r) t))
                (* 150 h)
                2000))
          (else (* 150 h)))))

;;The procedure WOOD-LOAD computes the uniform loading per foot of width for each
;;WOOD vector by taking the pressure (W) from Index 17 and multiplying it by the
;;spacing from Index 23.

(define wood-load
  (lambda (wood input)
    (cond ((equal? (type wood) 'stud)
           (s-w-load wood input))
          ((equal? (type wood) 'wale)
           (s-w-load wood input))
          (else (/ (* (vector-ref plyform 17)
                      (choose-space stud stud-vec)
                      (choose-space wale wale-vec))
                   144)))))

;;The procedure S-W-LOAD is an interim procedure that computes the concrete load on
;;the studs and wales.

(define s-w-load
  (lambda (wood input)
    (/ (* (vector-ref plyform 17)
          (choose-space wood input))
       12)))

;;The procedure WOOD-SPACING computes the maximum allowable spacing for each
;;WOOD by taking the smallest value of spacing computed from bending, shear,
;;deflection, compression and tie capacity criteria.
```

```
(define wood-spacing
  (lambda (wood)
    (cond ((equal? (type wood) 'plyform)
           (min-b-s-d wood))
          ((equal? (type wood) 'stud)
           (min (min-b-s-d wood)
                (compression wood wale)))
          (else
           (min (min-b-s-d wood)
                (compression wood tie)
                (m-max wood 6)
                (s-max wood 6)
                (capacity tie))))))

;;The procedure MIN-B-S-D is an interim procedure that computes the maximum
;;allowable spacing for each WOOD as governed by bending, shear and deflection.

(define min-b-s-d
  (lambda (wood)
    (min (bending wood)
         (shear wood)
         (deflection wood))))

;;The procedure BENDING computes the maximum allowable spacing for each WOOD
;;as governed by bending criteria.

(define bending
  (lambda (wood)
    ((lookup2d 'bending
               (sup-cond wood)
               cfde) (choose-stress f-b wood)
                     (sm wood)
                     (unif-load wood))))

;;The procedure CHOOSE-STRESS selects the applicable value of allowable bending
;;stress, allowable rolling shear stress, allowable compressive stress, or
;;modulus of elasticity, in pounds per square inch, for the type of wood.

(define choose-stress
  (lambda (psi wood)
    (cond ((equal? (type wood) 'stud)
           (if (equal? (vector-ref stud-vec (psi-index psi)) '?)
               (psi wood)
               (vector-ref stud-vec (psi-index psi))))
          ((equal? (type wood) 'wale)
           (if (equal? (vector-ref wale-vec (psi-index psi)) '?)
               (psi wood)
               (vector-ref wale-vec (psi-index psi))))
          (else
           (psi wood)))))

;;The procedure PSI-INDEX returns the WOOD index number where either f-b, f-c-perp,
;;f-v, or e is stored.

(define psi-index
  (lambda (psi)
    (cond ((equal? psi f-b) 9)
          ((equal? psi e) 10)
          ((equal? psi f-c-perp) 11)
          (else 12))))

;;The procedure SHEAR computes the maximum allowable spacing for each WOOD
;;as governed by ROLLING SHEAR criteria.

(define shear
  (lambda (wood)
```

```
    (cond ((equal? (type wood) 'plyform)
           ((lookup2d 'shearP
                      (sup-cond wood)
                      cfde) (choose-stress f-v wood)
                            (ib/q wood)
                            (unif-load wood)
                            (depth wood)))
          (else
           ((lookup2d 'shearL
                      (sup-cond wood)
                      cfde) (choose-stress f-v wood)
                            (area wood)
                            (unif-load wood)
                            (depth wood))))))

;;The procedure DEFLECTION computes the maximum allowable spacing for each
;;WOOD as governed by deflection criteria.

(define deflection
  (lambda (wood)
    (cond ((equal? (deflection-lim2 wood) 'none)
           (def1-eq wood))
          (else (min (def1-eq wood)
                     (def2-eq wood))))))

;;The procedure DEF1-EQ is an interim procedure that inserts the equation
;;parameters into the first deflection equations.

(define def1-eq
  (lambda (wood)
    ((lookup2d (deflection-lim1 wood)
               (sup-cond wood)
               cfde) (choose-stress e wood)
                     (moi wood)
                     (unif-load wood))))

;;The procedure DEF2-EQ is an interim procedure that inserts the equation
;;parameters into the second deflection equations.

(define def2-eq
  (lambda (wood)
    ((lookup2d (deflection-lim2 wood)
               (sup-cond wood)
               cfde) (choose-stress e wood)
                     (moi wood)
                     (unif-load wood))))

;;The procedure COMPRESSION computes the maximum allowable spacing for each
;;WOOD as governed by allowable compressive strength.

(define compression
  (lambda (mem1 mem2)
    (/ (* (modify-fc (choose-stress f-c-perp mem1) mem2)
          (bear-area mem1 mem2)
          144)
       (* (unif-load plyform)
          (cond ((equal? (type mem1) 'stud)
                 (choose-space mem1 stud-vec))
                (else
                 (choose-space mem1 wale-vec)))))))

(define modify-fc
  (lambda (fc mem2)
    (if (equal? (soldier mem2) 'double)
        (new-fc fc (+ 0.5 (* 2 (width mem2))) mem2)
        (new-fc fc (width mem2) mem2))))
```

```
(define new-fc
  (lambda (fc bear-length mem2)
    (cond ((< bear-length 6)
           (* fc
              (/ (+ (width mem2) 0.375)
                 (width mem2))))
          (else (* fc 1)))))


;;The procedure BEAR-AREA computes the net bearing area between two members of
;;the wall form system across which the perpendicular compressive force is
;;being applied.

(define bear-area
  (lambda (mem1 mem2)
    (if (equal? mem1 stud)
        (stud-wale-area mem1 mem2)
        (wale-tie-area mem1 mem2))))

(define wale-tie-area
  (lambda (mem1 mem2)
    (cond ((equal? (soldier wale) 'double)
           (cond ((equal? (species tie) 'rectangular)
                  (dbl-rec-area mem1 mem2))
                 (else
                  (dbl-cir-area mem1 mem2))))
          (else
           (cond ((equal? (species tie) 'rectangular)
                  (sgl-rec-area mem1 mem2))
                 (else
                  (sgl-cir-area mem1 mem2)))))))

;;The procedure STUD-WALE-AREA computes the bearing area specifically between
;;the stud and wale members for either a single or double wale system.

(define stud-wale-area
  (lambda (mem1 mem2)
    (* (width mem1)
       (width mem2)
       (if (equal? (soldier mem2) 'double)
           2
           1))))

;;The procedure DBL-REC-AREA computes the bearing area between the waler and
;;the tie plate or washer if:
;;      -  the system uses double wales
;;      -  and, the tie plates are rectangular.
;;The procedure first checks to see which is greater, the effective width of
;;the waler or the shortest dimension of the rectangular tie plate.  It is
;;assumed here that the longest dimension of the tie plate will always be
;;placed along the length of the waler.
;;If the effective wale width is greater, then the area is computed as tie
;;plate width, minus 1/2 inch (between walers not bearing), times the tie
;;plate depth.
;;If the tie plate width is greater, then the area is computed as effective
;;wale width, minus 1/2 inch, times the tie plate width.

(define dbl-rec-area
  (lambda (mem1 mem2)
    (if (<= (eff-wale-width mem1 mem2)
            (width mem2))
        (* (- (eff-wale-width mem1 mem2) 0.5)
           (depth mem2))
        (* (- (width mem2) 0.5)
           (depth mem2)))))
```

```
;;The procedure DBL-CIR-AREA computes the bearing area between the waler and
;;the tie plate or washer if:
;;      - the system uses double wales
;;      - and, the tie plates are circular
;;The same basic procedure is applied here as in DBL-REC-AREA above to compute
;;net bearing area.

(define dbl-cir-area
  (lambda (mem1 mem2)
    (if (<= (eff-wale-width mem1 mem2)
            (diameter mem2))
        (- (- (/ (* 3.1415
                    (square (diameter mem2)))
                 4)
              (* 2
                 (overlap-area mem1 mem2)))
           (* 0.5
              (diameter mem2)))
        (- (/ (* 3.1415
                 (square (diameter mem2)))
              4)
           (* 0.5
              (diameter mem2))))))

;;The procedure SGL-REC-AREA computes the bearing area between the waler and
;;the tie plate or washer if:
;;      - the system uses single wales
;;      - and, the tie plates are rectangular
;;This is a simplified case of DBL-REC-AREA above where there is a single wale
;;and, therefore no gap exists to be subtracted out of the net bearing area.

(define sgl-rec-area
  (lambda (mem1 mem2)
    (if (<= (width mem1)
            (width mem2))
        (* (width mem1)
           (depth mem2))
        (* (width mem2)
           (depth mem2)))))

;;The procedure SGL-CIR-AREA computes the bearing area between the waler and
;;the tie plate or washer if:
;;      - the system uses single wales
;;      - and, the tie plates are circular
;;This is a simplified case of DBL-CIR-AREA above where there is a single wale
;;and, therefore no gap exists to be subtracted out of the net bearing area.

(define sgl-cir-area
  (lambda (mem1 mem2)
    (if (<= (width mem1)
            (diameter mem2))
        (- (- (/ (* 3.1415
                    (square (diameter mem2)))
                 4)
              (overlap-area mem1 mem2))
           (/ (* 3.1415
                 (square 0.5))
              4))
        (- (/ (* 3.1415
                 (square (diameter mem2)))
              4)
           (/ (* 3.1415
                 (square 0.5))
              4)))))
```

```
;;The procedure EFF-WALE-WIDTH computes the total width of the double waler
;;from edge to edge, including the space between allowed for by the tie.  This
;;gap is estimated for all cases to be 1/2 inch.

(define eff-wale-width
  (lambda (mem1 mem2)
    (if (equal? (soldier wale) 'double)
        (+ 0.5
           (* 2
              (width mem1)))
        (width mem1))))

;;The procedure OVERLAP-AREA computes the area of the circular bearing plate or
;;washer of the tie that overlaps beyond the edge of the waler.  This area is
;;computed for the purpose of determining the net area of the bearing plate or
;;washer of the tie to be used in compressive force calculations.

(define overlap-area
  (lambda (mem1 mem2)
    (* (c mem1 mem2)
       (overlap-length mem1 mem2))))

;;The procedure OVERLAP-LENGTH determines the distance from the edge of the waler
;;to the edge of the circular tie plate or washer.  This value is required for
;;computing the overlap area

(define overlap-length
  (lambda (mem1 mem2)
    (/ (- (diameter mem2)
          (eff-wale-width mem1 mem2))
       2)))

;;The procedure C computes the overlap area for a given overlap length.

(define c
  (lambda (mem1 mem2)
    (* (diameter mem2)
       (sin (acos (/ (- 1
                        (/ (overlap-length mem1 mem2)
                           2))
                     2))))))

;;The procedure CAPACITY computes the maximum allowable spacing for each WOOD
;;as governed by the capacity of the ties being used in the wall form system.

(define capacity
  (lambda (wood)
    (* (/ (tie-cap wood)
          (unif-load wale))
       12)))

(define moment
  (lambda (wood guess-t-sp)
    (cond ((odd? (n-load guess-t-sp))
           (odd-load-mom wood guess-t-sp))
          (else
           (even-load-mom wood guess-t-sp)))))

(define n-load
  (lambda (guess-t-sp)
    (cond ((< guess-t-sp (choose-space stud stud-vec)) 1)
          (else
           (float->integer (truncate (/ guess-t-sp (choose-space stud stud-vec)))))))

(define odd-load-mom
  (lambda (wood guess-t-sp)
```

```
    (/ (- (* (n-load guess-t-sp)
             (unif-load plyform)
             (choose-space stud stud-vec)
             (choose-space wood wale-vec)
             guess-t-sp)
          (* (coef1 (n-load guess-t-sp))
             (unif-load plyform)
             (choose-space wood wale-vec)
             (square (choose-space stud stud-vec))))
       (* 144 4))))

(define coef1
  (lambda (guess-t-sp)
    (/ (- (square (n-load guess-t-sp))
          1)
       2)))

(define m-max
  (lambda (wood guess-t-sp)
    (cond ((m-close-enough? (moment wood guess-t-sp) wood)
           guess-t-sp)
          (else
           (m-max wood (improve guess-t-sp))))))

(define m-close-enough?
  (lambda (moment wood)
    (< (- (* (f-b wood)
             (sm wood))
          moment)
       0.01)))

(define improve
  (lambda (guess-t-sp)
    (+ guess-t-sp 0.05)))

(define float->integer
  (lambda (x)
    (cond ((equal? x 0.0) 0)
          ((equal? x 1.0) 1)
          ((equal? x 2.0) 2)
          ((equal? x 3.0) 3)
          ((equal? x 4.0) 4)
          ((equal? x 5.0) 5)
          ((equal? x 6.0) 6)
          ((equal? x 7.0) 7)
          ((equal? x 8.0) 8)
          ((equal? x 9.0) 9)
          ((equal? x 10.0) 10)
          ((equal? x 11.0) 11)
          ((equal? x 12.0) 12)
          ((equal? x 13.0) 13)
          ((equal? x 14.0) 14)
          ((equal? x 15.0) 15)
          (else x))))

(define even-load-mom
  (lambda (wood guess-t-sp)
    (* (/ (* (/ (n-load guess-t-sp) 2)
             (unif-load plyform)
             (choose-space stud stud-vec)
             (choose-space wood wale-vec))
          (* 2
             144
             guess-t-sp))
       (- (square (- guess-t-sp
                     (/ (choose-space stud stud-vec)
```

```
                        2)))
          (* (- (/ (n-load guess-t-sp) 2) 1)
             (choose-space stud stud-vec)
             guess-t-sp)))))

(define sum
  (lambda (n)
    (if (= n 0)
        0
        (+ n (sum (- n 1))))))

(define n-load-s
  (lambda (wood guess-t-sp)
    (float->integer (ceiling (/ (- guess-t-sp (depth wood))
                                (choose-space stud stud-vec))))))

(define point-shear
  (lambda (wood guess-t-sp)
    (* (/ (* (/ (unif-load plyform)
                144)
             (choose-space stud stud-vec)
             (choose-space wood wale-vec))
          guess-t-sp)
       (- (* (n-load-s wood guess-t-sp)
             guess-t-sp)
          (* (sum (- (n-load-s wood guess-t-sp) 1))
             (choose-space stud stud-vec))
          (* (n-load-s wood guess-t-sp)
             (depth wood))))))

(define s-actual
  (lambda (wood guess-t-sp)
    (/ (* (point-shear wood guess-t-sp)
          3)
       (* 2
          (depth wood)
          (if (equal? (soldier wood) 'double)
              (* 2 (width wood))
              (width wood))))))

(define s-max
  (lambda (wood guess-t-sp)
    (cond ((s-close-enough? (s-actual wood guess-t-sp) wood)
           guess-t-sp)
          (else
           (s-max wood (improve guess-t-sp))))))

(define s-close-enough?
  (lambda (s-actual wood)
    (< (- (f-v wood)
          s-actual)
       0.01)))
```

```
(module wood)

;;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
;;                    Defining the WOOD Abstract Data Type
;;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

;;The procedure LOAD-WOOD takes as arguments the six user defined vectors,
;;PLY-VEC, STUD-VEC, WALE-VEC, TIe-VEC, DEF-VEC, and CONC-VEC and creates
;;four new vectors called PLYFORM, STUD, WALE and TIE.  These four vectors
;;are the workhorses of the program and are specifically created from the
;;procedures LOAD-PLYFORM, LOAD-STUD, LOAD-WALE and LOAD-TIE.

(define load-wood
  (lambda (ply-vec stud-vec wale-vec tie-vec def-vec conc-vec)
    (load-plyform ply-vec def-vec conc-vec)
    (load-stud stud-vec def-vec conc-vec)
    (load-wale wale-vec def-vec conc-vec)
    (load-tie tie-vec def-vec conc-vec)))

;;The procedure LOAD-PLYFORM takes as arguments three of the user defined
;;vectors PLY-VEC, DEF-VEC, and CONC-VEC and creates the new vector PLYFORM.
;;LOAD-PLYFORM loads each index of PLYFORM with information to be used later
;;design the wall form system.  Each index is described as follows:
;;
;;  - Index 0 : Type of wall form member (i.e., plyform)
;;  - Index 1 : Width (N/A for plyform)
;;  - Index 2 : Depth (plyform thickness)
;;  - Index 3 : Cross-sectional area (N/A for plyform)
;;  - Index 4 : Moment of inertia
;;  - Index 5 : Effective section modulus
;;  - Index 6 : Species of wood (N/A for plyform)
;;  - Index 7 : Grade of wood (N/A for plyform)
;;  - Index 8 : Allowable bending stress (single) (N/A for plyform)
;;  - Index 9 : Allowable bending stress (repetitive)
;;  - Index 10: Allowable tensile stress (N/A for plyform)
;;  - Index 11: Allowable rolling shear stress
;;  - Index 12: Allowable perp. compressive stress (N/A for plyform)
;;  - Index 13: Modulus of elasticity
;;  - Index 14: Lumber size classification (N/A for plyform)
;;  - Index 15: Plyform class
;;  - Index 16: Rolling shear constant
;;  - Index 17: Wood loading
;;  - Index 18: Support conditions
;;  - Index 19: First deflection limit (span-dependent)
;;  - Index 20: Second deflection limit (absolute)
;;  - Index 21: Tie capacity (N/A for plyform)
;;  - Index 22: Direction of face grain relative to span
;;  - Index 23: Spacing (N/A for plyform)
;;  - Index 24: Single or double soldier wale (N/A for plyform)

(define load-plyform
  (lambda (ply-vec def-vec conc-vec)
    (set! plyform (make-wood))
    (vector-set! plyform 0 'plyform)
    (vector-set! plyform 2 (vector-ref ply-vec 2))
    (let ((properties '(i s)))
      (do ((j 0 (+ j 1)))
          ((equal? j 2))
        (vector-set! plyform (+ j 4) (lookup4d (vector-ref ply-vec 0)
                                               (vector-ref ply-vec 1)
                                               (vector-ref ply-vec 2)
                                               (list-ref properties j)
                                               ply-prop))))
    (vector-set! plyform 9 (lookup2d (vector-ref ply-vec 0)
                                     'f-b
                                     ply-stress))
```

```
    (vector-set! plyform 11 (lookup2d (vector-ref ply-vec 0)
                                      'f-v
                                      ply-stress))
    (vector-set! plyform 13 (lookup2d (vector-ref ply-vec 0)
                                      'e
                                      ply-stress))
    (vector-set! plyform 15 (vector-ref ply-vec 0))
    (vector-set! plyform 16 (lookup4d (vector-ref ply-vec 0)
                                      (vector-ref ply-vec 1)
                                      (vector-ref ply-vec 2)
                                      'ib/q
                                      ply-prop))
    (vector-set! plyform 17 (ply-press conc-vec))
    (vector-set! plyform 18 (vector-ref ply-vec 3))
    (vector-set! plyform 19 (vector-ref def-vec 0))
    (vector-set! plyform 20 (vector-ref def-vec 1))
    (vector-set! plyform 22 (vector-ref ply-vec 1))
    plyform))

;;The procedure PLY-PRESS computes the uniform loading created by the concrete
;;across a 12 inch strip of the plyform between the studs.  The plyform is
;;assumed to be acting as a simply supported beam.

(define ply-press
  (lambda (conc-vec)
    (/ (* 12 (cond ((equal? (vector-ref conc-vec 3) '?)
                    (w (vector-ref conc-vec 2)
                       (vector-ref conc-vec 0)
                       (vector-ref conc-vec 1)))
                   (else (vector-ref conc-vec 3))))
       12)))

;;The procedure LOAD-STUD takes as arguments three of the user defined
;;vectors STUD-VEC, DEF-VEC, and CONC-VEC and creates the new vector STUD.
;;LOAD-STUD loads each index of STUD with information to be used later
;;design the wall form system.  Each index is described as follows:
;;
;;  - Index 0 : Type of wall form member (i.e., stud)
;;  - Index 1 : Width
;;  - Index 2 : Depth
;;  - Index 3 : Cross-sectional area
;;  - Index 4 : Moment of inertia
;;  - Index 5 : Effective section modulus
;;  - Index 6 : Species of wood
;;  - Index 7 : Grade of wood
;;  - Index 8 : Allowable bending stress (single)
;;  - Index 9 : Allowable bending stress (repetitive)
;;  - Index 10: Allowable tensile stress
;;  - Index 11: Allowable shear stress
;;  - Index 12: Allowable perp. compressive stress
;;  - Index 13: Modulus of elasticity
;;  - Index 14: Lumber size classification
;;  - Index 15: Plyform class (N/A for studs)
;;  - Index 16: Rolling shear constant (N/A for studs)
;;  - Index 17: Wood loading
;;  - Index 18: Support conditions
;;  - Index 19: First deflection limit (span-dependent)
;;  - Index 20: Second deflection limit (absolute)
;;  - Index 21: Tie capacity (N/A for studs)
;;  - Index 22: Direction of face grain relative to span (N/A for studs)
;;  - Index 23: Spacing
;;  - Index 24: Single or double soldier wale (N/A for studs)

(define load-stud
  (lambda (stud-vec def-vec conc-vec)
    (set! stud (make-wood))
```

```
    (vector-set! stud 0 'stud)
    (let ((properties '(width depth area moi sm)))
      (do ((i 0 (+ i 1)))
          ((equal? i 5))
        (vector-set! stud (+ i 1) (prop-table stud-vec properties i))))
    (vector-set! stud 6 (vector-ref stud-vec 1))
    (vector-set! stud 7 (vector-ref stud-vec 2))
    (stress-table stud-vec stud)
    (vector-set! stud 14 (vector-ref stud-vec 3))
    (vector-set! stud 18 (vector-ref stud-vec 6))
    (vector-set! stud 19 (vector-ref def-vec 0))
    (vector-set! stud 20 (vector-ref def-vec 1))
    (vector-set! stud 23 (vector-ref stud-vec 0))
    stud))

;;The procedure LOAD-WALE takes as arguments three of the user defined
;;vectors WALE-VEC, DEF-VEC, and CONC-VEC and creates the new vector WALE.
;;LOAD-WALE loads each index of WALE with information to be used later
;;design the wall form system.  Each index is described as follows:
;;
;;  - Index 0 : Type of wall form member (i.e., wale)
;;  - Index 1 : Width
;;  - Index 2 : Depth
;;  - Index 3 : Cross-sectional area
;;  - Index 4 : Moment of inertia
;;  - Index 5 : Effective section modulus
;;  - Index 6 : Species of wood
;;  - Index 7 : Grade of wood
;;  - Index 8 : Allowable bending stress (single)
;;  - Index 9 : Allowable bending stress (repetitive)
;;  - Index 10: Allowable tensile stress
;;  - Index 11: Allowable shear stress
;;  - Index 12: Allowable perp. compressive stress
;;  - Index 13: Modulus of elasticity
;;  - Index 14: Lumber size classification
;;  - Index 15: Plyform class (N/A for wales)
;;  - Index 16: Rolling shear constant (N/A for wales)
;;  - Index 17: Wood loading
;;  - Index 18: Support conditions
;;  - Index 19: First deflection limit (span-dependent)
;;  - Index 20: Second deflection limit (absolute)
;;  - Index 21: Tie capacity (N/A for wales)
;;  - Index 22: Direction of face grain relative to span (N/A for wales)
;;  - Index 23: Spacing
;;  - Index 24: Single or double soldier wale

(define load-wale
  (lambda (wale-vec def-vec conc-vec)
    (set! wale (make-wood))
    (vector-set! wale 0 'wale)
    (let ((properties '(width depth)))
      (do ((i 0 (+ i 1)))
          ((equal? i 2))
        (vector-set! wale (+ i 1) (prop-table wale-vec properties i))))
    (let ((properties '(area moi sm)))
      (do ((i 0 (+ i 1)))
          ((equal? i 3))
        (vector-set! wale (+ i 3) (if (equal? (vector-ref wale-vec 8) 'double)
                                      (* 2
                                         (prop-table wale-vec properties i))
                                      (prop-table wale-vec properties i)))))
    (vector-set! wale 6 (vector-ref wale-vec 1))
    (vector-set! wale 7 (vector-ref wale-vec 2))
    (stress-table wale-vec wale)
    (vector-set! wale 14 (vector-ref stud-vec 3))
    (vector-set! wale 18 (vector-ref wale-vec 6))
```

```
    (vector-set! wale 19 (vector-ref def-vec 0))
    (vector-set! wale 20 (vector-ref def-vec 1))
    (vector-set! wale 23 (vector-ref wale-vec 0))
    (vector-set! wale 24 (vector-ref wale-vec 8))
    wale))

(define prop-table
  (lambda (input lst i)
    (lookup2d (lookup2d (vector-ref input 4)
                        (vector-ref input 5)
                        nominal)
              (list-ref lst i)
              lum-prop)))

(define stress-table
  (lambda (input wood)
    (cond ((empty? input)
           (let ((properties '(f-b-sing f-b-rep f-t f-v f-c-perp e)))
             (do ((i 0 (+ i 1)))
                 ((equal? i 6))
               (vector-set! wood (+ i 8) (lookup4d (vector-ref input 1)
                                                   (vector-ref input 2)
                                                   (vector-ref input 3)
                                                   (list-ref properties i)
                                                   lum-stress)))))
          (else (vector-set! wood 9 (vector-ref input 9))
                (vector-set! wood 11 (vector-ref input 12 ))
                (vector-set! wood 12 (vector-ref input 11))
                (vector-set! wood 13 (vector-ref input 10))))))

(define empty?
  (lambda (input)
    (or (equal? (vector-ref input 9) '?)
        (equal? (vector-ref input 10) '?)
        (equal? (vector-ref input 11) '?)
        (equal? (vector-ref input 12) '?))))

;;The procedure LOAD-TIE takes as arguments three of the user defined
;;vectors TIE-VEC, DEF-VEC, and CONC-VEC and creates the new vector TIE.
;;LOAD-TIE loads each index of TIE with information to be used later
;;design the wall form system.  Each index is described as follows:
;;
;;  - Index 0 : Type of wall form member (i.e., tie)
;;  - Index 1 : Width (for TIE, shortest dimension of tie plate)
;;  - Index 2 : Depth (for TIE, longest dimension of tie plate)
;;  - Index 3 : Tie plate diameter, if circular
;;  - Index 4 : Moment of inertia (N/A for ties)
;;  - Index 5 : Effective section modulus (N/A for ties)
;;  - Index 6 : Shape of tie plate or washer
;;  - Index 7 : Grade of wood (N/A for ties)
;;  - Index 8 : Allowable bending stress (single) (N/A for ties)
;;  - Index 9 : Allowable bending stress (repetitive) (N/A for ties)
;;  - Index 10: Allowable tensile stress (N/A for ties)
;;  - Index 11: Allowable shear stress (N/A for ties)
;;  - Index 12: Allowable perp. compressive stress (N/A for ties)
;;  - Index 13: Allowable para. compressive stress (N/A for ties)
;;  - Index 14: Modulus of elasticity (N/A for ties)
;;  - Index 15: Plyform class (N/A for ties)
;;  - Index 16: Rolling shear constant (N/A for ties)
;;  - Index 17: Wood loading (Tie loading)
;;  - Index 18: Support conditions (N/A for ties)
;;  - Index 19: First deflection limit (span-dependent) (N/A for ties)
;;  - Index 20: Second deflection limit (absolute) (N/A for ties)
;;  - Index 21: Tie capacity
;;  - Index 22: Direction of face grain relative to span (N/A for ties)
;;  - Index 23: Spacing
```

```
;;  - Index 24: Single or double soldier wale (N/A for ties)

(define load-tie
  (lambda (tie-vec def-vec conc-vec)
    (set! tie (make-wood))
    (vector-set! tie 0 'tie)
    (vector-set! tie 1 (vector-ref tie-vec 3))
    (vector-set! tie 2 (vector-ref tie-vec 4))
    (vector-set! tie 3 (vector-ref tie-vec 5))
    (vector-set! tie 6 (vector-ref tie-vec 2))
    (vector-set! tie 21 (vector-ref tie-vec 1))
    (vector-set! tie 23 (vector-ref tie-vec 0))
    tie))

;;The procedure MAKE-WOOD initializes a vector of length 25 and fills each
;;index with a question mark.

(define make-wood
  (lambda ()
    (make-vector 25 '?)))

;;The following series of procedures simply call the content of each index
;;of the four vectors PLYFORM, STUD, WALE, or TIE.

(define type
  (lambda (wood)
    (vector-ref wood 0)))

(define width
  (lambda (wood)
    (vector-ref wood 1)))

(define depth
  (lambda (wood)
    (vector-ref wood 2)))

(define area
  (lambda (wood)
    (vector-ref wood 3)))

(define moi
  (lambda (wood)
    (vector-ref wood 4)))

(define sm
  (lambda (wood)
    (vector-ref wood 5)))

(define species
  (lambda (wood)
    (vector-ref wood 6)))

(define grade
  (lambda (wood)
    (vector-ref wood 7)))

(define f-b
  (lambda (wood)
    (cond ((equal? (type wood) 'stud)
           (if (equal? (vector-ref stud-vec 6) 'single)
               (f-b-sing wood)
               (f-b-rep wood)))
          ((equal? (type wood) 'wale)
           (if (equal? (vector-ref wale-vec 6) 'single)
               (f-b-sing wood)
               (f-b-rep wood)))
```

```
          (else (f-b-rep wood)))))

(define f-b-sing
  (lambda (wood)
    (vector-ref wood 8)))

(define f-b-rep
  (lambda (wood)
    (vector-ref wood 9)))

(define f-t
  (lambda (wood)
    (vector-ref wood 10)))

(define f-v
  (lambda (wood)
    (vector-ref wood 11)))

(define f-c-perp
  (lambda (wood)
    (vector-ref wood 12)))

(define e
  (lambda (wood)
    (vector-ref wood 13)))

(define size-class
  (lambda (wood)
    (vector-ref wood 14)))

(define class
  (lambda (wood)
    (vector-ref wood 15)))

(define ib/q
  (lambda (wood)
    (vector-ref wood 16)))

(define unif-load
  (lambda (wood)
    (vector-ref wood 17)))

(define sup-cond
  (lambda (wood)
    (vector-ref wood 18)))

(define deflection-lim1
  (lambda (wood)
    (vector-ref wood 19)))

(define deflection-lim2
  (lambda (wood)
    (vector-ref wood 20)))

(define tie-cap
  (lambda (wood)
    (vector-ref wood 21)))

(define face-grain
  (lambda (wood)
    (vector-ref wood 22)))

(define spacing
  (lambda (wood)
    (vector-ref wood 23)))
```

```
(define soldier
  (lambda (wood)
    (vector-ref wood 24)))

(define diameter
  (lambda (wood)
    (vector-ref wood 3)))
```

```
(module table)

;;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
;;                                ADT TABLE
;;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

;;-----------------------------------------------------------------------------
;;  One-dimensional Table
;;-----------------------------------------------------------------------------

(define make-table
  (lambda ()
    (list 'table)))

(define table?
  (lambda (table)
    (if (equal? (car table) 'table)
        #t
        #f)))

(define lookup
  (lambda (key table)
    (let ((record (find-in-list key (cdr table))))
      (cond ((null? record) '())
            (else (cdr record))))))

(define find-in-list
  (lambda (key record)
    (cond ((null? record) '())
          ((equal? key (caar record)) (car record))
          (else (find-in-list key (cdr record))))))

(define insert!
  (lambda (key value table)
    (let ((record (find-in-list key (cdr table))))
      (cond ((null? record)
             (set-cdr! table (cons (cons key value)
                                   (cdr table))))
            (else (set-cdr! record value))))))

;;-----------------------------------------------------------------------------
;;  Two-dimensional Table
;;-----------------------------------------------------------------------------

(define make-table2d
  (lambda ()
    (list 'table)))

(define lookup2d
  (lambda (key1 key2 table)
    (cond ((null? (lookup key1
                          table))
           '())
          (else (lookup key2
                        (car (lookup key1
                                     table)))))))

(define insert2d!
  (lambda (key1 key2 value table)
    (cond ((null? (lookup key1
                          table))
           (insert! key1
                    (list (make-table)) table)
           (insert! key2
                    value
                    (cadadr table)))
```

```
          (else (insert! key2
                         value
                         (car (lookup key1
                                      table)))))))

;;-----------------------------------------------------------------------
;;  Three-dimensional Table
;;-----------------------------------------------------------------------

(define make-table3d
  (lambda ()
    (list 'table)))

(define lookup3d
  (lambda (key1 key2 key3 table)
    (cond ((or (null? (lookup key1 table))
               (null? (lookup2d key1 key2 table)))
           '())
          (else (lookup key3
                        (car (lookup key2
                                     (car (lookup key1
                                                  table)))))))))

(define insert3d!
  (lambda (key1 key2 key3 value table)
    (cond ((null? (lookup key1
                          table))
           (insert! key1
                    (list (make-table))
                    table)
           (insert! key2
                    (list (make-table))
                    (cadadr table))
           (insert! key3
                    value
                    (cadadr (cadadr table))))
          ((null? (lookup key2
                          (car (lookup key1
                                       table))))
           (insert2d! key2
                      key3
                      value
                      (car (lookup key1 table))))
          (else (insert! key3
                         value
                         (car (lookup key2
                                      (car (lookup key1
                                                   table)))))))))

;;-----------------------------------------------------------------------
;;  Four-dimensional Table
;;-----------------------------------------------------------------------

(define make-table4d
  (lambda ()
    (list 'table)))

(define lookup4d
  (lambda (key1 key2 key3 key4 table)
    (cond ((or (null? (lookup key1 table))
               (null? (lookup2d key1 key2 table))
               (null? (lookup3d key1 key2 key3 table)))
           '())
          (else (lookup key4
                        (car (lookup key3
                                     (car (lookup key2
```

```
                                                    (car (lookup key1
                                                                 table)))))))))))

(define insert4d!
  (lambda (key1 key2 key3 key4 value table)
    (cond ((null? (lookup key1
                          table))
           (insert! key1
                    (list (make-table))
                    table)
           (insert! key2
                    (list (make-table))
                    (cadadr table))
           (insert! key3
                    (list (make-table))
                    (cadadr (cadadr table)))
           (insert! key4
                    value
                    (cadadr (cadadr (cadadr table)))))
          ((null? (lookup2d key1
                            key2
                            table))
           (insert3d! key2
                      key3
                      key4
                      value
                      (car (lookup key1 table))))
          ((null? (lookup3d key1
                            key2
                            key3
                            table))
           (insert2d! key3
                      key4
                      value
                      (car (lookup key2
                                   (car (lookup key1
                                                table))))))
          (else (insert! key4
                         value
                         (car (lookup key3
                                      (car (lookup key2
                                                   (car (lookup key1
                                                                table)))))))))))
```

```
(module cfde)

;;-----------------------------------------------------------------------
;;Concrete form design equations to find spacing
;;-----------------------------------------------------------------------

;;DESIGN CONDITION:  Bending,  SUPPORT CONDITION:  1 span
(define b1
  (lambda (f-b S w)
    (* 9.8 (expt (/ (* f-b S) w) (/ 1 2)))))

;;DESIGN CONDITION:  Bending,  SUPPORT CONDITION:  2 span
(define b2
  (lambda (f-b S w)
    (* 9.8 (expt (/ (* f-b S) w) (/ 1 2)))))

;;DESIGN CONDITION:  Bending,  SUPPORT CONDITION:  3 span
(define b3
  (lambda (f-b S w)
    (* 10.95 (expt (/ (* f-b S) w) (/ 1 2)))))

;;DESIGN CONDITION:  Shear,  SUPPORT CONDITION:  1 span  (Plywood)
(define s1
  (lambda (f-v ib/q w h)
    (+ (* 24 (/ (* f-v ib/q) w)) (* 2 h))))

;;DESIGN CONDITION:  Shear,  SUPPORT CONDITION:  2 span  (Plywood)
(define s2
  (lambda (f-v ib/q w h)
    (+ (* 19.2 (/ (* f-v ib/q) w)) (* 2 h))))

;;DESIGN CONDITION:  Shear,  SUPPORT CONDITION:  3 span  (Plywood)
(define s3
  (lambda (f-v ib/q w h)
    (+ (* 20 (/ (* f-v ib/q) w)) (* 2 h))))

;;DESIGN CONDITION:  Shear,  SUPPORT CONDITION:  1 span  (Lumber)
(define s4
  (lambda (f-v area w h)
    (+ (* 16 (/ (* f-v area) w)) (* 2 h))))

;;DESIGN CONDITION:  Shear,  SUPPORT CONDITION:  2 span  (Lumber)
(define s5
  (lambda (f-v area w h)
    (+ (* 12.8 (/ (* f-v area) w)) (* 2 h))))

;;DESIGN CONDITION:  Shear,  SUPPORT CONDITION:  3 span  (Lumber)
(define s6
  (lambda (f-v area w h)
    (+ (* 13.3 (/ (* f-v area) w)) (* 2 h))))

;;DESIGN CONDITION:  Deflection - 1/180,  SUPPORT CONDITION:  1 span
(define d1/180
  (lambda (E I w)
    (* 1.72 (expt (/ (* E I) w) (/ 1 3)))))

;;DESIGN CONDITION:  Deflection - 1/180,  SUPPORT CONDITION:  2 span
(define d2/180
  (lambda (E I w)
    (* 2.31 (expt (/ (* E I) w) (/ 1 3)))))

;;DESIGN CONDITION:  Deflection - 1/180,  SUPPORT CONDITION:  3 span
(define d3/180
  (lambda (E I w)
    (* 2.13 (expt (/ (* E I) w) (/ 1 3)))))
```

```
;;DESIGN CONDITION:  Deflection - 1/240,  SUPPORT CONDITION:  1 span
(define d1/240
  (lambda (E I w)
    (* 1.57 (expt (/ (* E I) w) (/ 1 3)))))

;;DESIGN CONDITION:  Deflection - 1/240,  SUPPORT CONDITION:  2 span
(define d2/240
  (lambda (E I w)
    (* 2.10 (expt (/ (* E I) w) (/ 1 3)))))

;;DESIGN CONDITION:  Deflection - 1/240,  SUPPORT CONDITION:  3 span
(define d3/240
  (lambda (E I w)
    (* 1.94 (expt (/ (* E I) w) (/ 1 3)))))

;;DESIGN CONDITION:  Deflection - 1/360,  SUPPORT CONDITION:  1 span
(define d1/360
  (lambda (E I w)
    (* 1.37 (expt (/ (* E I) w) (/ 1 3)))))

;;DESIGN CONDITION:  Deflection - 1/360,  SUPPORT CONDITION:  2 span
(define d2/360
  (lambda (E I w)
    (* 1.83 (expt (/ (* E I) w) (/ 1 3)))))

;;DESIGN CONDITION:  Deflection - 1/360,  SUPPORT CONDITION:  3 span
(define d3/360
  (lambda (E I w)
    (* 1.69 (expt (/ (* E I) w) (/ 1 3)))))

;;DESIGN CONDITION:  Deflection - 1/8 inch  SUPPORT CONDITION:  1 span
(define d125mil
  (lambda (E I w)
    (* 3.84 (expt (/ (* E I) w) (/ 1 4)))))

;;DESIGN CONDITION:  Deflection - 1/16 inch  SUPPORT CONDITION:  1 span
(define d0625mil
  (lambda (E I w)
    (* 3.23 (expt (/ (* E I) w) (/ 1 4)))))

;;-----------------------------------------------------------------------
;;Create the Concrete form design equations Table
;;-----------------------------------------------------------------------

(define cfde (make-table2d))

(insert2d! 'bending 'span-1 b1 cfde)
(insert2d! 'bending 'span-2 b2 cfde)
(insert2d! 'bending 'span-3 b3 cfde)
(insert2d! 'shearP 'span-1 s1 cfde)
(insert2d! 'shearP 'span-2 s2 cfde)
(insert2d! 'shearP 'span-3 s3 cfde)
(insert2d! 'shearL 'span-1 s4 cfde)
(insert2d! 'shearL 'span-2 s5 cfde)
(insert2d! 'shearL 'span-3 s6 cfde)
(insert2d! 'span/180 'span-1 d1/180 cfde)
(insert2d! 'span/180 'span-2 d2/180 cfde)
(insert2d! 'span/180 'span-3 d3/180 cfde)
(insert2d! 'span/240 'span-1 d1/240 cfde)
(insert2d! 'span/240 'span-2 d2/240 cfde)
(insert2d! 'span/240 'span-3 d3/240 cfde)
(insert2d! 'span/360 'span-1 d1/360 cfde)
(insert2d! 'span/360 'span-2 d2/360 cfde)
(insert2d! 'span/360 'span-3 d3/360 cfde)
(insert2d! 'span1/8 'span-1 d125mil cfde)
(insert2d! 'span1/8 'span-2 d125mil cfde)
```

```
(insert2d! 'span1/8 'span-3 d125mil cfde)
(insert2d! 'span1/16 'span-1 d0625mil cfde)
(insert2d! 'span1/16 'span-2 d0625mil cfde)
(insert2d! 'span1/16 'span-3 d0625mil cfde)
```

```
(module prop)

;;-----------------------------------------------------------------------------
;;Create the Table for Section Propteries for Plyform Class I and Class II,
;;                      and Structural I Plyform
;;-----------------------------------------------------------------------------

(define ply-prop (make-table4d))

;;Entries into this table will be of the format
;;      (1)  Class or plywood grade
;;           (i.e., "c1" for Class I; "c2" for Class II; and "s1" for Structural I).
;;      (2)  Direction of span relative to face grain.
;;           (i.e., "para" for a plyform with the span running parallel to the
;;           face grain and "perp" for a plyform with the span running
;;           perpendicular to the face grain).
;;      (3)  Thickness of the Plyform.
;;      (4)  Parameter you are looking for
;;           (i.e., "i" for Moment of Inertia; "s" for section
;;           modulus; and "ib/q" for rolling shear constant.)
;;      (5)  Name of the table.

(insert4d! 'c1 'para 0.46875 'i 0.066 ply-prop)
(insert4d! 'c1 'para 0.5 'i 0.077 ply-prop)
(insert4d! 'c1 'para 0.59375 'i 0.115 ply-prop)
(insert4d! 'c1 'para 0.625 'i 0.130 ply-prop)
(insert4d! 'c1 'para 0.71875 'i 0.180 ply-prop)
(insert4d! 'c1 'para 0.75 'i 0.199 ply-prop)
(insert4d! 'c1 'para 0.875 'i 0.280 ply-prop)
(insert4d! 'c1 'para 1.0 'i 0.427 ply-prop)
(insert4d! 'c1 'para 1.125 'i 0.554 ply-prop)
(insert4d! 'c1 'para 0.46875 's 0.245 ply-prop)
(insert4d! 'c1 'para 0.5 's 0.268 ply-prop)
(insert4d! 'c1 'para 0.59375 's 0.335 ply-prop)
(insert4d! 'c1 'para 0.625 's 0.358 ply-prop)
(insert4d! 'c1 'para 0.71875 's 0.431 ply-prop)
(insert4d! 'c1 'para 0.75 's 0.455 ply-prop)
(insert4d! 'c1 'para 0.875 's '0.553 ply-prop)
(insert4d! 'c1 'para 1.0 's 0.737 ply-prop)
(insert4d! 'c1 'para 1.125 's 0.849 ply-prop)
(insert4d! 'c1 'para 0.46875 'ib/q 5.021 ply-prop)
(insert4d! 'c1 'para 0.5 'ib/q 5.418 ply-prop)
(insert4d! 'c1 'para 0.59375 'ib/q 6.383 ply-prop)
(insert4d! 'c1 'para 0.625 'ib/q 6.791 ply-prop)
(insert4d! 'c1 'para 0.71875 'ib/q 7.900 ply-prop)
(insert4d! 'c1 'para 0.75 'ib/q 8.299 ply-prop)
(insert4d! 'c1 'para 0.875 'ib/q 8.668 ply-prop)
(insert4d! 'c1 'para 1.0 'ib/q 9.373 ply-prop)
(insert4d! 'c1 'para 1.125 'ib/q 10.430 ply-prop)
(insert4d! 'c1 'perp 0.46875 'i 0.027 ply-prop)
(insert4d! 'c1 'perp 0.5 'i 0.035 ply-prop)
(insert4d! 'c1 'perp 0.59375 'i 0.052 ply-prop)
(insert4d! 'c1 'perp 0.625 'i 0.064 ply-prop)
(insert4d! 'c1 'perp 0.71875 'i 0.115 ply-prop)
(insert4d! 'c1 'perp 0.75 'i 0.136 ply-prop)
(insert4d! 'c1 'perp 0.875 'i 0.230 ply-prop)
(insert4d! 'c1 'perp 1.0 'i 0.373 ply-prop)
(insert4d! 'c1 'perp 1.125 'i 0.530 ply-prop)
(insert4d! 'c1 'perp 0.46875 's 0.139 ply-prop)
(insert4d! 'c1 'perp 0.5 's 0.167 ply-prop)
(insert4d! 'c1 'perp 0.59375 's 0.215 ply-prop)
(insert4d! 'c1 'perp 0.625 's 0.250 ply-prop)
(insert4d! 'c1 'perp 0.71875 's 0.370 ply-prop)
(insert4d! 'c1 'perp 0.75 's 0.415 ply-prop)
(insert4d! 'c1 'perp 0.875 's 0.581 ply-prop)
(insert4d! 'c1 'perp 1.0 's 0.798 ply-prop)
```

```
(insert4d! 'c1 'perp 1.125 's 0.986 ply-prop)
(insert4d! 'c1 'perp 0.46875 'ib/q 2.816 ply-prop)
(insert4d! 'c1 'perp 0.5 'ib/q 2.919 ply-prop)
(insert4d! 'c1 'perp 0.59375 'ib/q 3.623 ply-prop)
(insert4d! 'c1 'perp 0.625 'ib/q 3.692 ply-prop)
(insert4d! 'c1 'perp 0.71875 'ib/q 4.555 ply-prop)
(insert4d! 'c1 'perp 0.75 'ib/q 4.565 ply-prop)
(insert4d! 'c1 'perp 0.875 'ib/q 5.418 ply-prop)
(insert4d! 'c1 'perp 1.0 'ib/q 7.242 ply-prop)
(insert4d! 'c1 'perp 1.125 'ib/q 8.566 ply-prop)
(insert4d! 'c2 'para 0.46875 'i 0.063 ply-prop)
(insert4d! 'c2 'para 0.5 'i 0.075 ply-prop)
(insert4d! 'c2 'para 0.59375 'i 0.115 ply-prop)
(insert4d! 'c2 'para 0.625 'i 0.130 ply-prop)
(insert4d! 'c2 'para 0.71875 'i 0.180 ply-prop)
(insert4d! 'c2 'para 0.75 'i 0.198 ply-prop)
(insert4d! 'c2 'para 0.875 'i 0.280 ply-prop)
(insert4d! 'c2 'para 1.0 'i 0.421 ply-prop)
(insert4d! 'c2 'para 1.125 'i 0.566 ply-prop)
(insert4d! 'c2 'para 0.46875 's 0.245 ply-prop)
(insert4d! 'c2 'para 0.5 's 0.267 ply-prop)
(insert4d! 'c2 'para 0.59375 's 0.334 ply-prop)
(insert4d! 'c2 'para 0.625 's 0.357 ply-prop)
(insert4d! 'c2 'para 0.71875 's 0.431 ply-prop)
(insert4d! 'c2 'para 0.75 's 0.455 ply-prop)
(insert4d! 'c2 'para 0.875 's 0.553 ply-prop)
(insert4d! 'c2 'para 1.0 's 0.754 ply-prop)
(insert4d! 'c2 'para 1.125 's 0.869 ply-prop)
(insert4d! 'c2 'para 0.46875 'ib/q 4.751 ply-prop)
(insert4d! 'c2 'para 0.5 'ib/q 5.143 ply-prop)
(insert4d! 'c2 'para 0.59375 'ib/q 6.072 ply-prop)
(insert4d! 'c2 'para 0.625 'ib/q 6.463 ply-prop)
(insert4d! 'c2 'para 0.71875 'ib/q 7.510 ply-prop)
(insert4d! 'c2 'para 0.75 'ib/q 7.892 ply-prop)
(insert4d! 'c2 'para 0.875 'ib/q 8.031 ply-prop)
(insert4d! 'c2 'para 1.0 'ib/q 8.614 ply-prop)
(insert4d! 'c2 'para 1.125 'ib/q 9.571 ply-prop)
(insert4d! 'c2 'perp 0.46875 'i 0.017 ply-prop)
(insert4d! 'c2 'perp 0.5 'i 0.029 ply-prop)
(insert4d! 'c2 'perp 0.59375 'i 0.042 ply-prop)
(insert4d! 'c2 'perp 0.625 'i 0.053 ply-prop)
(insert4d! 'c2 'perp 0.71875 'i 0.093 ply-prop)
(insert4d! 'c2 'perp 0.75 'i 0.111 ply-prop)
(insert4d! 'c2 'perp 0.875 'i 0.186 ply-prop)
(insert4d! 'c2 'perp 1.0 'i 0.301 ply-prop)
(insert4d! 'c2 'perp 1.125 'i 0.429 ply-prop)
(insert4d! 'c2 'perp 0.46875 's 0.147 ply-prop)
(insert4d! 'c2 'perp 0.5 's 0.182 ply-prop)
(insert4d! 'c2 'perp 0.59375 's 0.276 ply-prop)
(insert4d! 'c2 'perp 0.625 's 0.320 ply-prop)
(insert4d! 'c2 'perp 0.71875 's 0.473 ply-prop)
(insert4d! 'c2 'perp 0.75 's 0.530 ply-prop)
(insert4d! 'c2 'perp 0.875 's 0.742 ply-prop)
(insert4d! 'c2 'perp 1.0 's 1.020 ply-prop)
(insert4d! 'c2 'perp 1.125 's 1.260 ply-prop)
(insert4d! 'c2 'perp 0.46875 'ib/q 2.406 ply-prop)
(insert4d! 'c2 'perp 0.5 'ib/q 2.671 ply-prop)
(insert4d! 'c2 'perp 0.59375 'ib/q 3.611 ply-prop)
(insert4d! 'c2 'perp 0.625 'ib/q 3.890 ply-prop)
(insert4d! 'c2 'perp 0.71875 'ib/q 4.545 ply-prop)
(insert4d! 'c2 'perp 0.75 'ib/q 4.814 ply-prop)
(insert4d! 'c2 'perp 0.875 'ib/q 5.716 ply-prop)
(insert4d! 'c2 'perp 1.0 'ib/q 7.642 ply-prop)
(insert4d! 'c2 'perp 1.125 'ib/q 9.032 ply-prop)
(insert4d! 's1 'para 0.46875 'i 0.067 ply-prop)
(insert4d! 's1 'para 0.5 'i 0.078 ply-prop)
```

```
(insert4d! 's1 'para 0.59375 'i 0.116 ply-prop)
(insert4d! 's1 'para 0.625 'i 0.131 ply-prop)
(insert4d! 's1 'para 0.71875 'i 0.183 ply-prop)
(insert4d! 's1 'para 0.75 'i 0.202 ply-prop)
(insert4d! 's1 'para 0.875 'i 0.288 ply-prop)
(insert4d! 's1 'para 1.0 'i 0.479 ply-prop)
(insert4d! 's1 'para 1.125 'i 0.623 ply-prop)
(insert4d! 's1 'para 0.46875 's 0.247 ply-prop)
(insert4d! 's1 'para 0.5 's 0.271 ply-prop)
(insert4d! 's1 'para 0.59375 's 0.338 ply-prop)
(insert4d! 's1 'para 0.625 's 0.361 ply-prop)
(insert4d! 's1 'para 0.71875 's 0.439 ply-prop)
(insert4d! 's1 'para 0.75 's 0.464 ply-prop)
(insert4d! 's1 'para 0.875 's 0.569 ply-prop)
(insert4d! 's1 'para 1.0 's 0.827 ply-prop)
(insert4d! 's1 'para 1.125 's 0.955 ply-prop)
(insert4d! 's1 'para 0.46875 'ib/q 4.783 ply-prop)
(insert4d! 's1 'para 0.5 'ib/q 5.166 ply-prop)
(insert4d! 's1 'para 0.59375 'ib/q 6.124 ply-prop)
(insert4d! 's1 'para 0.625 'ib/q 6.526 ply-prop)
(insert4d! 's1 'para 0.71875 'ib/q 7.540 ply-prop)
(insert4d! 's1 'para 0.75 'ib/q 7.926 ply-prop)
(insert4d! 's1 'para 0.875 'ib/q 7.539 ply-prop)
(insert4d! 's1 'para 1.0 'ib/q 7.978 ply-prop)
(insert4d! 's1 'para 1.125 'ib/q 8.841 ply-prop)
(insert4d! 's1 'perp 0.46875 'i 0.032 ply-prop)
(insert4d! 's1 'perp 0.5 'i 0.042 ply-prop)
(insert4d! 's1 'perp 0.59375 'i 0.061 ply-prop)
(insert4d! 's1 'perp 0.625 'i 0.077 ply-prop)
(insert4d! 's1 'perp 0.71875 'i 0.137 ply-prop)
(insert4d! 's1 'perp 0.75 'i 0.162 ply-prop)
(insert4d! 's1 'perp 0.875 'i 0.275 ply-prop)
(insert4d! 's1 'perp 1.0 'i 0.445 ply-prop)
(insert4d! 's1 'perp 1.125 'i 0.634 ply-prop)
(insert4d! 's1 'perp 0.46875 's 0.190 ply-prop)
(insert4d! 's1 'perp 0.5 's 0.229 ply-prop)
(insert4d! 's1 'perp 0.59375 's 0.295 ply-prop)
(insert4d! 's1 'perp 0.625 's 0.343 ply-prop)
(insert4d! 's1 'perp 0.71875 's 0.508 ply-prop)
(insert4d! 's1 'perp 0.75 's 0.570 ply-prop)
(insert4d! 's1 'perp 0.875 's 0.798 ply-prop)
(insert4d! 's1 'perp 1.0 's 1.098 ply-prop)
(insert4d! 's1 'perp 1.125 's 1.356 ply-prop)
(insert4d! 's1 'perp 0.46875 'ib/q 2.807 ply-prop)
(insert4d! 's1 'perp 0.5 'ib/q 3.076 ply-prop)
(insert4d! 's1 'perp 0.59375 'ib/q 3.609 ply-prop)
(insert4d! 's1 'perp 0.625 'ib/q 3.887 ply-prop)
(insert4d! 's1 'perp 0.71875 'ib/q 4.544 ply-prop)
(insert4d! 's1 'perp 0.75 'ib/q 4.812 ply-prop)
(insert4d! 's1 'perp 0.875 'ib/q 6.242 ply-prop)
(insert4d! 's1 'perp 1.0 'ib/q 7.639 ply-prop)
(insert4d! 's1 'perp 1.125 'ib/q 9.031 ply-prop)

;;-----------------------------------------------------------------------
;;Create the Table for Plywood Stresses
;;-----------------------------------------------------------------------

(define ply-stress (make-table2d))

(insert2d! 'c1 'e '1650000 ply-stress)
(insert2d! 'c2 'e '1430000 ply-stress)
(insert2d! 's1 'e '1650000 ply-stress)
(insert2d! 'c1 'f-b '1930 ply-stress)
(insert2d! 'c2 'f-b '1330 ply-stress)
(insert2d! 's1 'f-b '1930 ply-stress)
(insert2d! 'c1 'f-v '72 ply-stress)
```

```
(insert2d! 'c2 'f-v '72 ply-stress)
(insert2d! 's1 'f-v '102 ply-stress)

;;-----------------------------------------------------------------------
;;Create the Table for Properties of American Standard Board, Plank,
;;                      Dimension and Timber Sizes Commonly Used for
;;                      Form Construction
;;-----------------------------------------------------------------------

(define lum-prop (make-table2d))

(insert2d! 'n2x3 'area 3.75 lum-prop)
(insert2d! 'n2x4 'area 5.25 lum-prop)
(insert2d! 'n2x5 'area 6.75 lum-prop)
(insert2d! 'n2x6 'area 8.25 lum-prop)
(insert2d! 'n2x8 'area 10.88 lum-prop)
(insert2d! 'n2x10 'area 13.88 lum-prop)
(insert2d! 'n2x12 'area 16.88 lum-prop)
(insert2d! 'n2x3 'moi 1.953 lum-prop)
(insert2d! 'n2x4 'moi 5.359 lum-prop)
(insert2d! 'n2x5 'moi 11.39 lum-prop)
(insert2d! 'n2x6 'moi 20.8 lum-prop)
(insert2d! 'n2x8 'moi 47.64 lum-prop)
(insert2d! 'n2x10 'moi 98.93 lum-prop)
(insert2d! 'n2x12 'moi 178.0 lum-prop)
(insert2d! 'n2x3 'sm 1.563 lum-prop)
(insert2d! 'n2x4 'sm 3.063 lum-prop)
(insert2d! 'n2x5 'sm 5.063 lum-prop)
(insert2d! 'n2x6 'sm 7.563 lum-prop)
(insert2d! 'n2x8 'sm 13.14 lum-prop)
(insert2d! 'n2x10 'sm 21.39 lum-prop)
(insert2d! 'n2x12 'sm 31.64 lum-prop)
(insert2d! 'n2x3 'width 1.5 lum-prop)
(insert2d! 'n2x4 'width 1.5 lum-prop)
(insert2d! 'n2x5 'width 1.5 lum-prop)
(insert2d! 'n2x6 'width 1.5 lum-prop)
(insert2d! 'n2x8 'width 1.5 lum-prop)
(insert2d! 'n2x10 'width 1.5 lum-prop)
(insert2d! 'n2x12 'width 1.5 lum-prop)
(insert2d! 'n2x3 'depth 2.5 lum-prop)
(insert2d! 'n2x4 'depth 3.5 lum-prop)
(insert2d! 'n2x5 'depth 4.5 lum-prop)
(insert2d! 'n2x6 'depth 5.5 lum-prop)
(insert2d! 'n2x8 'depth 7.25 lum-prop)
(insert2d! 'n2x10 'depth 9.25 lum-prop)
(insert2d! 'n2x12 'depth 11.25 lum-prop)

(insert2d! 'n3x2 'area 3.75 lum-prop)
(insert2d! 'n3x4 'area 8.75 lum-prop)
(insert2d! 'n3x5 'area 11.25 lum-prop)
(insert2d! 'n3x6 'area 13.75 lum-prop)
(insert2d! 'n3x8 'area 18.12 lum-prop)
(insert2d! 'n3x10 'area 23.12 lum-prop)
(insert2d! 'n3x12 'area 28.12 lum-prop)
(insert2d! 'n3x2 'moi .7031 lum-prop)
(insert2d! 'n3x4 'moi 8.932 lum-prop)
(insert2d! 'n3x5 'moi 18.98 lum-prop)
(insert2d! 'n3x6 'moi 34.66 lum-prop)
(insert2d! 'n3x8 'moi 79.39 lum-prop)
(insert2d! 'n3x10 'moi 164.9 lum-prop)
(insert2d! 'n3x12 'moi 296.6 lum-prop)
(insert2d! 'n3x2 'sm .9375 lum-prop)
(insert2d! 'n3x4 'sm 5.104 lum-prop)
(insert2d! 'n3x5 'sm 8.438 lum-prop)
(insert2d! 'n3x6 'sm 12.6 lum-prop)
(insert2d! 'n3x8 'sm 21.9 lum-prop)
```

```
(insert2d! 'n3x10 'sm 35.65 lum-prop)
(insert2d! 'n3x12 'sm 52.73 lum-prop)
(insert2d! 'n3x2 'width 2.5 lum-prop)
(insert2d! 'n3x4 'width 2.5 lum-prop)
(insert2d! 'n3x5 'width 2.5 lum-prop)
(insert2d! 'n3x6 'width 2.5 lum-prop)
(insert2d! 'n3x8 'width 2.5 lum-prop)
(insert2d! 'n3x10 'width 2.5 lum-prop)
(insert2d! 'n3x12 'width 2.5 lum-prop)
(insert2d! 'n3x2 'depth 1.5 lum-prop)
(insert2d! 'n3x4 'depth 3.5 lum-prop)
(insert2d! 'n3x5 'depth 4.5 lum-prop)
(insert2d! 'n3x6 'depth 5.5 lum-prop)
(insert2d! 'n3x8 'depth 7.25 lum-prop)
(insert2d! 'n3x10 'depth 9.25 lum-prop)
(insert2d! 'n3x12 'depth 11.25 lum-prop)

(insert2d! 'n4x2 'area 5.25 lum-prop)
(insert2d! 'n4x3 'area 8.75 lum-prop)
(insert2d! 'n4x4 'area 12.25 lum-prop)
(insert2d! 'n4x5 'area 15.75 lum-prop)
(insert2d! 'n4x6 'area 19.25 lum-prop)
(insert2d! 'n4x8 'area 25.38 lum-prop)
(insert2d! 'n4x10 'area 32.38 lum-prop)
(insert2d! 'n4x12 'area 39.38 lum-prop)
(insert2d! 'n4x2 'moi .9844 lum-prop)
(insert2d! 'n4x3 'moi 4.557 lum-prop)
(insert2d! 'n4x4 'moi 12.5 lum-prop)
(insert2d! 'n4x5 'moi 26.58 lum-prop)
(insert2d! 'n4x6 'moi 48.53 lum-prop)
(insert2d! 'n4x8 'moi 111.1 lum-prop)
(insert2d! 'n4x10 'moi 230.8 lum-prop)
(insert2d! 'n4x12 'moi 415.3 lum-prop)
(insert2d! 'n4x2 'sm 1.313 lum-prop)
(insert2d! 'n4x3 'sm 3.646 lum-prop)
(insert2d! 'n4x4 'sm 7.146 lum-prop)
(insert2d! 'n4x5 'sm 11.81 lum-prop)
(insert2d! 'n4x6 'sm 17.65 lum-prop)
(insert2d! 'n4x8 'sm 30.66 lum-prop)
(insert2d! 'n4x10 'sm 49.91 lum-prop)
(insert2d! 'n4x12 'sm 73.83 lum-prop)
(insert2d! 'n4x2 'width 3.5 lum-prop)
(insert2d! 'n4x3 'width 3.5 lum-prop)
(insert2d! 'n4x4 'width 3.5 lum-prop)
(insert2d! 'n4x5 'width 3.5 lum-prop)
(insert2d! 'n4x6 'width 3.5 lum-prop)
(insert2d! 'n4x8 'width 3.5 lum-prop)
(insert2d! 'n4x10 'width 3.5 lum-prop)
(insert2d! 'n4x12 'width 3.5 lum-prop)
(insert2d! 'n4x2 'depth 1.5 lum-prop)
(insert2d! 'n4x3 'depth 2.5 lum-prop)
(insert2d! 'n4x4 'depth 3.5 lum-prop)
(insert2d! 'n4x5 'depth 4.5 lum-prop)
(insert2d! 'n4x6 'depth 5.5 lum-prop)
(insert2d! 'n4x8 'depth 7.25 lum-prop)
(insert2d! 'n4x10 'depth 9.25 lum-prop)
(insert2d! 'n4x12 'depth 11.25 lum-prop)

(insert2d! 'n5x2 'area 6.75 lum-prop)
(insert2d! 'n5x3 'area 11.25 lum-prop)
(insert2d! 'n5x4 'area 15.75 lum-prop)
(insert2d! 'n5x5 'area 20.25 lum-prop)
(insert2d! 'n5x2 'moi 1.266 lum-prop)
(insert2d! 'n5x3 'moi 5.859 lum-prop)
(insert2d! 'n5x4 'moi 16.08 lum-prop)
(insert2d! 'n5x5 'moi 34.17 lum-prop)
```

```
(insert2d! 'n5x2 'sm 1.688 lum-prop)
(insert2d! 'n5x3 'sm 4.688 lum-prop)
(insert2d! 'n5x4 'sm 9.188 lum-prop)
(insert2d! 'n5x5 'sm 15.19 lum-prop)
(insert2d! 'n5x2 'width 4.5 lum-prop)
(insert2d! 'n5x3 'width 4.5 lum-prop)
(insert2d! 'n5x4 'width 4.5 lum-prop)
(insert2d! 'n5x5 'width 4.5 lum-prop)
(insert2d! 'n5x2 'depth 1.5 lum-prop)
(insert2d! 'n5x3 'depth 2.5 lum-prop)
(insert2d! 'n5x4 'depth 3.5 lum-prop)
(insert2d! 'n5x5 'depth 4.5 lum-prop)

(insert2d! 'n6x2 'area 8.25 lum-prop)
(insert2d! 'n6x3 'area 13.75 lum-prop)
(insert2d! 'n6x4 'area 19.25 lum-prop)
(insert2d! 'n6x6 'area 30.25 lum-prop)
(insert2d! 'n6x8 'area 41.25 lum-prop)
(insert2d! 'n6x10 'area 52.25 lum-prop)
(insert2d! 'n6x12 'area 63.25 lum-prop)
(insert2d! 'n6x2 'moi 1.547 lum-prop)
(insert2d! 'n6x3 'moi 7.161 lum-prop)
(insert2d! 'n6x4 'moi 19.65 lum-prop)
(insert2d! 'n6x6 'moi 76.26 lum-prop)
(insert2d! 'n6x8 'moi 193.4 lum-prop)
(insert2d! 'n6x10 'moi 393.0 lum-prop)
(insert2d! 'n6x12 'moi 697.1 lum-prop)
(insert2d! 'n6x2 'sm 2.063 lum-prop)
(insert2d! 'n6x3 'sm 5.729 lum-prop)
(insert2d! 'n6x4 'sm 11.23 lum-prop)
(insert2d! 'n6x6 'sm 27.73 lum-prop)
(insert2d! 'n6x8 'sm 51.56 lum-prop)
(insert2d! 'n6x10 'sm 82.73 lum-prop)
(insert2d! 'n6x12 'sm 121.2 lum-prop)
(insert2d! 'n6x2 'width 5.5 lum-prop)
(insert2d! 'n6x3 'width 5.5 lum-prop)
(insert2d! 'n6x4 'width 5.5 lum-prop)
(insert2d! 'n6x6 'width 5.5 lum-prop)
(insert2d! 'n6x8 'width 5.5 lum-prop)
(insert2d! 'n6x10 'width 5.5 lum-prop)
(insert2d! 'n6x12 'width 5.5 lum-prop)
(insert2d! 'n6x2 'depth 1.5 lum-prop)
(insert2d! 'n6x3 'depth 2.5 lum-prop)
(insert2d! 'n6x4 'depth 3.5 lum-prop)
(insert2d! 'n6x6 'depth 5.5 lum-prop)
(insert2d! 'n6x8 'depth 7.5 lum-prop)
(insert2d! 'n6x10 'depth 9.5 lum-prop)
(insert2d! 'n6x12 'depth 11.5 lum-prop)

(insert2d! 'n8x2 'area 10.88 lum-prop)
(insert2d! 'n8x3 'area 18.12 lum-prop)
(insert2d! 'n8x4 'area 25.38 lum-prop)
(insert2d! 'n8x6 'area 41.25 lum-prop)
(insert2d! 'n8x8 'area 56.25 lum-prop)
(insert2d! 'n8x10 'area 71.25 lum-prop)
(insert2d! 'n8x12 'area 86.25 lum-prop)
(insert2d! 'n8x2 'moi 2.093 lum-prop)
(insert2d! 'n8x3 'moi 9.440 lum-prop)
(insert2d! 'n8x4 'moi 25.9 lum-prop)
(insert2d! 'n8x6 'moi 104.0 lum-prop)
(insert2d! 'n8x8 'moi 263.7 lum-prop)
(insert2d! 'n8x10 'moi 535.9 lum-prop)
(insert2d! 'n8x12 'moi 950.5 lum-prop)
(insert2d! 'n8x2 'sm 2.719 lum-prop)
(insert2d! 'n8x3 'sm 7.552 lum-prop)
(insert2d! 'n8x4 'sm 14.8 lum-prop)
```

```
(insert2d! 'n8x6 'sm 37.81 lum-prop)
(insert2d! 'n8x8 'sm 70.31 lum-prop)
(insert2d! 'n8x10 'sm 112.8 lum-prop)
(insert2d! 'n8x12 'sm 165.3 lum-prop)
(insert2d! 'n8x2 'width 7.25 lum-prop)
(insert2d! 'n8x3 'width 7.25 lum-prop)
(insert2d! 'n8x4 'width 7.25 lum-prop)
(insert2d! 'n8x6 'width 7.25 lum-prop)
(insert2d! 'n8x8 'width 7.25 lum-prop)
(insert2d! 'n8x10 'width 7.25 lum-prop)
(insert2d! 'n8x12 'width 7.25 lum-prop)
(insert2d! 'n8x2 'depth 1.5 lum-prop)
(insert2d! 'n8x3 'depth 2.5 lum-prop)
(insert2d! 'n8x4 'depth 3.5 lum-prop)
(insert2d! 'n8x6 'depth 5.5 lum-prop)
(insert2d! 'n8x8 'depth 7.5 lum-prop)
(insert2d! 'n8x10 'depth 9.5 lum-prop)
(insert2d! 'n8x12 'depth 11.5 lum-prop)

(insert2d! 'n10x2 'area 13.88 lum-prop)
(insert2d! 'n10x3 'area 23.12 lum-prop)
(insert2d! 'n10x4 'area 32.38 lum-prop)
(insert2d! 'n10x6 'area 52.25 lum-prop)
(insert2d! 'n10x8 'area 71.25 lum-prop)
(insert2d! 'n10x10 'area 90.25 lum-prop)
(insert2d! 'n10x12 'area 109.2 lum-prop)
(insert2d! 'n10x2 'moi 2.602 lum-prop)
(insert2d! 'n10x3 'moi 12.04 lum-prop)
(insert2d! 'n10x4 'moi 33.05 lum-prop)
(insert2d! 'n10x6 'moi 131.7 lum-prop)
(insert2d! 'n10x8 'moi 334.0 lum-prop)
(insert2d! 'n10x10 'moi 678.8 lum-prop)
(insert2d! 'n10x12 'moi 1204 lum-prop)
(insert2d! 'n10x2 'sm 3.469 lum-prop)
(insert2d! 'n10x3 'sm 9.635 lum-prop)
(insert2d! 'n10x4 'sm 18.88 lum-prop)
(insert2d! 'n10x6 'sm 47.9 lum-prop)
(insert2d! 'n10x8 'sm 89.06 lum-prop)
(insert2d! 'n10x10 'sm 142.9 lum-prop)
(insert2d! 'n10x12 'sm 209.4 lum-prop)
(insert2d! 'n10x2 'width 9.25 lum-prop)
(insert2d! 'n10x3 'width 9.25 lum-prop)
(insert2d! 'n10x4 'width 9.25 lum-prop)
(insert2d! 'n10x6 'width 9.25 lum-prop)
(insert2d! 'n10x8 'width 9.25 lum-prop)
(insert2d! 'n10x10 'width 9.25 lum-prop)
(insert2d! 'n10x12 'width 9.25 lum-prop)
(insert2d! 'n10x2 'depth 1.5 lum-prop)
(insert2d! 'n10x3 'depth 2.5 lum-prop)
(insert2d! 'n10x4 'depth 3.5 lum-prop)
(insert2d! 'n10x6 'depth 5.5 lum-prop)
(insert2d! 'n10x8 'depth 7.5 lum-prop)
(insert2d! 'n10x10 'depth 9.5 lum-prop)
(insert2d! 'n10x12 'depth 11.5 lum-prop)

(insert2d! 'n12x2 'area 16.88 lum-prop)
(insert2d! 'n12x3 'area 28.12 lum-prop)
(insert2d! 'n12x4 'area 39.38 lum-prop)
(insert2d! 'n12x6 'area 63.25 lum-prop)
(insert2d! 'n12x8 'area 86.25 lum-prop)
(insert2d! 'n12x10 'area 109.2 lum-prop)
(insert2d! 'n12x12 'area 132.2 lum-prop)
(insert2d! 'n12x2 'moi 3.164 lum-prop)
(insert2d! 'n12x3 'moi 14.65 lum-prop)
(insert2d! 'n12x4 'moi 40.20 lum-prop)
(insert2d! 'n12x6 'moi 159.4 lum-prop)
```

```
(insert2d! 'n12x8 'moi 404.3 lum-prop)
(insert2d! 'n12x10 'moi 821.7 lum-prop)
(insert2d! 'n12x12 'moi 1458 lum-prop)
(insert2d! 'n12x2 'sm 4.129 lum-prop)
(insert2d! 'n12x3 'sm 11.72 lum-prop)
(insert2d! 'n12x4 'sm 22.97 lum-prop)
(insert2d! 'n12x6 'sm 57.98 lum-prop)
(insert2d! 'n12x8 'sm 107.8 lum-prop)
(insert2d! 'n12x10 'sm 173.0 lum-prop)
(insert2d! 'n12x12 'sm 253.5 lum-prop)
(insert2d! 'n12x2 'width 11.25 lum-prop)
(insert2d! 'n12x3 'width 11.25 lum-prop)
(insert2d! 'n12x4 'width 11.25 lum-prop)
(insert2d! 'n12x6 'width 11.25 lum-prop)
(insert2d! 'n12x8 'width 11.25 lum-prop)
(insert2d! 'n12x10 'width 11.25 lum-prop)
(insert2d! 'n12x12 'width 11.25 lum-prop)
(insert2d! 'n12x2 'depth 1.5 lum-prop)
(insert2d! 'n12x3 'depth 2.5 lum-prop)
(insert2d! 'n12x4 'depth 3.5 lum-prop)
(insert2d! 'n12x6 'depth 5.5 lum-prop)
(insert2d! 'n12x8 'depth 7.5 lum-prop)
(insert2d! 'n12x10 'depth 9.5 lum-prop)
(insert2d! 'n12x12 'depth 11.5 lum-prop)

;;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
;;                          NOMINAL LUMBER DIMENSIONS
;;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

(define nominal (make-table2d))

(insert2d! 2 3 'n2x3 nominal)
(insert2d! 2 4 'n2x4 nominal)
(insert2d! 2 5 'n2x5 nominal)
(insert2d! 2 6 'n2x6 nominal)
(insert2d! 2 8 'n2x8 nominal)
(insert2d! 2 10 'n2x10 nominal)
(insert2d! 2 12 'n2x12 nominal)

(insert2d! 3 2 'n3x2 nominal)
(insert2d! 3 4 'n3x4 nominal)
(insert2d! 3 5 'n3x5 nominal)
(insert2d! 3 6 'n3x6 nominal)
(insert2d! 3 8 'n3x8 nominal)
(insert2d! 3 10 'n3x10 nominal)
(insert2d! 3 12 'n3x12 nominal)

(insert2d! 4 2 'n4x2 nominal)
(insert2d! 4 3 'n4x3 nominal)
(insert2d! 4 4 'n4x4 nominal)
(insert2d! 4 5 'n4x5 nominal)
(insert2d! 4 6 'n4x6 nominal)
(insert2d! 4 8 'n4x8 nominal)
(insert2d! 4 10 'n4x10 nominal)
(insert2d! 4 12 'n4x12 nominal)

(insert2d! 5 2 'n5x2 nominal)
(insert2d! 5 3 'n5x3 nominal)
(insert2d! 5 4 'n5x4 nominal)
(insert2d! 5 5 'n5x5 nominal)

(insert2d! 6 2 'n6x2 nominal)
(insert2d! 6 3 'n6x3 nominal)
(insert2d! 6 4 'n6x3 nominal)
(insert2d! 6 6 'n6x6 nominal)
(insert2d! 6 8 'n6x8 nominal)
```

```
(insert2d! 6 10 'n6x10 nominal)
(insert2d! 6 12 'n6x12 nominal)

(insert2d! 8 2 'n8x2 nominal)
(insert2d! 8 3 'n8x3 nominal)
(insert2d! 8 4 'n8x4 nominal)
(insert2d! 8 6 'n8x6 nominal)
(insert2d! 8 8 'n8x8 nominal)
(insert2d! 8 10 'n8x10 nominal)
(insert2d! 8 12 'n8x12 nominal)

(insert2d! 10 2 'n10x2 nominal)
(insert2d! 10 3 'n10x3 nominal)
(insert2d! 10 4 'n10x4 nominal)
(insert2d! 10 6 'n10x6 nominal)
(insert2d! 10 8 'n10x8 nominal)
(insert2d! 10 10 'n10x10 nominal)
(insert2d! 10 12 'n10x12 nominal)

(insert2d! 12 2 'n12x2 nominal)
(insert2d! 12 3 'n12x3 nominal)
(insert2d! 12 4 'n12x4 nominal)
(insert2d! 12 6 'n12x6 nominal)
(insert2d! 12 8 'n12x8 nominal)
(insert2d! 12 10 'n12x10 nominal)
(insert2d! 12 12 'n12x12 nominal)
```

```
(module stress)

;;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;;  Design Values for Visually Graded Structural Lumber (psi)
;;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

;;The table only includes size classifications for:
;;
;;     - 2 to 4 inch wide and 2 to 4 inch thick
;;     - 2 to 4 inch wide and 4 inch thick
;;     - 2 to 4 inch wide and 5 inches or greater thick

(define lum-stress (make-table4d))

;;Aspen

(insert4d! 'a 'ss 's2to4/2to4 'f-b-sing 1300 lum-stress)
(insert4d! 'a 'ss 's2to4/2to4 'f-b-rep 1500 lum-stress)
(insert4d! 'a 'ss 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'a 'ss 's2to4/2to4 'f-c-perp 265 lum-stress)
(insert4d! 'a 'ss 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'a 'n1 's2to4/2to4 'f-b-sing 1100 lum-stress)
(insert4d! 'a 'n1 's2to4/2to4 'f-b-rep 1300 lum-stress)
(insert4d! 'a 'n1 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'a 'n1 's2to4/2to4 'f-c-perp 265 lum-stress)
(insert4d! 'a 'n1 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'a 'n2 's2to4/2to4 'f-b-sing 925 lum-stress)
(insert4d! 'a 'n2 's2to4/2to4 'f-b-rep 1050 lum-stress)
(insert4d! 'a 'n2 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'a 'n2 's2to4/2to4 'f-c-perp 265 lum-stress)
(insert4d! 'a 'n2 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'a 'n3 's2to4/2to4 'f-b-sing 500 lum-stress)
(insert4d! 'a 'n3 's2to4/2to4 'f-b-rep 575 lum-stress)
(insert4d! 'a 'n3 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'a 'n3 's2to4/2to4 'f-c-perp 265 lum-stress)
(insert4d! 'a 'n3 's2to4/2to4 'e 900000 lum-stress)
(insert4d! 'a 'app 's2to4/2to4 'f-b-sing 1100 lum-stress)
(insert4d! 'a 'app 's2to4/2to4 'f-b-rep 1300 lum-stress)
(insert4d! 'a 'app 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'a 'app 's2to4/2to4 'f-c-perp 265 lum-stress)
(insert4d! 'a 'app 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'a 'stud 's2to4/2to4 'f-b-sing 500 lum-stress)
(insert4d! 'a 'stud 's2to4/2to4 'f-b-rep 575 lum-stress)
(insert4d! 'a 'stud 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'a 'stud 's2to4/2to4 'f-c-perp 265 lum-stress)
(insert4d! 'a 'stud 's2to4/2to4 'e 900000 lum-stress)

(insert4d! 'a 'con 's2to4/4 'f-b-sing 650 lum-stress)
(insert4d! 'a 'con 's2to4/4 'f-b-rep 750 lum-stress)
(insert4d! 'a 'con 's2to4/4 'f-v 60 lum-stress)
(insert4d! 'a 'con 's2to4/4 'f-c-perp 265 lum-stress)
(insert4d! 'a 'con 's2to4/4 'e 900000 lum-stress)
(insert4d! 'a 'std 's2to4/4 'f-b-sing 375 lum-stress)
(insert4d! 'a 'std 's2to4/4 'f-b-rep 425 lum-stress)
(insert4d! 'a 'std 's2to4/4 'f-v 60 lum-stress)
(insert4d! 'a 'std 's2to4/4 'f-c-perp 265 lum-stress)
(insert4d! 'a 'std 's2to4/4 'e 900000 lum-stress)
(insert4d! 'a 'util 's2to4/4 'f-b-sing 175 lum-stress)
(insert4d! 'a 'util 's2to4/4 'f-b-rep 200 lum-stress)
(insert4d! 'a 'util 's2to4/4 'f-v 60 lum-stress)
(insert4d! 'a 'util 's2to4/4 'f-c-perp 265 lum-stress)
(insert4d! 'a 'util 's2to4/4 'e 900000 lum-stress)

(insert4d! 'a 'ss 's2to4/>=5 'f-b-sing 1150 lum-stress)
(insert4d! 'a 'ss 's2to4/>=5 'f-b-rep 1300 lum-stress)
(insert4d! 'a 'ss 's2to4/>=5 'f-v 60 lum-stress)
```

```
(insert4d! 'a 'ss 's2to4/>=5 'f-c-perp 265 lum-stress)
(insert4d! 'a 'ss 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'a 'n1 's2to4/>=5 'f-b-sing 950 lum-stress)
(insert4d! 'a 'n1 's2to4/>=5 'f-b-rep 1100 lum-stress)
(insert4d! 'a 'n1 's2to4/>=5 'f-v 60 lum-stress)
(insert4d! 'a 'n1 's2to4/>=5 'f-c-perp 265 lum-stress)
(insert4d! 'a 'n1 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'a 'n2 's2to4/>=5 'f-b-sing 775 lum-stress)
(insert4d! 'a 'n2 's2so4/>=5 'f-b-rep 900 lum-stress)
(insert4d! 'a 'n2 's2so4/>=5 'f-v 60 lum-stress)
(insert4d! 'a 'n2 's2to4/>=5 'f-c-perp 265 lum-stress)
(insert4d! 'a 'n2 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'a 'n3 's2to4/>=5 'f-b-sing 450 lum-stress)
(insert4d! 'a 'n3 's2to4/>=5 'f-b-rep 525 lum-stress)
(insert4d! 'a 'n3 's2to4/>=5 'f-v 60 lum-stress)
(insert4d! 'a 'n3 's2to4/>=5 'f-c-perp 265 lum-stress)
(insert4d! 'a 'n3 's2to4/>=5 'e 900000 lum-stress)
(insert4d! 'a 'app 's2to4/>=5 'f-b-sing 950 lum-stress)
(insert4d! 'a 'app 's2to4/>=5 'f-b-rep 1100 lum-stress)
(insert4d! 'a 'app 's2to4/>=5 'f-v 60 lum-stress)
(insert4d! 'a 'app 's2to4/>=5 'f-c-perp 265 lum-stress)
(insert4d! 'a 'app 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'a 'stud 's2to4/>=5 'f-b-sing 450 lum-stress)
(insert4d! 'a 'stud 's2to4/>=5 'f-b-rep 525 lum-stress)
(insert4d! 'a 'stud 's2to4/>=5 'f-v 60 lum-stress)
(insert4d! 'a 'stud 's2to4/>=5 'f-c-perp 265 lum-stress)
(insert4d! 'a 'stud 's2to4/>=5 'e 900000 lum-stress)

;;Balsam Fir

(insert4d! 'bf 'ss 's2to4/2to4 'f-b-sing 1750 lum-stress)
(insert4d! 'bf 'n1 's2to4/2to4 'f-b-sing 1450 lum-stress)
(insert4d! 'bf 'n2 's2to4/2to4 'f-b-sing 1200 lum-stress)
(insert4d! 'bf 'n3 's2to4/2to4 'f-b-sing 675 lum-stress)
(insert4d! 'bf 'app 's2to4/2to4 'f-b-sing 1450 lum-stress)
(insert4d! 'bf 'stud 's2to4/2to4 'f-b-sing 675 lum-stress)
(insert4d! 'bf 'ss 's2to4/2to4 'f-b-rep 2000 lum-stress)
(insert4d! 'bf 'n1 's2to4/2to4 'f-b-rep 1700 lum-stress)
(insert4d! 'bf 'n2 's2to4/2to4 'f-b-rep 1400 lum-stress)
(insert4d! 'bf 'n3 's2to4/2to4 'f-b-rep 775 lum-stress)
(insert4d! 'bf 'app 's2to4/2to4 'f-b-rep 1700 lum-stress)
(insert4d! 'bf 'stud 's2to4/2to4 'f-b-rep 775 lum-stress)
(insert4d! 'bf 'ss 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'bf 'n1 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'bf 'n2 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'bf 'n3 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'bf 'app 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'bf 'stud 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'bf 'ss 's2to4/2to4 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'n1 's2to4/2to4 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'n2 's2to4/2to4 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'n3 's2to4/2to4 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'app 's2to4/2to4 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'stud 's2to4/2to4 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'ss 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'bf 'n1 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'bf 'n2 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'bf 'n3 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'bf 'app 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'bf 'stud 's2to4/2to4 'e 1200000 lum-stress)

(insert4d! 'bf 'con 's2to4/4 'f-b-sing 875 lum-stress)
(insert4d! 'bf 'con 's2to4/4 'f-b-rep 1000 lum-stress)
(insert4d! 'bf 'con 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'bf 'con 's2to4/4 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'con 's2to4/4 'e 1200000 lum-stress)
```

```
(insert4d! 'bf 'std 's2to4/4 'f-b-sing 500 lum-stress)
(insert4d! 'bf 'std 's2to4/4 'f-b-rep 575 lum-stress)
(insert4d! 'bf 'std 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'bf 'std 's2to4/4 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'std 's2to4/4 'e 1200000 lum-stress)
(insert4d! 'bf 'util 's2to4/4 'f-b-sing 225 lum-stress)
(insert4d! 'bf 'util 's2to4/4 'f-b-rep 275 lum-stress)
(insert4d! 'bf 'util 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'bf 'util 's2to4/4 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'util 's2to4/4 'e 1200000 lum-stress)

(insert4d! 'bf 'ss 's2to4/>=5 'f-b-sing 1500 lum-stress)
(insert4d! 'bf 'n1 's2to4/>=5 'f-b-sing 1250 lum-stress)
(insert4d! 'bf 'n2 's2to4/>=5 'f-b-sing 1050 lum-stress)
(insert4d! 'bf 'n3 's2to4/>=5 'f-b-sing 600 lum-stress)
(insert4d! 'bf 'app 's2to4/>=5 'f-b-sing 1250 lum-stress)
(insert4d! 'bf 'stud 's2to4/>=5 'f-b-sing 600 lum-stress)
(insert4d! 'bf 'ss 's2to4/>=5 'f-b-rep 1700 lum-stress)
(insert4d! 'bf 'n1 's2to4/>=5 'f-b-rep 1450 lum-stress)
(insert4d! 'bf 'n2 's2to4/>=5 'f-b-rep 1200 lum-stress)
(insert4d! 'bf 'n3 's2to4/>=5 'f-b-rep 700 lum-stress)
(insert4d! 'bf 'app 's2to4/>=5 'f-b-rep 1450 lum-stress)
(insert4d! 'bf 'stud 's2to4/>=5 'f-b-rep 700 lum-stress)
(insert4d! 'bf 'ss 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'bf 'n1 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'bf 'n2 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'bf 'n3 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'bf 'app 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'bf 'stud 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'bf 'ss 's2to4/>=5 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'n1 's2to4/>=5 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'n2 's2to4/>=5 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'n3 's2to4/>=5 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'app 's2to4/>=5 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'stud 's2to4/>=5 'f-c-perp 305 lum-stress)
(insert4d! 'bf 'ss 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'bf 'n1 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'bf 'n2 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'bf 'n3 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'bf 'app 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'bf 'stud 's2to4/>=5 'e 1200000 lum-stress)

;;Douglas Fir-Larch

(insert4d! 'dfl 'dss 's2to4/2to4 'f-b-sing 2450 lum-stress)
(insert4d! 'dfl 'ss 's2to4/2to4 'f-b-sing 2100 lum-stress)
(insert4d! 'dfl 'dn1 's2to4/2to4 'f-b-sing 2050 lum-stress)
(insert4d! 'dfl 'n1 's2to4/2to4 'f-b-sing 1750 lum-stress)
(insert4d! 'dfl 'dn2 's2to4/2to4 'f-b-sing 1700 lum-stress)
(insert4d! 'dfl 'n2 's2to4/2to4 'f-b-sing 1450 lum-stress)
(insert4d! 'dfl 'n3 's2to4/2to4 'f-b-sing 800 lum-stress)
(insert4d! 'dfl 'app 's2to4/2to4 'f-b-sing 1750 lum-stress)
(insert4d! 'dfl 'stud 's2to4/2to4 'f-b-sing 800 lum-stress)
(insert4d! 'dfl 'dss 's2to4/2to4 'f-b-rep 2800 lum-stress)
(insert4d! 'dfl 'ss 's2to4/2to4 'f-b-rep 2400 lum-stress)
(insert4d! 'dfl 'dn1 's2to4/2to4 'f-b-rep 2400 lum-stress)
(insert4d! 'dfl 'n1 's2to4/2to4 'f-b-rep 2050 lum-stress)
(insert4d! 'dfl 'dn2 's2to4/2to4 'f-b-rep 1950 lum-stress)
(insert4d! 'dfl 'n2 's2to4/2to4 'f-b-rep 1650 lum-stress)
(insert4d! 'dfl 'n3 's2to4/2to4 'f-b-rep 925 lum-stress)
(insert4d! 'dfl 'app 's2to4/2to4 'f-b-rep 2050 lum-stress)
(insert4d! 'dfl 'stud 's2to4/2to4 'f-b-rep 925 lum-stress)
(insert4d! 'dfl 'dss 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'dfl 'ss 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'dfl 'dn1 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'dfl 'n1 's2to4/2to4 'f-v 95 lum-stress)
```

```
(insert4d! 'dfl 'dn2 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'dfl 'n2 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'dfl 'n3 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'dfl 'app 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'dfl 'stud 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'dfl 'dss 's2to4/2to4 'f-c-perp 730 lum-stress)
(insert4d! 'dfl 'ss 's2to4/2to4 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'dn1 's2to4/2to4 'f-c-perp 730 lum-stress)
(insert4d! 'dfl 'n1 's2to4/2to4 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'dn2 's2to4/2to4 'f-c-perp 730 lum-stress)
(insert4d! 'dfl 'n2 's2to4/2to4 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'n3 's2to4/2to4 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'app 's2to4/2to4 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'stud 's2to4/2to4 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'dss 's2to4/2to4 'e 1900000 lum-stress)
(insert4d! 'dfl 'ss 's2to4/2to4 'e 1800000 lum-stress)
(insert4d! 'dfl 'dn1 's2to4/2to4 'e 1900000 lum-stress)
(insert4d! 'dfl 'n1 's2to4/2to4 'e 1800000 lum-stress)
(insert4d! 'dfl 'dn2 's2to4/2to4 'e 1700000 lum-stress)
(insert4d! 'dfl 'n2 's2to4/2to4 'e 1700000 lum-stress)
(insert4d! 'dfl 'n3 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'dfl 'app 's2to4/2to4 'e 1800000 lum-stress)
(insert4d! 'dfl 'stud 's2to4/2to4 'e 1500000 lum-stress)

(insert4d! 'dfl 'con 's2to4/4 'f-b-sing 1050 lum-stress)
(insert4d! 'dfl 'con 's2to4/4 'f-b-rep 1200 lum-stress)
(insert4d! 'dfl 'con 's2to4/4 'f-v 95 lum-stress)
(insert4d! 'dfl 'con 's2to4/4 'f-c-perp 385 lum-stress)
(insert4d! 'dfl 'con 's2to4/4 'e 1500000 lum-stress)
(insert4d! 'dfl 'std 's2to4/4 'f-b-sing 600 lum-stress)
(insert4d! 'dfl 'std 's2to4/4 'f-b-rep 675 lum-stress)
(insert4d! 'dfl 'std 's2to4/4 'f-v 95 lum-stress)
(insert4d! 'dfl 'std 's2to4/4 'f-c-perp 385 lum-stress)
(insert4d! 'dfl 'std 's2to4/4 'e 1500000 lum-stress)
(insert4d! 'dfl 'util 's2to4/4 'f-b-sing 275 lum-stress)
(insert4d! 'dfl 'util 's2to4/4 'f-b-rep 325 lum-stress)
(insert4d! 'dfl 'util 's2to4/4 'f-v 95 lum-stress)
(insert4d! 'dfl 'util 's2to4/4 'f-c-perp 385 lum-stress)
(insert4d! 'dfl 'util 's2to4/4 'e 1500000 lum-stress)

(insert4d! 'dfl 'dss 's2to4/>=5 'f-b-sing 2100 lum-stress)
(insert4d! 'dfl 'ss 's2to4/>=5 'f-b-sing 1800 lum-stress)
(insert4d! 'dfl 'dn1 's2to4/>=5 'f-b-sing 1800 lum-stress)
(insert4d! 'dfl 'n1 's2to4/>=5 'f-b-sing 1500 lum-stress)
(insert4d! 'dfl 'dn2 's2to4/>=5 'f-b-sing 1450 lum-stress)
(insert4d! 'dfl 'n2 's2to4/>=5 'f-b-sing 1250 lum-stress)
(insert4d! 'dfl 'n3 's2to4/>=5 'f-b-sing 725 lum-stress)
(insert4d! 'dfl 'app 's2to4/>=5 'f-b-sing 1500 lum-stress)
(insert4d! 'dfl 'stud 's2to4/>=5 'f-b-sing 725 lum-stress)
(insert4d! 'dfl 'dss 's2to4/>=5 'f-b-rep 2400 lum-stress)
(insert4d! 'dfl 'ss 's2to4/>=5 'f-b-rep 2050 lum-stress)
(insert4d! 'dfl 'dn1 's2to4/>=5 'f-b-rep 2050 lum-stress)
(insert4d! 'dfl 'n1 's2to4/>=5 'f-b-rep 1750 lum-stress)
(insert4d! 'dfl 'dn2 's2to4/>=5 'f-b-rep 1700 lum-stress)
(insert4d! 'dfl 'n2 's2to4/>=5 'f-b-rep 1450 lum-stress)
(insert4d! 'dfl 'n3 's2to4/>=5 'f-b-rep 850 lum-stress)
(insert4d! 'dfl 'app 's2to4/>=5 'f-b-rep 1750 lum-stress)
(insert4d! 'dfl 'stud 's2to4/>=5 'f-b-rep 850 lum-stress)
(insert4d! 'dfl 'dss 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'dfl 'ss 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'dfl 'dn1 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'dfl 'n1 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'dfl 'dn2 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'dfl 'n2 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'dfl 'n3 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'dfl 'app 's2to4/>=5 'f-v 95 lum-stress)
```

```
(insert4d! 'dfl 'stud 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'dfl 'dss 's2to4/>=5 'f-c-perp 730 lum-stress)
(insert4d! 'dfl 'ss 's2to4/>=5 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'dn1 's2to4/>=5 'f-c-perp 730 lum-stress)
(insert4d! 'dfl 'n1 's2to4/>=5 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'dn2 's2to4/>=5 'f-c-perp 730 lum-stress)
(insert4d! 'dfl 'n2 's2to4/>=5 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'n3 's2to4/>=5 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'app 's2to4/>=5 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'stud 's2to4/>=5 'f-c-perp 625 lum-stress)
(insert4d! 'dfl 'dss 's2to4/>=5 'e 1900000 lum-stress)
(insert4d! 'dfl 'ss 's2to4/>=5 'e 1800000 lum-stress)
(insert4d! 'dfl 'dn1 's2to4/>=5 'e 1900000 lum-stress)
(insert4d! 'dfl 'n1 's2to4/>=5 'e 1800000 lum-stress)
(insert4d! 'dfl 'dn2 's2to4/>=5 'e 1700000 lum-stress)
(insert4d! 'dfl 'n2 's2to4/>=5 'e 1700000 lum-stress)
(insert4d! 'dfl 'n3 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'dfl 'app 's2to4/>=5 'e 1800000 lum-stress)
(insert4d! 'dfl 'stud 's2to4/>=5 'e 1500000 lum-stress)

;;Douglas Fir South

(insert4d! 'dfs 'ss 's2to4/2to4 'f-b-sing 2000 lum-stress)
(insert4d! 'dfs 'n1 's2to4/2to4 'f-b-sing 1700 lum-stress)
(insert4d! 'dfs 'n2 's2to4/2to4 'f-b-sing 1400 lum-stress)
(insert4d! 'dfs 'n3 's2to4/2to4 'f-b-sing 775 lum-stress)
(insert4d! 'dfs 'app 's2to4/2to4 'f-b-sing 1700 lum-stress)
(insert4d! 'dfs 'stud 's2to4/2to4 'f-b-sing 775 lum-stress)
(insert4d! 'dfs 'ss 's2to4/2to4 'f-b-rep 2300 lum-stress)
(insert4d! 'dfs 'n1 's2to4/2to4 'f-b-rep 1950 lum-stress)
(insert4d! 'dfs 'n2 's2to4/2to4 'f-b-rep 1600 lum-stress)
(insert4d! 'dfs 'n3 's2to4/2to4 'f-b-rep 875 lum-stress)
(insert4d! 'dfs 'app 's2to4/2to4 'f-b-rep 1950 lum-stress)
(insert4d! 'dfs 'stud 's2to4/2to4 'f-b-rep 875 lum-stress)
(insert4d! 'dfs 'ss 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'dfs 'n1 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'dfs 'n2 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'dfs 'n3 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'dfs 'app 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'dfs 'stud 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'dfs 'ss 's2to4/2to4 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'n1 's2to4/2to4 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'n2 's2to4/2to4 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'n3 's2to4/2to4 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'app 's2to4/2to4 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'stud 's2to4/2to4 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'ss 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'dfs 'n1 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'dfs 'n2 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'dfs 'n3 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'dfs 'app 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'dfs 'stud 's2to4/2to4 'e 1100000 lum-stress)

(insert4d! 'dfs 'con 's2to4/4 'f-b-sing 1000 lum-stress)
(insert4d! 'dfs 'con 's2to4/4 'f-b-rep 1150 lum-stress)
(insert4d! 'dfs 'con 's2to4/4 'f-v 90 lum-stress)
(insert4d! 'dfs 'con 's2to4/4 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'con 's2to4/4 'e 1100000 lum-stress)
(insert4d! 'dfs 'std 's2to4/4 'f-b-sing 550 lum-stress)
(insert4d! 'dfs 'std 's2to4/4 'f-b-rep 650 lum-stress)
(insert4d! 'dfs 'std 's2to4/4 'f-v 90 lum-stress)
(insert4d! 'dfs 'std 's2to4/4 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'std 's2to4/4 'e 1100000 lum-stress)
(insert4d! 'dfs 'util 's2to4/4 'f-b-sing 275 lum-stress)
(insert4d! 'dfs 'util 's2to4/4 'f-b-rep 300 lum-stress)
(insert4d! 'dfs 'util 's2to4/4 'f-v 90 lum-stress)
```

```
(insert4d! 'dfs 'util 's2to4/4 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'util 's2to4/4 'e 1100000 lum-stress)

(insert4d! 'dfs 'ss 's2to4/>=5 'f-b-sing 1700 lum-stress)
(insert4d! 'dfs 'n1 's2to4/>=5 'f-b-sing 1450 lum-stress)
(insert4d! 'dfs 'n2 's2to4/>=5 'f-b-sing 1200 lum-stress)
(insert4d! 'dfs 'n3 's2to4/>=5 'f-b-sing 700 lum-stress)
(insert4d! 'dfs 'app 's2to4/>=5 'f-b-sing 1450 lum-stress)
(insert4d! 'dfs 'stud 's2to4/>=5 'f-b-sing 700 lum-stress)
(insert4d! 'dfs 'ss 's2to4/>=5 'f-b-rep 1950 lum-stress)
(insert4d! 'dfs 'n1 's2to4/>=5 'f-b-rep 1650 lum-stress)
(insert4d! 'dfs 'n2 's2to4/>=5 'f-b-rep 1350 lum-stress)
(insert4d! 'dfs 'n3 's2to4/>=5 'f-b-rep 800 lum-stress)
(insert4d! 'dfs 'app 's2to4/>=5 'f-b-rep 1650 lum-stress)
(insert4d! 'dfs 'stud 's2to4/>=5 'f-b-rep 800 lum-stress)
(insert4d! 'dfs 'ss 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'dfs 'n1 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'dfs 'n2 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'dfs 'n3 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'dfs 'app 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'dfs 'stud 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'dfs 'ss 's2to4/>=5 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'n1 's2to4/>=5 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'n2 's2to4/>=5 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'n3 's2to4/>=5 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'app 's2to4/>=5 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'stud 's2to4/>=5 'f-c-perp 520 lum-stress)
(insert4d! 'dfs 'ss 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'dfs 'n1 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'dfs 'n2 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'dfs 'n3 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'dfs 'app 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'dfs 'stud 's2to4/>=5 'e 1100000 lum-stress)

;;Eastern Hemlock

(insert4d! 'eh 'ss 's2to4/2to4 'f-b-sing 1750 lum-stress)
(insert4d! 'eh 'n1 's2to4/2to4 'f-b-sing 1500 lum-stress)
(insert4d! 'eh 'n2 's2to4/2to4 'f-b-sing 1250 lum-stress)
(insert4d! 'eh 'n3 's2to4/2to4 'f-b-sing 675 lum-stress)
(insert4d! 'eh 'app 's2to4/2to4 'f-b-sing 1500 lum-stress)
(insert4d! 'eh 'stud 's2to4/2to4 'f-b-sing 675 lum-stress)
(insert4d! 'eh 'ss 's2to4/2to4 'f-b-rep 2050 lum-stress)
(insert4d! 'eh 'n1 's2to4/2to4 'f-b-rep 1750 lum-stress)
(insert4d! 'eh 'n2 's2to4/2to4 'f-b-rep 1450 lum-stress)
(insert4d! 'eh 'n3 's2to4/2to4 'f-b-rep 800 lum-stress)
(insert4d! 'eh 'app 's2to4/2to4 'f-b-rep 1750 lum-stress)
(insert4d! 'eh 'stud 's2to4/2to4 'f-b-rep 800 lum-stress)
(insert4d! 'eh 'ss 's2to4/2to4 'f-v 85 lum-stress)
(insert4d! 'eh 'n1 's2to4/2to4 'f-v 85 lum-stress)
(insert4d! 'eh 'n2 's2to4/2to4 'f-v 85 lum-stress)
(insert4d! 'eh 'n3 's2to4/2to4 'f-v 85 lum-stress)
(insert4d! 'eh 'app 's2to4/2to4 'f-v 85 lum-stress)
(insert4d! 'eh 'stud 's2to4/2to4 'f-v 85 lum-stress)
(insert4d! 'eh 'ss 's2to4/2to4 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'n1 's2to4/2to4 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'n2 's2to4/2to4 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'n3 's2to4/2to4 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'app 's2to4/2to4 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'stud 's2to4/2to4 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'ss 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'eh 'n1 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'eh 'n2 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'eh 'n3 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'eh 'app 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'eh 'stud 's2to4/2to4 'e 1000000 lum-stress)
```

```
(insert4d! 'eh 'con 's2to4/4 'f-b-sing 900 lum-stress)
(insert4d! 'eh 'con 's2to4/4 'f-b-rep 1050 lum-stress)
(insert4d! 'eh 'con 's2to4/4 'f-v 85 lum-stress)
(insert4d! 'eh 'con 's2to4/4 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'con 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'eh 'std 's2to4/4 'f-b-sing 500 lum-stress)
(insert4d! 'eh 'std 's2to4/4 'f-b-rep 575 lum-stress)
(insert4d! 'eh 'std 's2to4/4 'f-v 85 lum-stress)
(insert4d! 'eh 'std 's2to4/4 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'std 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'eh 'util 's2to4/4 'f-b-sing 250 lum-stress)
(insert4d! 'eh 'util 's2to4/4 'f-b-rep 275 lum-stress)
(insert4d! 'eh 'util 's2to4/4 'f-v 85 lum-stress)
(insert4d! 'eh 'util 's2to4/4 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'util 's2to4/4 'e 1000000 lum-stress)

(insert4d! 'eh 'ss 's2to4/>=5 'f-b-sing 1550 lum-stress)
(insert4d! 'eh 'n1 's2to4/>=5 'f-b-sing 1300 lum-stress)
(insert4d! 'eh 'n2 's2to4/>=5 'f-b-sing 1050 lum-stress)
(insert4d! 'eh 'n3 's2to4/>=5 'f-b-sing 625 lum-stress)
(insert4d! 'eh 'app 's2to4/>=5 'f-b-sing 1300 lum-stress)
(insert4d! 'eh 'stud 's2to4/>=5 'f-b-sing 625 lum-stress)
(insert4d! 'eh 'ss 's2to4/>=5 'f-b-rep 1750 lum-stress)
(insert4d! 'eh 'n1 's2to4/>=5 'f-b-rep 1500 lum-stress)
(insert4d! 'eh 'n2 's2to4/>=5 'f-b-rep 1250 lum-stress)
(insert4d! 'eh 'n3 's2to4/>=5 'f-b-rep 700 lum-stress)
(insert4d! 'eh 'app 's2to4/>=5 'f-b-rep 1500 lum-stress)
(insert4d! 'eh 'stud 's2to4/>=5 'f-b-rep 700 lum-stress)
(insert4d! 'eh 'ss 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eh 'n1 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eh 'n2 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eh 'n3 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eh 'app 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eh 'stud 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eh 'ss 's2to4/>=5 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'n1 's2to4/>=5 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'n2 's2to4/>=5 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'n3 's2to4/>=5 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'app 's2to4/>=5 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'stud 's2to4/>=5 'f-c-perp 550 lum-stress)
(insert4d! 'eh 'ss 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'eh 'n1 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'eh 'n2  s2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'eh 'n3 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'eh 'app 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'eh 'stud 's2to4/>=5 'e 1000000 lum-stress)

;;Eastern Hemlock-Tamarack

(insert4d! 'eht 'ss 's2to4/2to4 'f-b-sing 1800 lum-stress)
(insert4d! 'eht 'n1 's2to4/2to4 'f-b-sing 1500 lum-stress)
(insert4d! 'eht 'n2 's2to4/2to4 'f-b-sing 1250 lum-stress)
(insert4d! 'eht 'n3 's2to4/2to4 'f-b-sing 700 lum-stress)
(insert4d! 'eht 'app 's2to4/2to4 'f-b-sing 1300 lum-stress)
(insert4d! 'eht 'stud 's2to4/2to4 'f-b-sing 700 lum-stress)
(insert4d! 'eht 'ss 's2to4/2to4 'f-b-rep 2050 lum-stress)
(insert4d! 'eht 'n1 's2to4/2to4 'f-b-rep 1750 lum-stress)
(insert4d! 'eht 'n2 's2to4/2to4 'f-b-rep 1450 lum-stress)
(insert4d! 'eht 'n3 's2to4/2to4 'f-b-rep 800 lum-stress)
(insert4d! 'eht 'app 's2to4/2to4 'f-b-rep 1500 lum-stress)
(insert4d! 'eht 'stud 's2to4/2to4 'f-b-rep 800 lum-stress)
(insert4d! 'eht 'ss 's2to4/2to4 'f-v 85 lum-stress)
(insert4d! 'eht 'n1 's2to4/2to4 'f-v 85 lum-stress)
(insert4d! 'eht 'n2 's2to4/2to4 'f-v 85 lum-stress)
(insert4d! 'eht 'n3 's2to4/2to4 'f-v 85 lum-stress)
```

```
(insert4d! 'eht 'app 's2to4/2to4 'f-v 85 lum-stress)
(insert4d! 'eht 'stud 's2to4/2to4 'f-v 85 lum-stress)
(insert4d! 'eht 'ss 's2to4/2to4 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'n1 's2to4/2to4 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'n2 's2to4/2to4 'f-c-perp 555 lum-stress)
(insert4d! 'eht'n3 's2to4/2to4 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'app 's2to4/2to4 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'stud 's2to4/2to4 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'ss 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'eht 'n1 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'eht 'n2 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'eht 'n3 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'eht 'app 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'eht 'stud 's2to4/2to4 'e 1000000 lum-stress)

(insert4d! 'eht 'con 's2to4/4 'f-b-sing 900 lum-stress)
(insert4d! 'eht 'con 's2to4/4 'f-b-rep 1050 lum-stress)
(insert4d! 'eht 'con 's2to4/4 'f-v 85 lum-stress)
(insert4d! 'eht 'con 's2to4/4 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'con 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'eht 'std 's2to4/4 'f-b-sing 500 lum-stress)
(insert4d! 'eht 'std 's2to4/4 'f-b-rep 575 lum-stress)
(insert4d! 'eht 'std 's2to4/4 'f-v 85 lum-stress)
(insert4d! 'eht 'std 's2to4/4 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'std 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'eht 'util 's2to4/4 'f-b-sing 250 lum-stress)
(insert4d! 'eht 'util 's2to4/4 'f-b-rep 275 lum-stress)
(insert4d! 'eht 'util 's2to4/4 'f-v 85 lum-stress)
(insert4d! 'eht 'util 's2to4/4 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'util 's2to4/4 'e 1000000 lum-stress)

(insert4d! 'eht 'ss 's2to4/>=5 'f-b-sing 1550 lum-stress)
(insert4d! 'eht 'n1 's2to4/>=5 'f-b-sing 1300 lum-stress)
(insert4d! 'eht 'n2 's2to4/>=5 'f-b-sing 1050 lum-stress)
(insert4d! 'eht 'n3 's2to4/>=5 'f-b-sing 625 lum-stress)
(insert4d! 'eht 'app 's2to4/>=5 'f-b-sing 1300 lum-stress)
(insert4d! 'eht 'stud 's2to4/>=5 'f-b-sing 625 lum-stress)
(insert4d! 'eht 'ss 's2to4/>=5 'f-b-rep 1750 lum-stress)
(insert4d! 'eht 'n1 's2to4/>=5 'f-b-rep 1500 lum-stress)
(insert4d! 'eht 'n2 's2to4/>=5 'f-b-rep 1200 lum-stress)
(insert4d! 'eht 'n3 's2to4/>=5 'f-b-rep 725 lum-stress)
(insert4d! 'eht 'app 's2to4/>=5 'f-b-rep 1500 lum-stress)
(insert4d! 'eht 'stud 's2to4/>=5 'f-b-rep 725 lum-stress)
(insert4d! 'eht 'ss 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eht 'n1 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eht 'n2 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eht 'n3 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eht 'app 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eht 'stud 's2to4/>=5 'f-v 85 lum-stress)
(insert4d! 'eht 'ss 's2to4/>=5 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'n1 's2to4/>=5 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'n2 's2to4/>=5 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'n3 's2to4/>=5 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'app 's2to4/>=5 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'stud 's2to4/>=5 'f-c-perp 555 lum-stress)
(insert4d! 'eht 'ss 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'eht 'n1 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'eht 'n2 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'eht'n3 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'eht 'app 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'eht 'stud 's2to4/>=5 'e 1000000 lum-stress)

;;Eastern Spruce

(insert4d! 'es 'ss 's2to4/2to4 'f-b-sing 1400 lum-stress)
(insert4d! 'es 'n1 's2to4/2to4 'f-b-sing 1200 lum-stress)
```

```
(insert4d! 'es 'n2 's2to4/2to4 'f-b-sing 975 lum-stress)
(insert4d! 'es 'n3 's2to4/2to4 'f-b-sing 550 lum-stress)
(insert4d! 'es 'app 's2to4/2to4 'f-b-sing 1200 lum-stress)
(insert4d! 'es 'stud 's2to4/2to4 'f-b-sing 550 lum-stress)
(insert4d! 'es 'ss 's2to4/2to4 'f-b-rep 1600 lum-stress)
(insert4d! 'es 'n1 's2to4/2to4 'f-b-rep 1350 lum-stress)
(insert4d! 'es 'n2 's2to4/2to4 'f-b-rep 1100 lum-stress)
(insert4d! 'es 'n3 's2to4/2to4 'f-b-rep 625 lum-stress)
(insert4d! 'es 'app 's2to4/2to4 'f-b-rep 1350 lum-stress)
(insert4d! 'es 'stud 's2to4/2to4 'f-b-rep 625 lum-stress)
(insert4d! 'es 'ss 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'es 'n1 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'es 'n2 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'es 'n3 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'es 'app 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'es 'stud 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'es 'ss 's2to4/2to4 'f-c-perp 390 lum-stress)
(insert4d! 'es 'n1 's2to4/2to4 'f-c-perp 390 lum-stress)
(insert4d! 'es 'n2 's2to4/2to4 'f-c-perp 390 lum-stress)
(insert4d! 'es 'n3 's2to4/2to4 'f-c-perp 390 lum-stress)
(insert4d! 'es 'app 's2to4/2to4 'f-c-perp 390 lum-stress)
(insert4d! 'es 'stud 's2to4/2to4 'f-c-perp 390 lum-stress)
(insert4d! 'es 'ss 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'es 'n1 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'es 'n2 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'es 'n3 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'es 'app 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'es 'stud 's2to4/2to4 'e 1200000 lum-stress)

(insert4d! 'es 'con 's2to4/4 'f-b-sing 700 lum-stress)
(insert4d! 'es 'con 's2to4/4 'f-b-rep 800 lum-stress)
(insert4d! 'es 'con 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'es 'con 's2to4/4 'f-c-perp 390 lum-stress)
(insert4d! 'es 'con 's2to4/4 'e 1200000 lum-stress)
(insert4d! 'es 'std 's2to4/4 'f-b-sing 400 lum-stress)
(insert4d! 'es 'std 's2to4/4 'f-b-rep 450 lum-stress)
(insert4d! 'es 'std 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'es 'std 's2to4/4 'f-c-perp 390 lum-stress)
(insert4d! 'es 'std 's2to4/4 'e 1200000 lum-stress)
(insert4d! 'es 'util 's2to4/4 'f-b-sing 175 lum-stress)
(insert4d! 'es 'util 's2to4/4 'f-b-rep 225 lum-stress)
(insert4d! 'es 'util 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'es 'util 's2to4/4 'f-c-perp 390 lum-stress)
(insert4d! 'es 'util 's2to4/4 'e 1200000 lum-stress)

(insert4d! 'es 'ss 's2to4/>=5 'f-b-sing 1200 lum-stress)
(insert4d! 'es 'n1 's2to4/>=5 'f-b-sing 1000 lum-stress)
(insert4d! 'es 'n2 's2to4/>=5 'f-b-sing 825 lum-stress)
(insert4d! 'es 'n3 's2to4/>=5 'f-b-sing 475 lum-stress)
(insert4d! 'es 'app 's2to4/>=5 'f-b-sing 1000 lum-stress)
(insert4d! 'es 'stud 's2to4/>=5 'f-b-sing 475 lum-stress)
(insert4d! 'es 'ss 's2to4/>=5 'f-b-rep 1350 lum-stress)
(insert4d! 'es 'n1 's2to4/>=5 'f-b-rep 1150 lum-stress)
(insert4d! 'es 'n2 's2to4/>=5 'f-b-rep 950 lum-stress)
(insert4d! 'es 'n3 's2to4/>=5 'f-b-rep 550 lum-stress)
(insert4d! 'es 'app 's2to4/>=5 'f-b-rep 1150 lum-stress)
(insert4d! 'es 'stud 's2to4/>=5 'f-b-rep 550 lum-stress)
(insert4d! 'es 'ss 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'es 'n1 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'es 'n2 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'es 'n3 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'es 'app 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'es 'stud 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'es 'ss 's2to4/>=5 'f-c-perp 390 lum-stress)
(insert4d! 'es 'n1 's2to4/>=5 'f-c-perp 390 lum-stress)
(insert4d! 'es 'n2 's2to4/>=5 'f-c-perp 390 lum-stress)
```

```
(insert4d! 'es 'n3 's2to4/>=5 'f-c-perp 390 lum-stress)
(insert4d! 'es 'app 's2to4/>=5 'f-c-perp 390 lum-stress)
(insert4d! 'es 'stud 's2to4/>=5 'f-c-perp 390 lum-stress)
(insert4d! 'es 'ss 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'es 'n1 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'es 'n2 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'es 'n3 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'es 'app 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'es 'stud 's2to4/>=5 'e 1200000 lum-stress)

;;Eastern White Pine

(insert4d! 'ewp 'ss 's2to4/2to4 'f-b-sing 1350 lum-stress)
(insert4d! 'ewp 'n1 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'ewp 'n2 's2to4/2to4 'f-b-sing 950 lum-stress)
(insert4d! 'ewp 'n3 's2to4/2to4 'f-b-sing 525 lum-stress)
(insert4d! 'ewp 'app 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'ewp 'stud 's2to4/2to4 'f-b-sing 525 lum-stress)
(insert4d! 'ewp 'ss 's2to4/2to4 'f-b-rep 1550 lum-stress)
(insert4d! 'ewp 'n1 's2to4/2to4 'f-b-rep 1350 lum-stress)
(insert4d! 'ewp 'n2 's2to4/2to4 'f-b-rep 1100 lum-stress)
(insert4d! 'ewp 'n3 's2to4/2to4 'f-b-rep 600 lum-stress)
(insert4d! 'ewp 'app 's2to4/2to4 'f-b-rep 1350 lum-stress)
(insert4d! 'ewp 'stud 's2to4/2to4 'f-b-rep 600 lum-stress)
(insert4d! 'ewp 'ss 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ewp 'n1 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ewp 'n2 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ewp 'n3 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ewp 'app 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ewp 'stud 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ewp 'ss 's2to4/2to4 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'n1 's2to4/2to4 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'n2 's2to4/2to4 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'n3 's2to4/2to4 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'app 's2to4/2to4 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'stud 's2to4/2to4 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'ss 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'ewp 'n1 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'ewp 'n2 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'ewp 'n3 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'ewp 'app 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'ewp 'stud 's2to4/2to4 'e 1000000 lum-stress)

(insert4d! 'ewp 'con 's2to4/4 'f-b-sing 700 lum-stress)
(insert4d! 'ewp 'con 's2to4/4 'f-b-rep 800 lum-stress)
(insert4d! 'ewp 'con 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'ewp 'con 's2to4/4 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'con 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'ewp 'std 's2to4/4 'f-b-sing 375 lum-stress)
(insert4d! 'ewp 'std 's2to4/4 'f-b-rep 450 lum-stress)
(insert4d! 'ewp 'std 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'ewp 'std 's2to4/4 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'std 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'ewp 'util 's2to4/4 'f-b-sing 175 lum-stress)
(insert4d! 'ewp 'util 's2to4/4 'f-b-rep 200 lum-stress)
(insert4d! 'ewp 'util 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'ewp 'util 's2to4/4 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'util 's2to4/4 'e 1000000 lum-stress)

(insert4d! 'ewp 'ss 's2to4/>=5 'f-b-sing 1150 lum-stress)
(insert4d! 'ewp 'n1 's2to4/>=5 'f-b-sing 1000 lum-stress)
(insert4d! 'ewp 'n2 's2to4/>=5 'f-b-sing 825 lum-stress)
(insert4d! 'ewp 'n3 's2to4/>=5 'f-b-sing 475 lum-stress)
(insert4d! 'ewp 'app 's2to4/>=5 'f-b-sing 1000 lum-stress)
(insert4d! 'ewp 'stud 's2to4/>=5 'f-b-sing 475 lum-stress)
(insert4d! 'ewp 'ss 's2to4/>=5 'f-b-rep 1350 lum-stress)
```

```
(insert4d! 'ewp 'n1 's2to4/>=5 'f-b-rep 1150 lum-stress)
(insert4d! 'ewp 'n2 's2to4/>=5 'f-b-rep 950 lum-stress)
(insert4d! 'ewp 'n3 's2to4/>=5 'f-b-rep 550 lum-stress)
(insert4d! 'ewp 'app 's2to4/>=5 'f-b-rep 1150 lum-stress)
(insert4d! 'ewp 'stud 's2to4/>=5 'f-b-rep 550 lum-stress)
(insert4d! 'ewp 'ss 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ewp 'n1 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ewp 'n2 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ewp 'n3 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ewp 'app 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ewp 'stud 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ewp 'ss 's2to4/>=5 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'n1 's2to4/>=5 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'n2 's2to4/>=5 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'n3 's2to4/>=5 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'app 's2to4/>=5 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'stud 's2to4/>=5 'f-c-perp 350 lum-stress)
(insert4d! 'ewp 'ss 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'ewp 'n1 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'ewp 'n2 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'ewp 'n3 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'ewp 'app 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'ewp 'stud 's2to4/>=5 'e 1000000 lum-stress)

;;Eastern Woods

(insert4d! 'ew 'ss 's2to4/2to4 'f-b-sing 1300 lum-stress)
(insert4d! 'ew 'n1 's2to4/2to4 'f-b-sing 1100 lum-stress)
(insert4d! 'ew 'n2 's2to4/2to4 'f-b-sing 925 lum-stress)
(insert4d! 'ew 'n3 's2to4/2to4 'f-b-sing 500 lum-stress)
(insert4d! 'ew 'app 's2to4/2to4 'f-b-sing 1100 lum-stress)
(insert4d! 'ew 'stud 's2to4/2to4 'f-b-sing 500 lum-stress)
(insert4d! 'ew 'ss 's2to4/2to4 'f-b-rep 1500 lum-stress)
(insert4d! 'ew 'n1 's2to4/2to4 'f-b-rep 1300 lum-stress)
(insert4d! 'ew 'n2 's2to4/2to4 'f-b-rep 1050 lum-stress)
(insert4d! 'ew 'n3 's2to4/2to4 'f-b-rep 575 lum-stress)
(insert4d! 'ew 'app 's2to4/2to4 'f-b-rep 1300 lum-stress)
(insert4d! 'ew 'stud 's2to4/2to4 'f-b-rep 575 lum-stress)
(insert4d! 'ew 'ss 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'ew 'n1 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'ew 'n2 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'ew 'n3 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'ew 'app 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'ew 'stud 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'ew 'ss 's2to4/2to4 'f-c-perp 270 lum-stress)
(insert4d! 'ew 'n1 's2to4/2to4 'f-c-perp 270 lum-stress)
(insert4d! 'ew 'n2 's2to4/2to4 'f-c-perp 270 lum-stress)
(insert4d! 'ew 'n3 's2to4/2to4 'f-c-perp 270 lum-stress)
(insert4d! 'ew 'app 's2to4/2to4 'f-c-perp 270 lum-stress)
(insert4d! 'ew 'stud 's2to4/2to4 'f-c-perp 270 lum-stress)
(insert4d! 'ew 'ss 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'ew 'n1 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'ew 'n2 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'ew 'n3 's2to4/2to4 'e 900000 lum-stress)
(insert4d! 'ew 'app 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'ew 'stud 's2to4/2to4 'e 900000 lum-stress)

(insert4d! 'ew 'con 's2to4/4 'f-b-sing 650 lum-stress)
(insert4d! 'ew 'con 's2to4/4 'f-b-rep 750 lum-stress)
(insert4d! 'ew 'con 's2to4/4 'f-v 60 lum-stress)
(insert4d! 'ew 'con 's2to4/4 'f-c-perp 270 lum-stress)
(insert4d! 'ew 'con 's2to4/4 'e 900000 lum-stress)
(insert4d! 'ew 'std 's2to4/4 'f-b-sing 375 lum-stress)
(insert4d! 'ew 'std 's2to4/4 'f-b-rep 425 lum-stress)
(insert4d! 'ew 'std 's2to4/4 'f-v 60 lum-stress)
(insert4d! 'ew 'std 's2to4/4 'f-c-perp 270 lum-stress)
```

```
(insert4d! 'ew 'std 's2to4/4 'e 900000 lum-stress)
(insert4d! 'ew 'util 's2to4/4 'f-b-sing 175 lum-stress)
(insert4d! 'ew 'util 's2to4/4 'f-b-rep 200 lum-stress)
(insert4d! 'ew 'util 's2to4/4 'f-v 60 lum-stress)
(insert4d! 'ew 'util 's2to4/4 'f-c-perp 270 lum-stress)
(insert4d! 'ew 'util 's2to4/4 'e 900000 lum-stress)

(insert4d! 'ew 'ss 's2to4/>=5 'f-b-sing 1150 lum-stress)
(insert4d! 'ew 'n1 's2to4/>=5 'f-b-sing 950 lum-stress)
(insert4d! 'ew 'n2 's2to4/>=5 'f-b-sing 775 lum-stress)
(insert4d! 'ew 'n3 's2to4/>=5 'f-b-sing 450 lum-stress)
(insert4d! 'ew 'app 's2to4/>=5 'f-b-sing 950 lum-stress)
(insert4d! 'ew 'stud 's2to4/>=5 'f-b-sing 450 lum-stress)
(insert4d! 'ew 'ss 's2to4/>=5 'f-b-rep 1300 lum-stress)
(insert4d! 'ew 'n1 's2to4/>=5 'f-b-rep 1100 lum-stress)
(insert4d! 'ew 'n2 's2to4/>=5 'f-b-rep 900 lum-stress)
(insert4d! 'ew 'n3 's2to4/>=5 'f-b-rep 525 lum-stress)
(insert4d! 'ew 'app 's2to4/>=5 'f-b-rep 1100 lum-stress)
(insert4d! 'ew 'stud 's2to4/>=5 'f-b-rep 525 lum-stress)
(insert4d! 'ew 'ss 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ew 'n1 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ew 'n2 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ew 'n3 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ew 'app 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ew 'stud 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ew 'ss 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'ew 'n1 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'ew 'n2 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'ew 'n3 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'ew 'app 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'ew 'stud 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'ew 'ss 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'ew 'n1 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'ew 'n2 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'ew 'n3 's2to4/>=5 'e 900000 lum-stress)
(insert4d! 'ew 'app 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'ew 'stud 's2to4/>=5 'e 900000 lum-stress)

;;Engelmann Spruce-Alpine Fir (Engelmann Spruce-Lodgepole Pine)

(insert4d! 'esaf 'ss 's2to4/2to4 'f-b-sing 1350 lum-stress)
(insert4d! 'esaf 'n1 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'esaf 'n2 's2to4/2to4 'f-b-sing 950 lum-stress)
(insert4d! 'esaf 'n3 's2to4/2to4 'f-b-sing 525 lum-stress)
(insert4d! 'esaf 'app 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'esaf 'stud 's2to4/2to4 'f-b-sing 525 lum-stress)
(insert4d! 'esaf 'ss 's2to4/2to4 'f-b-rep 1550 lum-stress)
(insert4d! 'esaf 'n1 's2to4/2to4 'f-b-rep 1350 lum-stress)
(insert4d! 'esaf 'n2 's2to4/2to4 'f-b-rep 1100 lum-stress)
(insert4d! 'esaf 'n3 's2to4/2to4 'f-b-rep 600 lum-stress)
(insert4d! 'esaf 'app 's2to4/2to4 'f-b-rep 1350 lum-stress)
(insert4d! 'esaf 'stud 's2to4/2to4 'f-b-rep 600 lum-stress)
(insert4d! 'esaf 'ss 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'esaf 'n1 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'esaf 'n2 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'esaf 'n3 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'esaf 'app 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'esaf 'stud 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'esaf 'ss 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'n1 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'n2 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'n3 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'app 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'stud 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'ss 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'esaf 'n1 's2to4/2to4 'e 1300000 lum-stress)
```

```
(insert4d! 'esaf 'n2 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'esaf 'n3 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'esaf 'app 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'esaf 'stud 's2to4/2to4 'e 1000000 lum-stress)

(insert4d! 'esaf 'con 's2to4/4 'f-b-sing 700 lum-stress)
(insert4d! 'esaf 'con 's2to4/4 'f-b-rep 800 lum-stress)
(insert4d! 'esaf 'con 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'esaf 'con 's2to4/4 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'con 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'esaf 'std 's2to4/4 'f-b-sing 375 lum-stress)
(insert4d! 'esaf 'std 's2to4/4 'f-b-rep 450 lum-stress)
(insert4d! 'esaf 'std 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'esaf 'std 's2to4/4 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'std 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'esaf 'util 's2to4/4 'f-b-sing 175 lum-stress)
(insert4d! 'esaf 'util 's2to4/4 'f-b-rep 200 lum-stress)
(insert4d! 'esaf 'util 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'esaf 'util 's2to4/4 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'util 's2to4/4 'e 1000000 lum-stress)

(insert4d! 'esaf 'ss 's2to4/>=5 'f-b-sing 1200 lum-stress)
(insert4d! 'esaf 'n1 's2to4/>=5 'f-b-sing 1000 lum-stress)
(insert4d! 'esaf 'n2 's2to4/>=5 'f-b-sing 825 lum-stress)
(insert4d! 'esaf 'n3 's2to4/>=5 'f-b-sing 475 lum-stress)
(insert4d! 'esaf 'app 's2to4/>=5 'f-b-sing 1000 lum-stress)
(insert4d! 'esaf 'stud 's2to4/>=5 'f-b-sing 475 lum-stress)
(insert4d! 'esaf 'ss 's2to4/>=5 'f-b-rep 1350 lum-stress)
(insert4d! 'esaf 'n1 's2to4/>=5 'f-b-rep 1150 lum-stress)
(insert4d! 'esaf 'n2 's2to4/>=5 'f-b-rep 950 lum-stress)
(insert4d! 'esaf 'n3 's2to4/>=5 'f-b-rep 550 lum-stress)
(insert4d! 'esaf 'app 's2to4/>=5 'f-b-rep 1150 lum-stress)
(insert4d! 'esaf 'stud 's2to4/>=5 'f-b-rep 550 lum-stress)
(insert4d! 'esaf 'ss 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'esaf 'n1 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'esaf 'n2 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'esaf 'n3 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'esaf 'app 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'esaf 'stud 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'esaf 'ss 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'n1 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'n2 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'n3 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'app 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'stud 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'esaf 'ss 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'esaf 'n1 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'esaf 'n2 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'esaf 'n3 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'esaf 'app 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'esaf 'stud 's2to4/>=5 'e 1000000 lum-stress)

;;Hem-Fir

(insert4d! 'hf 'ss 's2to4/2to4 'f-b-sing 1650 lum-stress)
(insert4d! 'hf 'n1 's2to4/2to4 'f-b-sing 1400 lum-stress)
(insert4d! 'hf 'n2 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'hf 'n3 's2to4/2to4 'f-b-sing 650 lum-stress)
(insert4d! 'hf 'app 's2to4/2to4 'f-b-sing 1400 lum-stress)
(insert4d! 'hf 'stud 's2to4/2to4 'f-b-sing 650 lum-stress)
(insert4d! 'hf 'ss 's2to4/2to4 'f-b-rep 1900 lum-stress)
(insert4d! 'hf 'n1 's2to4/2to4 'f-b-rep 1600 lum-stress)
(insert4d! 'hf 'n2 's2to4/2to4 'f-b-rep 1350 lum-stress)
(insert4d! 'hf 'n3 's2to4/2to4 'f-b-rep 725 lum-stress)
(insert4d! 'hf 'app 's2to4/2to4 'f-b-rep 1600 lum-stress)
(insert4d! 'hf 'stud 's2to4/2to4 'f-b-rep 725 lum-stress)
```

```
(insert4d! 'hf 'ss 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'hf 'n1 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'hf 'n2 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'hf 'n3 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'hf 'app 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'hf 'stud 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'hf 'ss 's2to4/2to4 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'n1 's2to4/2to4 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'n2 's2to4/2to4 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'n3 's2to4/2to4 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'app 's2to4/2to4 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'stud 's2to4/2to4 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'ss 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'hf 'n1 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'hf 'n2 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'hf 'n3 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'hf 'app 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'hf 'stud 's2to4/2to4 'e 1200000 lum-stress)

(insert4d! 'hf 'con 's2to4/4 'f-b-sing 825 lum-stress)
(insert4d! 'hf 'con 's2to4/4 'f-b-rep 975 lum-stress)
(insert4d! 'hf 'con 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'hf 'con 's2to4/4 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'con 's2to4/4 'e 1200000 lum-stress)
(insert4d! 'hf 'std 's2to4/4 'f-b-sing 475 lum-stress)
(insert4d! 'hf 'std 's2to4/4 'f-b-rep 550 lum-stress)
(insert4d! 'hf 'std 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'hf 'std 's2to4/4 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'std 's2to4/4 'e 1200000 lum-stress)
(insert4d! 'hf 'util 's2to4/4 'f-b-sing 225 lum-stress)
(insert4d! 'hf 'util 's2to4/4 'f-b-rep 250 lum-stress)
(insert4d! 'hf 'util 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'hf 'util 's2to4/4 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'util 's2to4/4 'e 1200000 lum-stress)

(insert4d! 'hf 'ss 's2to4/>=5 'f-b-sing 1400 lum-stress)
(insert4d! 'hf 'n1 's2to4/>=5 'f-b-sing 1200 lum-stress)
(insert4d! 'hf 'n2 's2to4/>=5 'f-b-sing 1000 lum-stress)
(insert4d! 'hf 'n3 's2to4/>=5 'f-b-sing 575 lum-stress)
(insert4d! 'hf 'app 's2to4/>=5 'f-b-sing 1200 lum-stress)
(insert4d! 'hf 'stud 's2to4/>=5 'f-b-sing 575 lum-stress)
(insert4d! 'hf 'ss 's2to4/>=5 'f-b-rep 1650 lum-stress)
(insert4d! 'hf 'n1 's2to4/>=5 'f-b-rep 1400 lum-stress)
(insert4d! 'hf 'n2 's2to4/>=5 'f-b-rep 1150 lum-stress)
(insert4d! 'hf 'n3 's2to4/>=5 'f-b-rep 675 lum-stress)
(insert4d! 'hf 'app 's2to4/>=5 'f-b-rep 1400 lum-stress)
(insert4d! 'hf 'stud 's2to4/>=5 'f-b-rep 675 lum-stress)
(insert4d! 'hf 'ss 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'hf 'n1 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'hf 'n2 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'hf 'n3 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'hf 'app 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'hf 'stud 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'hf 'ss 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'n1 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'n2 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'n3 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'app 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'stud 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'hf 'ss 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'hf 'n1 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'hf 'n2 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'hf 'n3 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'hf 'app 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'hf 'stud 's2to4/>=5 'e 1200000 lum-stress)
```

```
;;Idaho White Pine

(insert4d! 'iwp 'ss 's2to4/2to4 'f-b-sing 1350 lum-stress)
(insert4d! 'iwp 'n1 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'iwp 'n2 's2to4/2to4 'f-b-sing 925 lum-stress)
(insert4d! 'iwp 'n3 's2to4/2to4 'f-b-sing 525 lum-stress)
(insert4d! 'iwp 'app 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'iwp 'stud 's2to4/2to4 'f-b-sing 525 lum-stress)
(insert4d! 'iwp 'ss 's2to4/2to4 'f-b-rep 1550 lum-stress)
(insert4d! 'iwp 'n1 's2to4/2to4 'f-b-rep 1300 lum-stress)
(insert4d! 'iwp 'n2 's2to4/2to4 'f-b-rep 1050 lum-stress)
(insert4d! 'iwp 'n3 's2to4/2to4 'f-b-rep 600 lum-stress)
(insert4d! 'iwp 'app 's2to4/2to4 'f-b-rep 1300 lum-stress)
(insert4d! 'iwp 'stud 's2to4/2to4 'f-b-rep 600 lum-stress)
(insert4d! 'iwp 'ss 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'iwp 'n1 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'iwp 'n2 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'iwp 'n3 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'iwp 'app 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'iwp 'stud 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'iwp 'ss 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'n1 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'n2 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'n3 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'app 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'stud 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'ss 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'iwp 'n1 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'iwp 'n2 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'iwp 'n3 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'iwp 'app 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'iwp 'stud 's2to4/2to4 'e 1200000 lum-stress)

(insert4d! 'iwp 'con 's2to4/4 'f-b-sing 675 lum-stress)
(insert4d! 'iwp 'con 's2to4/4 'f-b-rep 775 lum-stress)
(insert4d! 'iwp 'con 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'iwp 'con 's2to4/4 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'con 's2to4/4 'e 1200000 lum-stress)
(insert4d! 'iwp 'std 's2to4/4 'f-b-sing 375 lum-stress)
(insert4d! 'iwp 'std 's2to4/4 'f-b-rep 425 lum-stress)
(insert4d! 'iwp 'std 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'iwp 'std 's2to4/4 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'std 's2to4/4 'e 1200000 lum-stress)
(insert4d! 'iwp 'util 's2to4/4 'f-b-sing 175 lum-stress)
(insert4d! 'iwp 'util 's2to4/4 'f-b-rep 200 lum-stress)
(insert4d! 'iwp 'util 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'iwp 'util 's2to4/4 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'util 's2to4/4 'e 1200000 lum-stress)

(insert4d! 'iwp 'ss 's2to4/>=5 'f-b-sing 1150 lum-stress)
(insert4d! 'iwp 'n1 's2to4/>=5 'f-b-sing 975 lum-stress)
(insert4d! 'iwp 'n2 's2to4/>=5 'f-b-sing 800 lum-stress)
(insert4d! 'iwp 'n3 's2to4/>=5 'f-b-sing 475 lum-stress)
(insert4d! 'iwp 'app 's2to4/>=5 'f-b-sing 975 lum-stress)
(insert4d! 'iwp 'stud 's2to4/>=5 'f-b-sing 475 lum-stress)
(insert4d! 'iwp 'ss 's2to4/>=5 'f-b-rep 1300 lum-stress)
(insert4d! 'iwp 'n1 's2to4/>=5 'f-b-rep 1100 lum-stress)
(insert4d! 'iwp 'n2 's2to4/>=5 'f-b-rep 925 lum-stress)
(insert4d! 'iwp 'n3 's2to4/>=5 'f-b-rep 550 lum-stress)
(insert4d! 'iwp 'app 's2to4/>=5 'f-b-rep 1100 lum-stress)
(insert4d! 'iwp 'stud 's2to4/>=5 'f-b-rep 550 lum-stress)
(insert4d! 'iwp 'ss 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'iwp 'n1 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'iwp 'n2 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'iwp 'n3 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'iwp 'app 's2to4/>=5 'f-v 70 lum-stress)
```

```
(insert4d! 'iwp 'stud 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'iwp 'ss 's2to4/>=5 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'n1 's2to4/>=5 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'n2 's2to4/>=5 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'n3 's2to4/>=5 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'app 's2to4/>=5 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'stud 's2to4/>=5 'f-c-perp 315 lum-stress)
(insert4d! 'iwp 'ss 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'iwp 'n1 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'iwp 'n2 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'iwp 'n3 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'iwp 'app 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'iwp 'stud 's2to4/>=5 'e 1200000 lum-stress)

;;Lodgepole Pine

(insert4d! 'lp 'ss 's2to4/2to4 'f-b-sing 1500 lum-stress)
(insert4d! 'lp 'n1 's2to4/2to4 'f-b-sing 1300 lum-stress)
(insert4d! 'lp 'n2 's2to4/2to4 'f-b-sing 1050 lum-stress)
(insert4d! 'lp 'n3 's2to4/2to4 'f-b-sing 600 lum-stress)
(insert4d! 'lp 'app 's2to4/2to4 'f-b-sing 1300 lum-stress)
(insert4d! 'lp 'stud 's2to4/2to4 'f-b-sing 600 lum-stress)
(insert4d! 'lp 'ss 's2to4/2to4 'f-b-rep 1750 lum-stress)
(insert4d! 'lp 'n1 's2to4/2to4 'f-b-rep 1500 lum-stress)
(insert4d! 'lp 'n2 's2to4/2to4 'f-b-rep 1200 lum-stress)
(insert4d! 'lp 'n3 's2to4/2to4 'f-b-rep 675 lum-stress)
(insert4d! 'lp 'app 's2to4/2to4 'f-b-rep 1500 lum-stress)
(insert4d! 'lp 'stud 's2to4/2to4 'f-b-rep 675 lum-stress)
(insert4d! 'lp 'ss 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'lp 'n1 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'lp 'n2 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'lp 'n3 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'lp 'app 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'lp 'stud 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'lp 'ss 's2to4/2to4 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'n1 's2to4/2to4 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'n2 's2to4/2to4 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'n3 's2to4/2to4 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'app 's2to4/2to4 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'stud 's2to4/2to4 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'ss 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'lp 'n1 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'lp 'n2 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'lp 'n3 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'lp 'app 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'lp 'stud 's2to4/2to4 'e 1000000 lum-stress)

(insert4d! 'lp 'con 's2to4/4 'f-b-sing 775 lum-stress)
(insert4d! 'lp 'con 's2to4/4 'f-b-rep 875 lum-stress)
(insert4d! 'lp 'con 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'lp 'con 's2to4/4 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'con 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'lp 'std 's2to4/4 'f-b-sing 425 lum-stress)
(insert4d! 'lp 'std 's2to4/4 'f-b-rep 500 lum-stress)
(insert4d! 'lp 'std 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'lp 'std 's2to4/4 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'std 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'lp 'util 's2to4/4 'f-b-sing 200 lum-stress)
(insert4d! 'lp 'util 's2to4/4 'f-b-rep 225 lum-stress)
(insert4d! 'lp 'util 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'lp 'util 's2to4/4 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'util 's2to4/4 'e 1000000 lum-stress)

(insert4d! 'lp 'ss 's2to4/>=5 'f-b-sing 1300 lum-stress)
(insert4d! 'lp 'n1 's2to4/>=5 'f-b-sing 1100 lum-stress)
(insert4d! 'lp 'n2 's2to4/>=5 'f-b-sing 925 lum-stress)
```

```
(insert4d! 'lp 'n3 's2to4/>=5 'f-b-sing 525 lum-stress)
(insert4d! 'lp 'app 's2to4/>=5 'f-b-sing 1100 lum-stress)
(insert4d! 'lp 'stud 's2to4/>=5 'f-b-sing 525 lum-stress)
(insert4d! 'lp 'ss 's2to4/>=5 'f-b-rep 1500 lum-stress)
(insert4d! 'lp 'n1 's2to4/>=5 'f-b-rep 1300 lum-stress)
(insert4d! 'lp 'n2 's2to4/>=5 'f-b-rep 1050 lum-stress)
(insert4d! 'lp 'n3 's2to4/>=5 'f-b-rep 625 lum-stress)
(insert4d! 'lp 'app 's2to4/>=5 'f-b-rep 1300 lum-stress)
(insert4d! 'lp 'stud 's2to4/>=5 'f-b-rep 625 lum-stress)
(insert4d! 'lp 'ss 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'lp 'n1 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'lp 'n2 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'lp 'n3 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'lp 'app 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'lp 'stud 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'lp 'ss 's2to4/>=5 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'n1 's2to4/>=5 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'n2 's2to4/>=5 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'n3 's2to4/>=5 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'app 's2to4/>=5 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'stud 's2to4/>=5 'f-c-perp 400 lum-stress)
(insert4d! 'lp 'ss 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'lp 'n1 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'lp 'n2 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'lp 'n3 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'lp 'app 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'lp 'stud 's2to4/>=5 'e 1000000 lum-stress)

;;Mountain Hemlock

(insert4d! 'mh 'ss 's2to4/2to4 'f-b-sing 1750 lum-stress)
(insert4d! 'mh 'n1 's2to4/2to4 'f-b-sing 1450 lum-stress)
(insert4d! 'mh 'n2 's2to4/2to4 'f-b-sing 1200 lum-stress)
(insert4d! 'mh 'n3 's2to4/2to4 'f-b-sing 675 lum-stress)
(insert4d! 'mh 'app 's2to4/2to4 'f-b-sing 1450 lum-stress)
(insert4d! 'mh 'stud 's2to4/2to4 'f-b-sing 675 lum-stress)
(insert4d! 'mh 'ss 's2to4/2to4 'f-b-rep 2000 lum-stress)
(insert4d! 'mh 'n1 's2to4/2to4 'f-b-rep 1700 lum-stress)
(insert4d! 'mh 'n2 's2to4/2to4 'f-b-rep 1400 lum-stress)
(insert4d! 'mh 'n3 's2to4/2to4 'f-b-rep 775 lum-stress)
(insert4d! 'mh 'app 's2to4/2to4 'f-b-rep 1700 lum-stress)
(insert4d! 'mh 'stud 's2to4/2to4 'f-b-rep 775 lum-stress)
(insert4d! 'mh 'ss 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'mh 'n1 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'mh 'n2 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'mh 'n3 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'mh 'app 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'mh 'stud 's2to4/2to4 'f-v 95 lum-stress)
(insert4d! 'mh 'ss 's2to4/2to4 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'n1 's2to4/2to4 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'n2 's2to4/2to4 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'n3 's2to4/2to4 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'app 's2to4/2to4 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'stud 's2to4/2to4 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'ss 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'mh 'n1 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'mh 'n2 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'mh 'n3 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'mh 'app 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'mh 'stud 's2to4/2to4 'e 1000000 lum-stress)

(insert4d! 'mh 'con 's2to4/4 'f-b-sing 875 lum-stress)
(insert4d! 'mh 'con 's2to4/4 'f-b-rep 1000 lum-stress)
(insert4d! 'mh 'con 's2to4/4 'f-v 95 lum-stress)
(insert4d! 'mh 'con 's2to4/4 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'con 's2to4/4 'e 1000000 lum-stress)
```

```
(insert4d! 'mh 'std 's2to4/4 'f-b-sing 500 lum-stress)
(insert4d! 'mh 'std 's2to4/4 'f-b-rep 575 lum-stress)
(insert4d! 'mh 'std 's2to4/4 'f-v 95 lum-stress)
(insert4d! 'mh 'std 's2to4/4 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'std 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'mh 'util 's2to4/4 'f-b-sing 225 lum-stress)
(insert4d! 'mh 'util 's2to4/4 'f-b-rep 275 lum-stress)
(insert4d! 'mh 'util 's2to4/4 'f-v 95 lum-stress)
(insert4d! 'mh 'util 's2to4/4 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'util 's2to4/4 'e 1000000 lum-stress)

(insert4d! 'mh 'ss 's2to4/>=5 'f-b-sing 1500 lum-stress)
(insert4d! 'mh 'n1 's2to4/>=5 'f-b-sing 1250 lum-stress)
(insert4d! 'mh 'n2 's2to4/>=5 'f-b-sing 1050 lum-stress)
(insert4d! 'mh 'n3 's2to4/>=5 'f-b-sing 625 lum-stress)
(insert4d! 'mh 'app 's2to4/>=5 'f-b-sing 1250 lum-stress)
(insert4d! 'mh 'stud 's2to4/>=5 'f-b-sing 625 lum-stress)
(insert4d! 'mh 'ss 's2to4/>=5 'f-b-rep 1700 lum-stress)
(insert4d! 'mh 'n1 's2to4/>=5 'f-b-rep 1450 lum-stress)
(insert4d! 'mh 'n2 's2to4/>=5 'f-b-rep 1200 lum-stress)
(insert4d! 'mh 'n3 's2to4/>=5 'f-b-rep 700 lum-stress)
(insert4d! 'mh 'app 's2to4/>=5 'f-b-rep 1450 lum-stress)
(insert4d! 'mh 'stud 's2to4/>=5 'f-b-rep 700 lum-stress)
(insert4d! 'mh 'ss 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'mh 'n1 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'mh 'n2 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'mh 'n3 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'mh 'app 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'mh 'stud 's2to4/>=5 'f-v 95 lum-stress)
(insert4d! 'mh 'ss 's2to4/>=5 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'n1 's2to4/>=5 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'n2 's2to4/>=5 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'n3 's2to4/>=5 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'app 's2to4/>=5 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'stud 's2to4/>=5 'f-c-perp 570 lum-stress)
(insert4d! 'mh 'ss 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'mh 'n1 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'mh 'n2 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'mh 'n3 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'mh 'app 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'mh 'stud 's2to4/>=5 'e 1000000 lum-stress)

;;Mountain Hemlock-Hem-Fir

(insert4d! 'mhhf 'ss 's2to4/2to4 'f-b-sing 1650 lum-stress)
(insert4d! 'mhhf 'n1 's2to4/2to4 'f-b-sing 1400 lum-stress)
(insert4d! 'mhhf 'n2 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'mhhf 'n3 's2to4/2to4 'f-b-sing 650 lum-stress)
(insert4d! 'mhhf 'app 's2to4/2to4 'f-b-sing 1400 lum-stress)
(insert4d! 'mhhf 'stud 's2to4/2to4 'f-b-sing 650 lum-stress)
(insert4d! 'mhhf 'ss 's2to4/2to4 'f-b-rep 1900 lum-stress)
(insert4d! 'mhhf 'n1 's2to4/2to4 'f-b-rep 1600 lum-stress)
(insert4d! 'mhhf 'n2 's2to4/2to4 'f-b-rep 1350 lum-stress)
(insert4d! 'mhhf 'n3 's2to4/2to4 'f-b-rep 725 lum-stress)
(insert4d! 'mhhf 'app 's2to4/2to4 'f-b-rep 1600 lum-stress)
(insert4d! 'mhhf 'stud 's2to4/2to4 'f-b-rep 725 lum-stress)
(insert4d! 'mhhf 'ss 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'mhhf 'n1 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'mhhf 'n2 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'mhhf 'n3 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'mhhf 'app 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'mhhf 'stud 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'mhhf 'ss 's2to4/2to4 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'n1 's2to4/2to4 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'n2 's2to4/2to4 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'n3 's2to4/2to4 'f-c-perp 405 lum-stress)
```

```
(insert4d! 'mhhf 'app 's2to4/2to4 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'stud 's2to4/2to4 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'ss 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'mhhf 'n1 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'mhhf 'n2 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'mhhf 'n3 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'mhhf 'app 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'mhhf 'stud 's2to4/2to4 'e 1000000 lum-stress)

(insert4d! 'mhhf 'con 's2to4/4 'f-b-sing 825 lum-stress)
(insert4d! 'mhhf 'con 's2to4/4 'f-b-rep 975 lum-stress)
(insert4d! 'mhhf 'con 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'mhhf 'con 's2to4/4 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'con 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'mhhf 'std 's2to4/4 'f-b-sing 475 lum-stress)
(insert4d! 'mhhf 'std 's2to4/4 'f-b-rep 550 lum-stress)
(insert4d! 'mhhf 'std 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'mhhf 'std 's2to4/4 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'std 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'mhhf 'util 's2to4/4 'f-b-sing 225 lum-stress)
(insert4d! 'mhhf 'util 's2to4/4 'f-b-rep 250 lum-stress)
(insert4d! 'mhhf 'util 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'mhhf 'util 's2to4/4 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'util 's2to4/4 'e 1000000 lum-stress)

(insert4d! 'mhhf 'ss 's2to4/>=5 'f-b-sing 1400 lum-stress)
(insert4d! 'mhhf 'n1 's2to4/>=5 'f-b-sing 1200 lum-stress)
(insert4d! 'mhhf 'n2 's2to4/>=5 'f-b-sing 1000 lum-stress)
(insert4d! 'mhhf 'n3 's2to4/>=5 'f-b-sing 575 lum-stress)
(insert4d! 'mhhf 'app 's2to4/>=5 'f-b-sing 1200 lum-stress)
(insert4d! 'mhhf 'stud 's2to4/>=5 'f-b-sing 575 lum-stress)
(insert4d! 'mhhf 'ss 's2to4/>=5 'f-b-rep 1650 lum-stress)
(insert4d! 'mhhf 'n1 's2to4/>=5 'f-b-rep 1400 lum-stress)
(insert4d! 'mhhf 'n2 's2to4/>=5 'f-b-rep 1150 lum-stress)
(insert4d! 'mhhf 'n3 's2to4/>=5 'f-b-rep 675 lum-stress)
(insert4d! 'mhhf 'app 's2to4/>=5 'f-b-rep 1400 lum-stress)
(insert4d! 'mhhf 'stud 's2to4/>=5 'f-b-rep 675 lum-stress)
(insert4d! 'mhhf 'ss 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'mhhf 'n1 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'mhhf 'n2 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'mhhf 'n3 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'mhhf 'app 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'mhhf 'stud 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'mhhf 'ss 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'n1 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'n2 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'n3 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'app 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'stud 's2to4/>=5 'f-c-perp 405 lum-stress)
(insert4d! 'mhhf 'ss 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'mhhf 'n1 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'mhhf 'n2 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'mhhf 'n3 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'mhhf 'app 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'mhhf 'stud 's2to4/>=5 'e 1000000 lum-stress)

;;Northern Aspen

(insert4d! 'na 'ss 's2to4/2to4 'f-b-sing 1300 lum-stress)
(insert4d! 'na 'n1 's2to4/2to4 'f-b-sing 1100 lum-stress)
(insert4d! 'na 'n2 's2to4/2to4 'f-b-sing 900 lum-stress)
(insert4d! 'na 'n3 's2to4/2to4 'f-b-sing 500 lum-stress)
(insert4d! 'na 'app 's2to4/2to4 'f-b-sing 1100 lum-stress)
(insert4d! 'na 'stud 's2to4/2to4 'f-b-sing 500 lum-stress)
(insert4d! 'na 'ss 's2to4/2to4 'f-b-rep 1500 lum-stress)
(insert4d! 'na 'n1 's2to4/2to4 'f-b-rep 1250 lum-stress)
```

```
(insert4d! 'na 'n2 's2to4/2to4 'f-b-rep 1050 lum-stress)
(insert4d! 'na 'n3 's2to4/2to4 'f-b-rep 575 lum-stress)
(insert4d! 'na 'app 's2to4/2to4 'f-b-rep 1250 lum-stress)
(insert4d! 'na 'stud 's2to4/2to4 'f-b-rep 575 lum-stress)
(insert4d! 'na 'ss 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'na 'n1 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'na 'n2 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'na 'n3 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'na 'app 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'na 'stud 's2to4/2to4 'f-v 60 lum-stress)
(insert4d! 'na 'ss 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'na 'n1 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'na 'n2 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'na 'n3 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'na 'app 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'na 'stud 's2to4/2to4 'f-c-perp 320 lum-stress)
(insert4d! 'na 'ss 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'na 'n1 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'na 'n2 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'na 'n3 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'na 'app 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'na 'stud 's2to4/2to4 'e 1100000 lum-stress)

(insert4d! 'na 'con 's2to4/4 'f-b-sing 650 lum-stress)
(insert4d! 'na 'con 's2to4/4 'f-b-rep 750 lum-stress)
(insert4d! 'na 'con 's2to4/4 'f-v 60 lum-stress)
(insert4d! 'na 'con 's2to4/4 'f-c-perp 320 lum-stress)
(insert4d! 'na 'con 's2to4/4 'e 1100000 lum-stress)
(insert4d! 'na 'std 's2to4/4 'f-b-sing 350 lum-stress)
(insert4d! 'na 'std 's2to4/4 'f-b-rep 425 lum-stress)
(insert4d! 'na 'std 's2to4/4 'f-v 60 lum-stress)
(insert4d! 'na 'std 's2to4/4 'f-c-perp 320 lum-stress)
(insert4d! 'na 'std 's2to4/4 'e 1100000 lum-stress)
(insert4d! 'na 'util 's2to4/4 'f-b-sing 175 lum-stress)
(insert4d! 'na 'util 's2to4/4 'f-b-rep 200 lum-stress)
(insert4d! 'na 'util 's2to4/4 'f-v 60 lum-stress)
(insert4d! 'na 'util 's2to4/4 'f-c-perp 320 lum-stress)
(insert4d! 'na 'util 's2to4/4 'e 1100000 lum-stress)

(insert4d! 'na 'ss 's2to4/>=5 'f-b-sing 1100 lum-stress)
(insert4d! 'na 'n1 's2to4/>=5 'f-b-sing 950 lum-stress)
(insert4d! 'na 'n2 's2to4/>=5 'f-b-sing 775 lum-stress)
(insert4d! 'na 'n3 's2to4/>=5 'f-b-sing 450 lum-stress)
(insert4d! 'na 'app 's2to4/>=5 'f-b-sing 950 lum-stress)
(insert4d! 'na 'stud 's2to4/>=5 'f-b-sing 450 lum-stress)
(insert4d! 'na 'ss 's2to4/>=5 'f-b-rep 1250 lum-stress)
(insert4d! 'na 'n1 's2to4/>=5 'f-b-rep 1100 lum-stress)
(insert4d! 'na 'n2 's2to4/>=5 'f-b-rep 900 lum-stress)
(insert4d! 'na 'n3 's2to4/>=5 'f-b-rep 525 lum-stress)
(insert4d! 'na 'app 's2to4/>=5 'f-b-rep 1100 lum-stress)
(insert4d! 'na 'stud 's2to4/>=5 'f-b-rep 525 lum-stress)
(insert4d! 'na 'ss 's2to4/>=5 'f-v 60 lum-stress)
(insert4d! 'na 'n1 's2to4/>=5 'f-v 60 lum-stress)
(insert4d! 'na 'n2 's2to4/>=5 'f-v 60 lum-stress)
(insert4d! 'na 'n3 's2to4/>=5 'f-v 60 lum-stress)
(insert4d! 'na 'app 's2to4/>=5 'f-v 60 lum-stress)
(insert4d! 'na 'stud 's2to4/>=5 'f-v 60 lum-stress)
(insert4d! 'na 'ss 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'na 'n1 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'na 'n2 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'na 'n3 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'na 'app 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'na 'stud 's2to4/>=5 'f-c-perp 320 lum-stress)
(insert4d! 'na 'ss 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'na 'n1 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'na 'n2 's2to4/>=5 'e 1200000 lum-stress)
```

```
(insert4d! 'na 'n3 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'na 'app 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'na 'stud 's2to4/>=5 'e 1100000 lum-stress)

;;Northern Pine

(insert4d! 'np 'ss 's2to4/2to4 'f-b-sing 1650 lum-stress)
(insert4d! 'np 'n1 's2to4/2to4 'f-b-sing 1400 lum-stress)
(insert4d! 'np 'n2 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'np 'n3 's2to4/2to4 'f-b-sing 625 lum-stress)
(insert4d! 'np 'app 's2to4/2to4 'f-b-sing 1200 lum-stress)
(insert4d! 'np 'stud 's2to4/2to4 'f-b-sing 625 lum-stress)
(insert4d! 'np 'ss 's2to4/2to4 'f-b-rep 1850 lum-stress)
(insert4d! 'np 'n1 's2to4/2to4 'f-b-rep 1600 lum-stress)
(insert4d! 'np 'n2 's2to4/2to4 'f-b-rep 1300 lum-stress)
(insert4d! 'np 'n3 's2to4/2to4 'f-b-rep 725 lum-stress)
(insert4d! 'np 'app 's2to4/2to4 'f-b-rep 1400 lum-stress)
(insert4d! 'np 'stud 's2to4/2to4 'f-b-rep 725 lum-stress)
(insert4d! 'np 'ss 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'np 'n1 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'np 'n2 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'np 'n3 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'np 'app 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'np 'stud 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'np 'ss 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'np 'n1 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'np 'n2 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'np 'n3 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'np 'app 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'np 'stud 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'np 'ss 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'np 'n1 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'np 'n2 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'np 'n3 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'np 'app 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'np 'stud 's2to4/2to4 'e 1100000 lum-stress)

(insert4d! 'np 'con 's2to4/4 'f-b-sing 825 lum-stress)
(insert4d! 'np 'con 's2to4/4 'f-b-rep 950 lum-stress)
(insert4d! 'np 'con 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'np 'con 's2to4/4 'f-c-perp 435 lum-stress)
(insert4d! 'np 'con 's2to4/4 'e 1100000 lum-stress)
(insert4d! 'np 'std 's2to4/4 'f-b-sing 450 lum-stress)
(insert4d! 'np 'std 's2to4/4 'f-b-rep 525 lum-stress)
(insert4d! 'np 'std 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'np 'std 's2to4/4 'f-c-perp 435 lum-stress)
(insert4d! 'np 'std 's2to4/4 'e 1100000 lum-stress)
(insert4d! 'np 'util 's2to4/4 'f-b-sing 225 lum-stress)
(insert4d! 'np 'util 's2to4/4 'f-b-rep 250 lum-stress)
(insert4d! 'np 'util 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'np 'util 's2to4/4 'f-c-perp 435 lum-stress)
(insert4d! 'np 'util 's2to4/4 'e 1100000 lum-stress)

(insert4d! 'np 'ss 's2to4/>=5 'f-b-sing 1400 lum-stress)
(insert4d! 'np 'n1 's2to4/>=5 'f-b-sing 1200 lum-stress)
(insert4d! 'np 'n2 's2to4/>=5 'f-b-sing 950 lum-stress)
(insert4d! 'np 'n3 's2to4/>=5 'f-b-sing 575 lum-stress)
(insert4d! 'np 'app 's2to4/>=5 'f-b-sing 1200 lum-stress)
(insert4d! 'np 'stud 's2to4/>=5 'f-b-sing 575 lum-stress)
(insert4d! 'np 'ss 's2to4/>=5 'f-b-rep 1600 lum-stress)
(insert4d! 'np 'n1 's2to4/>=5 'f-b-rep 1400 lum-stress)
(insert4d! 'np 'n2 's2to4/>=5 'f-b-rep 1100 lum-stress)
(insert4d! 'np 'n3 's2to4/>=5 'f-b-rep 650 lum-stress)
(insert4d! 'np 'app 's2to4/>=5 'f-b-rep 1400 lum-stress)
(insert4d! 'np 'stud 's2to4/>=5 'f-b-rep 650 lum-stress)
(insert4d! 'np 'ss 's2to4/>=5 'f-v 70 lum-stress)
```

```
(insert4d! 'np 'n1 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'np 'n2 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'np 'n3 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'np 'app 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'np 'stud 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'np 'ss 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'np 'n1 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'np 'n2 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'np 'n3 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'np 'app 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'np 'stud 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'np 'ss 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'np 'n1 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'np 'n2 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'np 'n3 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'np 'app 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'np 'stud 's2to4/>=5 'e 1100000 lum-stress)

;;Northern White Cedar

(insert4d! 'nwc 'ss 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'nwc 'n1 's2to4/2to4 'f-b-sing 1000 lum-stress)
(insert4d! 'nwc 'n2 's2to4/2to4 'f-b-sing 825 lum-stress)
(insert4d! 'nwc 'n3 's2to4/2to4 'f-b-sing 450 lum-stress)
(insert4d! 'nwc 'app 's2to4/2to4 'f-b-sing 850 lum-stress)
(insert4d! 'nwc 'stud 's2to4/2to4 'f-b-sing 450 lum-stress)
(insert4d! 'nwc 'ss 's2to4/2to4 'f-b-rep 1350 lum-stress)
(insert4d! 'nwc 'n1 's2to4/2to4 'f-b-rep 1150 lum-stress)
(insert4d! 'nwc 'n2 's2to4/2to4 'f-b-rep 950 lum-stress)
(insert4d! 'nwc 'n3 's2to4/2to4 'f-b-rep 525 lum-stress)
(insert4d! 'nwc 'app 's2to4/2to4 'f-b-rep 1000 lum-stress)
(insert4d! 'nwc 'stud 's2to4/2to4 'f-b-rep 525 lum-stress)
(insert4d! 'nwc 'ss 's2to4/2to4 'f-v 65 lum-stress)
(insert4d! 'nwc 'n1 's2to4/2to4 'f-v 65 lum-stress)
(insert4d! 'nwc 'n2 's2to4/2to4 'f-v 65 lum-stress)
(insert4d! 'nwc 'n3 's2to4/2to4 'f-v 65 lum-stress)
(insert4d! 'nwc 'app 's2to4/2to4 'f-v 65 lum-stress)
(insert4d! 'nwc 'stud 's2to4/2to4 'f-v 65 lum-stress)
(insert4d! 'nwc 'ss 's2to4/2to4 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'n1 's2to4/2to4 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'n2 's2to4/2to4 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'n3 's2to4/2to4 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'app 's2to4/2to4 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'stud 's2to4/2to4 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'ss 's2to4/2to4 'e 800000 lum-stress)
(insert4d! 'nwc 'n1 's2to4/2to4 'e 800000 lum-stress)
(insert4d! 'nwc 'n2 's2to4/2to4 'e 700000 lum-stress)
(insert4d! 'nwc 'n3 's2to4/2to4 'e 600000 lum-stress)
(insert4d! 'nwc 'app 's2to4/2to4 'e 800000 lum-stress)
(insert4d! 'nwc 'stud 's2to4/2to4 'e 600000 lum-stress)

(insert4d! 'nwc 'con 's2to4/4 'f-b-sing 600 lum-stress)
(insert4d! 'nwc 'con 's2to4/4 'f-b-rep 675 lum-stress)
(insert4d! 'nwc 'con 's2to4/4 'f-v 65 lum-stress)
(insert4d! 'nwc 'con 's2to4/4 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'con 's2to4/4 'e 600000 lum-stress)
(insert4d! 'nwc 'std 's2to4/4 'f-b-sing 325 lum-stress)
(insert4d! 'nwc 'std 's2to4/4 'f-b-rep 375 lum-stress)
(insert4d! 'nwc 'std 's2to4/4 'f-v 65 lum-stress)
(insert4d! 'nwc 'std 's2to4/4 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'std 's2to4/4 'e 600000 lum-stress)
(insert4d! 'nwc 'util 's2to4/4 'f-b-sing 150 lum-stress)
(insert4d! 'nwc 'util 's2to4/4 'f-b-rep 175 lum-stress)
(insert4d! 'nwc 'util 's2to4/4 'f-v 65 lum-stress)
(insert4d! 'nwc 'util 's2to4/4 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'util 's2to4/4 'e 600000 lum-stress)
```

```
(insert4d! 'nwc 'ss 's2to4/>=5 'f-b-sing 1000 lum-stress)
(insert4d! 'nwc 'n1 's2to4/>=5 'f-b-sing 850 lum-stress)
(insert4d! 'nwc 'n2 's2to4/>=5 'f-b-sing 700 lum-stress)
(insert4d! 'nwc 'n3 's2to4/>=5 'f-b-sing 425 lum-stress)
(insert4d! 'nwc 'app 's2to4/>=5 'f-b-sing 850 lum-stress)
(insert4d! 'nwc 'stud 's2to4/>=5 'f-b-sing 425 lum-stress)
(insert4d! 'nwc 'ss 's2to4/>=5 'f-b-rep 1150 lum-stress)
(insert4d! 'nwc 'n1 's2to4/>=5 'f-b-rep 1000 lum-stress)
(insert4d! 'nwc 'n2 's2to4/>=5 'f-b-rep 825 lum-stress)
(insert4d! 'nwc 'n3 's2to4/>=5 'f-b-rep 475 lum-stress)
(insert4d! 'nwc 'app 's2to4/>=5 'f-b-rep 1000 lum-stress)
(insert4d! 'nwc 'stud 's2to4/>=5 'f-b-rep 475 lum-stress)
(insert4d! 'nwc 'ss 's2to4/>=5 'f-v 65 lum-stress)
(insert4d! 'nwc 'n1 's2to4/>=5 'f-v 65 lum-stress)
(insert4d! 'nwc 'n2 's2to4/>=5 'f-v 65 lum-stress)
(insert4d! 'nwc 'n3 's2to4/>=5 'f-v 65 lum-stress)
(insert4d! 'nwc 'app 's2to4/>=5 'f-v 65 lum-stress)
(insert4d! 'nwc 'stud 's2to4/>=5 'f-v 65 lum-stress)
(insert4d! 'nwc 'ss 's2to4/>=5 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'n1 's2to4/>=5 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'n2 's2to4/>=5 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'n3 's2to4/>=5 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'app 's2to4/>=5 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'stud 's2to4/>=5 'f-c-perp 370 lum-stress)
(insert4d! 'nwc 'ss 's2to4/>=5 'e 800000 lum-stress)
(insert4d! 'nwc 'n1 's2to4/>=5 'e 800000 lum-stress)
(insert4d! 'nwc 'n2 's2to4/>=5 'e 700000 lum-stress)
(insert4d! 'nwc 'n3 's2to4/>=5 'e 600000 lum-stress)
(insert4d! 'nwc 'app 's2to4/>=5 'e 800000 lum-stress)
(insert4d! 'nwc 'stud 's2to4/>=5 'e 600000 lum-stress)

;;Ponderosa Pine-Sugar Pine (Ponderosa Pine-Lodgepole Pine)

(insert4d! 'ppsp 'ss 's2to4/2to4 'f-b-sing 1400 lum-stress)
(insert4d! 'ppsp 'n1 's2to4/2to4 'f-b-sing 1200 lum-stress)
(insert4d! 'ppsp 'n2 's2to4/2to4 'f-b-sing 1000 lum-stress)
(insert4d! 'ppsp 'n3 's2to4/2to4 'f-b-sing 550 lum-stress)
(insert4d! 'ppsp 'app 's2to4/2to4 'f-b-sing 1200 lum-stress)
(insert4d! 'ppsp 'stud 's2to4/2to4 'f-b-sing 550 lum-stress)
(insert4d! 'ppsp 'ss 's2to4/2to4 'f-b-rep 1650 lum-stress)
(insert4d! 'ppsp 'n1 's2to4/2to4 'f-b-rep 1400 lum-stress)
(insert4d! 'ppsp 'n2 's2to4/2to4 'f-b-rep 1150 lum-stress)
(insert4d! 'ppsp 'n3 's2to4/2to4 'f-b-rep 625 lum-stress)
(insert4d! 'ppsp 'app 's2to4/2to4 'f-b-rep 1400 lum-stress)
(insert4d! 'ppsp 'stud 's2to4/2to4 'f-b-rep 625 lum-stress)
(insert4d! 'ppsp 'ss 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ppsp 'n1 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ppsp 'n2 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ppsp 'n3 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ppsp 'app 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ppsp 'stud 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ppsp 'ss 's2to4/2to4 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'n1 's2to4/2to4 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'n2 's2to4/2to4 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'n3 's2to4/2to4 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'app 's2to4/2to4 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'stud 's2to4/2to4 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'ss 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'ppsp 'n1 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'ppsp 'n2 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'ppsp 'n3 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'ppsp 'app 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'ppsp 'stud 's2to4/2to4 'e 1000000 lum-stress)

(insert4d! 'ppsp 'con 's2to4/4 'f-b-sing 725 lum-stress)
```

```
(insert4d! 'ppsp 'con 's2to4/4 'f-b-rep 825 lum-stress)
(insert4d! 'ppsp 'con 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'ppsp 'con 's2to4/4 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'con 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'ppsp 'std 's2to4/4 'f-b-sing 400 lum-stress)
(insert4d! 'ppsp 'std 's2to4/4 'f-b-rep 450 lum-stress)
(insert4d! 'ppsp 'std 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'ppsp 'std 's2to4/4 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'std 's2to4/4 'e 1000000 lum-stress)
(insert4d! 'ppsp 'util 's2to4/4 'f-b-sing 200 lum-stress)
(insert4d! 'ppsp 'util 's2to4/4 'f-b-rep 225 lum-stress)
(insert4d! 'ppsp 'util 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'ppsp 'util 's2to4/4 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'util 's2to4/4 'e 1000000 lum-stress)

(insert4d! 'ppsp 'ss 's2to4/>=5 'f-b-sing 1200 lum-stress)
(insert4d! 'ppsp 'n1 's2to4/>=5 'f-b-sing 1050 lum-stress)
(insert4d! 'ppsp 'n2 's2to4/>=5 'f-b-sing 850 lum-stress)
(insert4d! 'ppsp 'n3 's2to4/>=5 'f-b-sing 500 lum-stress)
(insert4d! 'ppsp 'app 's2to4/>=5 'f-b-sing 1050 lum-stress)
(insert4d! 'ppsp 'stud 's2to4/>=5 'f-b-sing 500 lum-stress)
(insert4d! 'ppsp 'ss 's2to4/>=5 'f-b-rep 1400 lum-stress)
(insert4d! 'ppsp 'n1 's2to4/>=5 'f-b-rep 1200 lum-stress)
(insert4d! 'ppsp 'n2 's2to4/>=5 'f-b-rep 975 lum-stress)
(insert4d! 'ppsp 'n3 's2to4/>=5 'f-b-rep 575 lum-stress)
(insert4d! 'ppsp 'app 's2to4/>=5 'f-b-rep 1200 lum-stress)
(insert4d! 'ppsp 'stud 's2to4/>=5 'f-b-rep 575 lum-stress)
(insert4d! 'ppsp 'ss 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ppsp 'n1 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ppsp 'n2 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ppsp 'n3 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ppsp 'app 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ppsp 'stud 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ppsp 'ss 's2to4/>=5 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'n1 's2to4/>=5 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'n2 's2to4/>=5 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'n3 's2to4/>=5 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'app 's2to4/>=5 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'stud 's2to4/>=5 'f-c-perp 375 lum-stress)
(insert4d! 'ppsp 'ss 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'ppsp 'n1 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'ppsp 'n2 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'ppsp 'n3 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'ppsp 'app 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'ppsp 'stud 's2to4/>=5 'e 1000000 lum-stress)

;;Sitka Spruce

(insert4d! 'ss 'ss 's2to4/2to4 'f-b-sing 1550 lum-stress)
(insert4d! 'ss 'n1 's2to4/2to4 'f-b-sing 1350 lum-stress)
(insert4d! 'ss 'n2 's2to4/2to4 'f-b-sing 1100 lum-stress)
(insert4d! 'ss 'n3 's2to4/2to4 'f-b-sing 600 lum-stress)
(insert4d! 'ss 'app 's2to4/2to4 'f-b-sing 1350 lum-stress)
(insert4d! 'ss 'stud 's2to4/2to4 'f-b-sing 600 lum-stress)
(insert4d! 'ss 'ss 's2to4/2to4 'f-b-rep 1800 lum-stress)
(insert4d! 'ss 'n1 's2to4/2to4 'f-b-rep 1550 lum-stress)
(insert4d! 'ss 'n2 's2to4/2to4 'f-b-rep 1250 lum-stress)
(insert4d! 'ss 'n3 's2to4/2to4 'f-b-rep 700 lum-stress)
(insert4d! 'ss 'app 's2to4/2to4 'f-b-rep 1550 lum-stress)
(insert4d! 'ss 'stud 's2to4/2to4 'f-b-rep 700 lum-stress)
(insert4d! 'ss 'ss 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'ss 'n1 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'ss 'n2 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'ss 'n3 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'ss 'app 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'ss 'stud 's2to4/2to4 'f-v 75 lum-stress)
```

```
(insert4d! 'ss 'ss 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'n1 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'n2 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'n3 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'app 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'stud 's2to4/2to4 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'ss 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'ss 'n1 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'ss 'n2 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'ss 'n3 's2to4/2to4 'e 1200000 lum-stress)
(insert4d! 'ss 'app 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'ss 'stud 's2to4/2to4 'e 1200000 lum-stress)

(insert4d! 'ss 'con 's2to4/4 'f-b-sing 800 lum-stress)
(insert4d! 'ss 'con 's2to4/4 'f-b-rep 925 lum-stress)
(insert4d! 'ss 'con 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'ss 'con 's2to4/4 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'con 's2to4/4 'e 1200000 lum-stress)
(insert4d! 'ss 'std 's2to4/4 'f-b-sing 450 lum-stress)
(insert4d! 'ss 'std 's2to4/4 'f-b-rep 500 lum-stress)
(insert4d! 'ss 'std 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'ss 'std 's2to4/4 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'std 's2to4/4 'e 1200000 lum-stress)
(insert4d! 'ss 'util 's2to4/4 'f-b-sing 200 lum-stress)
(insert4d! 'ss 'util 's2to4/4 'f-b-rep 250 lum-stress)
(insert4d! 'ss 'util 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'ss 'util 's2to4/4 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'util 's2to4/4 'e 1200000 lum-stress)

(insert4d! 'ss 'ss 's2to4/>=5 'f-b-sing 1350 lum-stress)
(insert4d! 'ss 'n1 's2to4/>=5 'f-b-sing 1150 lum-stress)
(insert4d! 'ss 'n2 's2to4/>=5 'f-b-sing 925 lum-stress)
(insert4d! 'ss 'n3 's2to4/>=5 'f-b-sing 525 lum-stress)
(insert4d! 'ss 'app 's2to4/>=5 'f-b-sing 1150 lum-stress)
(insert4d! 'ss 'stud 's2to4/>=5 'f-b-sing 525 lum-stress)
(insert4d! 'ss 'ss 's2to4/>=5 'f-b-rep 1550 lum-stress)
(insert4d! 'ss 'n1 's2to4/>=5 'f-b-rep 1300 lum-stress)
(insert4d! 'ss 'n2 's2to4/>=5 'f-b-rep 1050 lum-stress)
(insert4d! 'ss 'n3 's2to4/>=5 'f-b-rep 600 lum-stress)
(insert4d! 'ss 'app 's2to4/>=5 'f-b-rep 1300 lum-stress)
(insert4d! 'ss 'stud 's2to4/>=5 'f-b-rep 600 lum-stress)
(insert4d! 'ss 'ss 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'ss 'n1 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'ss 'n2 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'ss 'n3 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'ss 'app 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'ss 'stud 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'ss 'ss 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'n1 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'n2 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'n3 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'app 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'stud 's2to4/>=5 'f-c-perp 435 lum-stress)
(insert4d! 'ss 'ss 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'ss 'n1 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'ss 'n2 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'ss 'n3 's2to4/>=5 'e 1200000 lum-stress)
(insert4d! 'ss 'app 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'ss 'stud 's2to4/>=5 'e 1200000 lum-stress)

;;Southern Pine

(insert4d! 'sp 'ss 's2to4/2to4 'f-b-sing 2000 lum-stress)
(insert4d! 'sp 'ss 's2to4/2to4 'f-b-rep 2300 lum-stress)
(insert4d! 'sp 'ss 's2to4/2to4 'f-v 100 lum-stress)
(insert4d! 'sp 'ss 's2to4/2to4 'f-c-perp 565 lum-stress)
```

```
(insert4d! 'sp 'ss 's2to4/2to4 'e 1700000 lum-stress)
(Insert4d! 'sp 'dss 's2to4/2to4 'f-b-sing 2350 lum-stress)
(insert4d! 'sp 'dss 's2to4/2to4 'f-b-rep 2700 lum-stress)
(insert4d! 'sp 'dss 's2to4/2to4 'f-v 100 lum-stress)
(insert4d! 'sp 'dss 's2to4/2to4 'f-c-perp 660 lum-stress)
(insert4d! 'sp 'dss 's2to4/2to4 'e 1800000 lum-stress)
(insert4d! 'sp 'n1 's2to4/2to4 'f-b-sing 1700 lum-stress)
(insert4d! 'sp 'n1 's2to4/2to4 'f-b-rep 1950 lum-stress)
(insert4d! 'sp 'n1 's2to4/2to4 'f-v 100 lum-stress)
(insert4d! 'sp 'n1 's2to4/2to4 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'n1 's2to4/2to4 'e 1700000 lum-stress)
(insert4d! 'sp 'n1d 's2to4/2to4 'f-b-sing 2000 lum-stress)
(insert4d! 'sp 'n1d 's2to4/2to4 'f-b-rep 2300 lum-stress)
(insert4d! 'sp 'n1d 's2to4/2to4 'f-v 100 lum-stress)
(insert4d! 'sp 'n1d 's2to4/2to4 'f-c-perp 660 lum-stress)
(insert4d! 'sp 'n1d 's2to4/2to4 'e 1800000 lum-stress)
(insert4d! 'sp 'n2 's2to4/2to4 'f-b-sing 1400 lum-stress)
(insert4d! 'sp 'n2 's2to4/2to4 'f-b-rep 1650 lum-stress)
(insert4d! 'sp 'n2 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'sp 'n2 's2to4/2to4 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'n2 's2to4/2to4 'e 1600000 lum-stress)
(insert4d! 'sp 'n2d 's2to4/2to4 'f-b-sing 1650 lum-stress)
(insert4d! 'sp 'n2d 's2to4/2to4 'f-b-rep 1900 lum-stress)
(insert4d! 'sp 'n2d 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'sp 'n2d 's2to4/2to4 'f-c-perp 660 lum-stress)
(insert4d! 'sp 'n2d 's2to4/2to4 'e 1600000 lum-stress)
(insert4d! 'sp 'n3 's2to4/2to4 'f-b-sing 775 lum-stress)
(insert4d! 'sp 'n3 's2to4/2to4 'f-b-rep 900 lum-stress)
(insert4d! 'sp 'n3 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'sp 'n3 's2to4/2to4 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'n3 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'sp 'n3d 's2to4/2to4 'f-b-sing 925 lum-stress)
(insert4d! 'sp 'n3d 's2to4/2to4 'f-b-rep 1050 lum-stress)
(insert4d! 'sp 'n3d 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'sp 'n3d 's2to4/2to4 'f-c-perp 660 lum-stress)
(insert4d! 'sp 'n3d 's2to4/2to4 'e 1500000 lum-stress)
(insert4d! 'sp 'stud 's2to4/2to4 'f-b-sing 775 lum-stress)
(insert4d! 'sp 'stud 's2to4/2to4 'f-b-rep 900 lum-stress)
(insert4d! 'sp 'stud 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'sp 'stud 's2to4/2to4 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'stud 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'sp 'con 's2to4/4 'f-b-sing 1000 lum-stress)
(insert4d! 'sp 'con 's2to4/4 'f-b-rep 1150 lum-stress)
(insert4d! 'sp 'con 's2to4/4 'f-v 100 lum-stress)
(insert4d! 'sp 'con 's2to4/4 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'con 's2to4/4 'e 1400000 lum-stress)
(insert4d! 'sp 'std 's2to4/4 'f-b-sing 575 lum-stress)
(insert4d! 'sp 'std 's2to4/4 'f-b-rep 675 lum-stress)
(insert4d! 'sp 'std 's2to4/4 'f-v 90 lum-stress)
(insert4d! 'sp 'std 's2to4/4 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'std 's2to4/4 'e 1400000 lum-stress)
(insert4d! 'sp 'util 's2to4/4 'f-b-sing 275 lum-stress)
(insert4d! 'sp 'util 's2to4/4 'f-b-rep 300 lum-stress)
(insert4d! 'sp 'util 's2to4/4 'f-v 90 lum-stress)
(insert4d! 'sp 'util 's2to4/4 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'util 's2to4/4 'e 1400000 lum-stress)
(insert4d! 'sp 'ss 's2to4/>=5 'f-b-sing 1750 lum-stress)
(insert4d! 'sp 'ss 's2to4/>=5 'f-b-rep 2000 lum-stress)
(insert4d! 'sp 'ss 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'sp 'ss 's2to4/>=5 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'ss 's2to4/>=5 'e 1700000 lum-stress)
(insert4d! 'sp 'dss 's2to4/>=5 'f-b-sing 2050 lum-stress)
(insert4d! 'sp 'dss 's2to4/>=5 'f-b-rep 2350 lum-stress)
(insert4d! 'sp 'dss 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'sp 'dss 's2to4/>=5 'f-c-perp 660 lum-stress)
(insert4d! 'sp 'dss 's2to4/>=5 'e 1800000 lum-stress)
```

```
(insert4d! 'sp 'n1 's2to4/>=5 'f-b-sing 1450 lum-stress)
(insert4d! 'sp 'n1 's2to4/>=5 'f-b-rep 1700 lum-stress)
(insert4d! 'sp 'n1 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'sp 'n1 's2to4/>=5 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'n1 's2to4/>=5 'e 1700000 lum-stress)
(insert4d! 'sp 'n1d 's2to4/>=5 'f-b-sing 1700 lum-stress)
(insert4d! 'sp 'n1d 's2to4/>=5 'f-b-rep 2000 lum-stress)
(insert4d! 'sp 'n1d 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'sp 'n1d 's2to4/>=5 'f-c-perp 660 lum-stress)
(insert4d! 'sp 'n1d 's2to4/>=5 'e 1800000 lum-stress)
(insert4d! 'sp 'n2 's2to4/>=5 'f-b-sing 1200 lum-stress)
(insert4d! 'sp 'n2 's2to4/>=5 'f-b-rep 1400 lum-stress)
(insert4d! 'sp 'n2 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'sp 'n2 's2to4/>=5 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'n2 's2to4/>=5 'e 1600000 lum-stress)
(insert4d! 'sp 'n2d 's2to4/>=5 'f-b-sing 1400 lum-stress)
(insert4d! 'sp 'n2d 's2to4/>=5 'f-b-rep 1650 lum-stress)
(insert4d! 'sp 'n2d 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'sp 'n2d 's2to4/>=5 'f-c-perp 660 lum-stress)
(insert4d! 'sp 'n2d 's2to4/>=5 'e 1600000 lum-stress)
(insert4d! 'sp 'n3 's2to4/>=5 'f-b-sing 700 lum-stress)
(insert4d! 'sp 'n3 's2to4/>=5 'f-b-rep 800 lum-stress)
(insert4d! 'sp 'n3 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'sp 'n3 's2to4/>=5 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'n3 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'sp 'n3d 's2to4/>=5 'f-b-sing 825 lum-stress)
(insert4d! 'sp 'n3d 's2to4/>=5 'f-b-rep 925 lum-stress)
(insert4d! 'sp 'n3d 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'sp 'n3d 's2to4/>=5 'f-c-perp 660 lum-stress)
(insert4d! 'sp 'n3d 's2to4/>=5 'e 1500000 lum-stress)
(insert4d! 'sp 'stud 's2to4/>=5 'f-b-sing 725 lum-stress)
(insert4d! 'sp 'stud 's2to4/>=5 'f-b-rep 850 lum-stress)
(insert4d! 'sp 'stud 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'sp 'stud 's2to4/>=5 'f-c-perp 565 lum-stress)
(insert4d! 'sp 'stud 's2to4/>=5 'e 1400000 lum-stress)

;;Western Cedars

(insert4d! 'wc 'ss 's2to4/2to4 'f-b-sing 1500 lum-stress)
(insert4d! 'wc 'n1 's2to4/2to4 'f-b-sing 1300 lum-stress)
(insert4d! 'wc 'n2 's2to4/2to4 'f-b-sing 1050 lum-stress)
(insert4d! 'wc 'n3 's2to4/2to4 'f-b-sing 600 lum-stress)
(insert4d! 'wc 'app 's2to4/2to4 'f-b-sing 1300 lum-stress)
(insert4d! 'wc 'stud 's2to4/2to4 'f-b-sing 600 lum-stress)
(insert4d! 'wc 'ss 's2to4/2to4 'f-b-rep 1750 lum-stress)
(insert4d! 'wc 'n1 's2to4/2to4 'f-b-rep 1500 lum-stress)
(insert4d! 'wc 'n2 's2to4/2to4 'f-b-rep 1200 lum-stress)
(insert4d! 'wc 'n3 's2to4/2to4 'f-b-rep 675 lum-stress)
(insert4d! 'wc 'app 's2to4/2to4 'f-b-rep 1500 lum-stress)
(insert4d! 'wc 'stud 's2to4/2to4 'f-b-rep 675 lum-stress)
(insert4d! 'wc 'ss 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'wc 'n1 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'wc 'n2 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'wc 'n3 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'wc 'app 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'wc 'stud 's2to4/2to4 'f-v 75 lum-stress)
(insert4d! 'wc 'ss 's2to4/2to4 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'n1 's2to4/2to4 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'n2 's2to4/2to4 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'n3 's2to4/2to4 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'app 's2to4/2to4 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'stud 's2to4/2to4 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'ss 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'wc 'n1 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'wc 'n2 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'wc 'n3 's2to4/2to4 'e 900000 lum-stress)
```

```
(insert4d! 'wc 'app 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'wc 'stud 's2to4/2to4 'e 900000 lum-stress)

(insert4d! 'wc 'con 's2to4/4 'f-b-sing 775 lum-stress)
(insert4d! 'wc 'con 's2to4/4 'f-b-rep 875 lum-stress)
(insert4d! 'wc 'con 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'wc 'con 's2to4/4 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'con 's2to4/4 'e 900000 lum-stress)
(insert4d! 'wc 'std 's2to4/4 'f-b-sing 425 lum-stress)
(insert4d! 'wc 'std 's2to4/4 'f-b-rep 500 lum-stress)
(insert4d! 'wc 'std 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'wc 'std 's2to4/4 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'std 's2to4/4 'e 900000 lum-stress)
(insert4d! 'wc 'util 's2to4/4 'f-b-sing 200 lum-stress)
(insert4d! 'wc 'util 's2to4/4 'f-b-rep 225 lum-stress)
(insert4d! 'wc 'util 's2to4/4 'f-v 75 lum-stress)
(insert4d! 'wc 'util 's2to4/4 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'util 's2to4/4 'e 900000 lum-stress)

(insert4d! 'wc 'ss 's2to4/>=5 'f-b-sing 1300 lum-stress)
(insert4d! 'wc 'n1 's2to4/>=5 'f-b-sing 1100 lum-stress)
(insert4d! 'wc 'n2 's2to4/>=5 'f-b-sing 925 lum-stress)
(insert4d! 'wc 'n3 's2to4/>=5 'f-b-sing 525 lum-stress)
(insert4d! 'wc 'app 's2to4/>=5 'f-b-sing 1100 lum-stress)
(insert4d! 'wc 'stud 's2to4/>=5 'f-b-sing 525 lum-stress)
(insert4d! 'wc 'ss 's2to4/>=5 'f-b-rep 1500 lum-stress)
(insert4d! 'wc 'n1 's2to4/>=5 'f-b-rep 1300 lum-stress)
(insert4d! 'wc 'n2 's2to4/>=5 'f-b-rep 1050 lum-stress)
(insert4d! 'wc 'n3 's2to4/>=5 'f-b-rep 625 lum-stress)
(insert4d! 'wc 'app 's2to4/>=5 'f-b-rep 1300 lum-stress)
(insert4d! 'wc 'stud 's2to4/>=5 'f-b-rep 625 lum-stress)
(insert4d! 'wc 'ss 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'wc 'n1 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'wc 'n2 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'wc 'n3 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'wc 'app 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'wc 'stud 's2to4/>=5 'f-v 75 lum-stress)
(insert4d! 'wc 'ss 's2to4/>=5 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'n1 's2to4/>=5 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'n2 's2to4/>=5 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'n3 's2to4/>=5 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'app 's2to4/>=5 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'stud 's2to4/>=5 'f-c-perp 425 lum-stress)
(insert4d! 'wc 'ss 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'wc 'n1 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'wc 'n2 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'wc 'n3 's2to4/>=5 'e 900000 lum-stress)
(insert4d! 'wc 'app 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'wc 'stud 's2to4/>=5 'e 900000 lum-stress)

;;Western Hemlock

(insert4d! 'wh 'ss 's2to4/2to4 'f-b-sing 1800 lum-stress)
(insert4d! 'wh 'n1 's2to4/2to4 'f-b-sing 1550 lum-stress)
(insert4d! 'wh 'n2 's2to4/2to4 'f-b-sing 1300 lum-stress)
(insert4d! 'wh 'n3 's2to4/2to4 'f-b-sing 700 lum-stress)
(insert4d! 'wh 'app 's2to4/2to4 'f-b-sing 1550 lum-stress)
(insert4d! 'wh 'stud 's2to4/2to4 'f-b-sing 700 lum-stress)
(insert4d! 'wh 'ss 's2to4/2to4 'f-b-rep 2100 lum-stress)
(insert4d! 'wh 'n1 's2to4/2to4 'f-b-rep 1800 lum-stress)
(insert4d! 'wh 'n2 's2to4/2to4 'f-b-rep 1450 lum-stress)
(insert4d! 'wh 'n3 's2to4/2to4 'f-b-rep 800 lum-stress)
(insert4d! 'wh 'app 's2to4/2to4 'f-b-rep 1800 lum-stress)
(insert4d! 'wh 'stud 's2to4/2to4 'f-b-rep 800 lum-stress)
(insert4d! 'wh 'ss 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'wh 'n1 's2to4/2to4 'f-v 90 lum-stress)
```

```
(insert4d! 'wh 'n2 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'wh 'n3 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'wh 'app 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'wh 'stud 's2to4/2to4 'f-v 90 lum-stress)
(insert4d! 'wh 'ss 's2to4/2to4 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'n1 's2to4/2to4 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'n2 's2to4/2to4 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'n3 's2to4/2to4 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'app 's2to4/2to4 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'stud 's2to4/2to4 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'ss 's2to4/2to4 'e 1600000 lum-stress)
(insert4d! 'wh 'n1 's2to4/2to4 'e 1600000 lum-stress)
(insert4d! 'wh 'n2 's2to4/2to4 'e 1400000 lum-stress)
(insert4d! 'wh 'n3 's2to4/2to4 'e 1300000 lum-stress)
(insert4d! 'wh 'app 's2to4/2to4 'e 1600000 lum-stress)
(insert4d! 'wh 'stud 's2to4/2to4 'e 1300000 lum-stress)

(insert4d! 'wh 'con 's2to4/4 'f-b-sing 925 lum-stress)
(insert4d! 'wh 'con 's2to4/4 'f-b-rep 1050 lum-stress)
(insert4d! 'wh 'con 's2to4/4 'f-v 90 lum-stress)
(insert4d! 'wh 'con 's2to4/4 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'con 's2to4/4 'e 1300000 lum-stress)
(insert4d! 'wh 'std 's2to4/4 'f-b-sing 525 lum-stress)
(insert4d! 'wh 'std 's2to4/4 'f-b-rep 600 lum-stress)
(insert4d! 'wh 'std 's2to4/4 'f-v 90 lum-stress)
(insert4d! 'wh 'std 's2to4/4 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'std 's2to4/4 'e 1300000 lum-stress)
(insert4d! 'wh 'util 's2to4/4 'f-b-sing 250 lum-stress)
(insert4d! 'wh 'util 's2to4/4 'f-b-rep 275 lum-stress)
(insert4d! 'wh 'util 's2to4/4 'f-v 90 lum-stress)
(insert4d! 'wh 'util 's2to4/4 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'util 's2to4/4 'e 1300000 lum-stress)

(insert4d! 'wh 'ss 's2to4/>=5 'f-b-sing 1550 lum-stress)
(insert4d! 'wh 'n1 's2to4/>=5 'f-b-sing 1350 lum-stress)
(insert4d! 'wh 'n2 's2to4/>=5 'f-b-sing 1100 lum-stress)
(insert4d! 'wh 'n3 's2to4/>=5 'f-b-sing 650 lum-stress)
(insert4d! 'wh 'app 's2to4/>=5 'f-b-sing 1350 lum-stress)
(insert4d! 'wh 'stud 's2to4/>=5 'f-b-sing 650 lum-stress)
(insert4d! 'wh 'ss 's2to4/>=5 'f-b-rep 1800 lum-stress)
(insert4d! 'wh 'n1 's2to4/>=5 'f-b-rep 1550 lum-stress)
(insert4d! 'wh 'n2 's2to4/>=5 'f-b-rep 1250 lum-stress)
(insert4d! 'wh 'n3 's2to4/>=5 'f-b-rep 750 lum-stress)
(insert4d! 'wh 'app 's2to4/>=5 'f-b-rep 1550 lum-stress)
(insert4d! 'wh 'stud 's2to4/>=5 'f-b-rep 750 lum-stress)
(insert4d! 'wh 'ss 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'wh 'n1 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'wh 'n2 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'wh 'n3 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'wh 'app 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'wh 'stud 's2to4/>=5 'f-v 90 lum-stress)
(insert4d! 'wh 'ss 's2to4/>=5 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'n1 's2to4/>=5 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'n2 's2to4/>=5 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'n3 's2to4/>=5 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'app 's2to4/>=5 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'stud 's2to4/>=5 'f-c-perp 410 lum-stress)
(insert4d! 'wh 'ss 's2to4/>=5 'e 1600000 lum-stress)
(insert4d! 'wh 'n1 's2to4/>=5 'e 1600000 lum-stress)
(insert4d! 'wh 'n2 's2to4/>=5 'e 1400000 lum-stress)
(insert4d! 'wh 'n3 's2to4/>=5 'e 1300000 lum-stress)
(insert4d! 'wh 'app 's2to4/>=5 'e 1600000 lum-stress)
(insert4d! 'wh 'stud 's2to4/>=5 'e 1300000 lum-stress)

;;White Woods (Western Woods)
```

```
(insert4d! 'ww 'ss 's2to4/2to4 'f-b-sing 1350 lum-stress)
(insert4d! 'ww 'n1 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'ww 'n2 's2to4/2to4 'f-b-sing 925 lum-stress)
(insert4d! 'ww 'n3 's2to4/2to4 'f-b-sing 525 lum-stress)
(insert4d! 'ww 'app 's2to4/2to4 'f-b-sing 1150 lum-stress)
(insert4d! 'ww 'stud 's2to4/2to4 'f-b-sing 525 lum-stress)
(insert4d! 'ww 'ss 's2to4/2to4 'f-b-rep 1550 lum-stress)
(insert4d! 'ww 'n1 's2to4/2to4 'f-b-rep 1300 lum-stress)
(insert4d! 'ww 'n2 's2to4/2to4 'f-b-rep 1050 lum-stress)
(insert4d! 'ww 'n3 's2to4/2to4 'f-b-rep 600 lum-stress)
(insert4d! 'ww 'app 's2to4/2to4 'f-b-rep 1300 lum-stress)
(insert4d! 'ww 'stud 's2to4/2to4 'f-b-rep 600 lum-stress)
(insert4d! 'ww 'ss 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ww 'n1 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ww 'n2 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ww 'n3 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ww 'app 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ww 'stud 's2to4/2to4 'f-v 70 lum-stress)
(insert4d! 'ww 'ss 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'n1 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'n2 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'n3 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'app 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'stud 's2to4/2to4 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'ss 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'ww 'n1 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'ww 'n2 's2to4/2to4 'e 1000000 lum-stress)
(insert4d! 'ww 'n3 's2to4/2to4 'e 900000 lum-stress)
(insert4d! 'ww 'app 's2to4/2to4 'e 1100000 lum-stress)
(insert4d! 'ww 'stud 's2to4/2to4 'e 900000 lum-stress)

(insert4d! 'ww 'con 's2to4/4 'f-b-sing 675 lum-stress)
(insert4d! 'ww 'con 's2to4/4 'f-b-rep 775 lum-stress)
(insert4d! 'ww 'con 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'ww 'con 's2to4/4 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'con 's2to4/4 'e 900000 lum-stress)
(insert4d! 'ww 'std 's2to4/4 'f-b-sing 375 lum-stress)
(insert4d! 'ww 'std 's2to4/4 'f-b-rep 425 lum-stress)
(insert4d! 'ww 'std 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'ww 'std 's2to4/4 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'std 's2to4/4 'e 900000 lum-stress)
(insert4d! 'ww 'util 's2to4/4 'f-b-sing 175 lum-stress)
(insert4d! 'ww 'util 's2to4/4 'f-b-rep 200 lum-stress)
(insert4d! 'ww 'util 's2to4/4 'f-v 70 lum-stress)
(insert4d! 'ww 'util 's2to4/4 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'util 's2to4/4 'e 900000 lum-stress)

(insert4d! 'ww 'ss 's2to4/>=5 'f-b-sing 1150 lum-stress)
(insert4d! 'ww 'n1 's2to4/>=5 'f-b-sing 975 lum-stress)
(insert4d! 'ww 'n2 's2to4/>=5 'f-b-sing 800 lum-stress)
(insert4d! 'ww 'n3 's2to4/>=5 'f-b-sing 475 lum-stress)
(insert4d! 'ww 'app 's2to4/>=5 'f-b-sing 975 lum-stress)
(insert4d! 'ww 'stud 's2to4/>=5 'f-b-sing 475 lum-stress)
(insert4d! 'ww 'ss 's2to4/>=5 'f-b-rep 1300 lum-stress)
(insert4d! 'ww 'n1 's2to4/>=5 'f-b-rep 1100 lum-stress)
(insert4d! 'ww ' 'n2 's2to4/>=5 'f-b-rep 925 lum-stress)
(insert4d! 'ww 'n3 's2to4/>=5 'f-b-rep 550 lum-stress)
(insert4d! 'ww 'app 's2to4/>=5 'f-b-rep 1100 lum-stress)
(insert4d! 'ww 'stud 's2to4/>=5 'f-b-rep 550 lum-stress)
(insert4d! 'ww 'ss 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ww 'n1 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ww 'n2 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ww 'n3 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ww 'app 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ww 'stud 's2to4/>=5 'f-v 70 lum-stress)
(insert4d! 'ww 'ss 's2to4/>=5 'f-c-perp 315 lum-stress)
```

```
(insert4d! 'ww 'n1 's2to4/>=5 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'n2 's2to4/>=5 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'n3 's2to4/>=5 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'app 's2to4/>=5 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'stud 's2to4/>=5 'f-c-perp 315 lum-stress)
(insert4d! 'ww 'ss 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'ww 'n1 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'ww 'n2 's2to4/>=5 'e 1000000 lum-stress)
(insert4d! 'ww 'n3 's2to4/>=5 'e 900000 lum-stress)
(insert4d! 'ww 'app 's2to4/>=5 'e 1100000 lum-stress)
(insert4d! 'ww 'stud 's2to4/>=5 'e 900000 lum-stress)
```