DTIC FILE COPY

(2)

USAFA TR 90-4

COMPUTER PROGRAMMING AND GROUP THEORY

MICHAEL K. KEANE, CAPT, USAF
DAVID W. JENSEN, LT COL, USAF

AD-A225 155

MAY 1990
FINAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
ELECTE
AUG 13 1990
S E D

DEAN OF THE FACULTY

UNITED STATES AIR FORCE ACADEMY

COLORADO SPRINGS, CO 80840

90 07 25 106

Technical Review by Capt. Sylvia Ferry
Department of Physics
USAF Academy, Colorado 80840

Technical Review by Capt. Eric R. Bussian
Department of Mathematical Sciences
USAF Academy, Colorado 80840

Editorial Review by Maj. David Harvey
Department of English
USAF Academy, Colorado 80840

This research report is presented as a competent treatment of the subject, worthy of publication. The United States Air Force Academy vouches for the quality of the research, without necessarily endorsing the opinions and conclusions of the authors.

This report has been cleared for open publication and/or public release by the appropriate Office of Information in accordance with AFR 190-1 and AFR 12-30. There is no objection to unlimited distribution of this report to the public at large, or by DTIC to the National Technical Information Service.

This research report has been reviewed and is approved for publication.

Richard Durham

RICHARD DURHAM, Lt Col, USAF
Director of Research, Studies
and Analysis

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release. Unlimited Distribution.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Dept of Mathematical Sciences 6b. OFFICE SYMBOL (If applicable) DFMS	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) US Air Force Academy Colorado Springs, CO 80840-5701	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Computer Programming and Group Theory

12. PERSONAL AUTHOR(S)

Michael K. Keane, Capt, USAF and David W. Jensen, Lt Col, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final Report	FROM Aug 89 TO Jun 90		24

16. SUPPLEMENTARY NOTATION

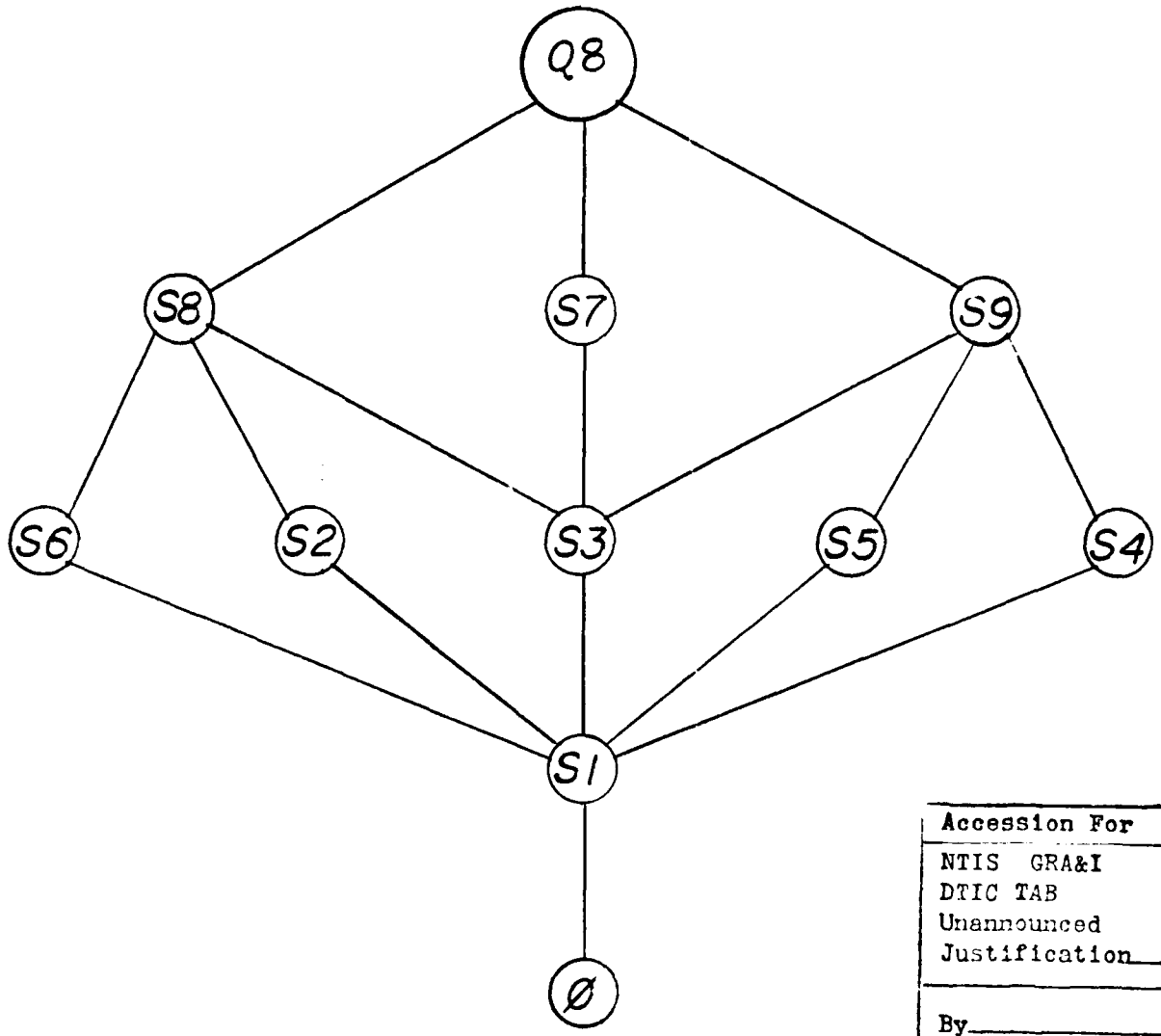
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report investigates the use of the personal computer as an aid in the study of Group Theory. Chapter I discusses the basic mathematical concepts used and how they are translated into a programming language. chapter II discribes testing results using the forty non-isomorphic groups of order three to sixteen. Chapter III ~~then~~ builds upon the previous chapter, presenting several examples an instructor could use to focus and clarify the ideas presented in the class and text. Keywords: Mathematical [illegible] Mathematics. (CP)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Michael K. Keane, Capt, USAF	22b. TELEPHONE (Include Area Code) (719)472-3419 22c. OFFICE SYMBOL DFMS

DD Form 1473, JUN 86 Previous editions are obsolete. 259 SECURITY CLASSIFICATION OF THIS PAGE
Unclassified



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

"Computer Programming and Group Theory"

MICHAEL K. KEANE
DAVID W. JENSEN

May 1990

Department of Mathematical Sciences
U.S. Air Force Academy, Co 80840

QUALITY INSPECTED 4

On the title page is the lattice diagram for the quaternion group of order sixteen.

TABLE OF CONTENTS

Introduction 1

Chapter I: Program Description and Capabilities 2

Chapter II: Program Testing 20

Chapter III: Suggested Assignments using ABSALGS 32

Bibliography 35

Appendix A: The 40 Non-isomorphic Groups of Order 3<= n <=40 36

Appendix B: Listing of Programs 78

Appendix C: ABSALGS 80

~~Appendix D: Working Disk . . . . . 152~~

Accession For

NTIS GRA&I ☒
DTIC TAB ☐
Unannounced ☐
Justification

By
Distribution/

Availability Codes

Dist	Avail and/or Special
A-1	

Introduction

Undergraduate students usually develop their appreciation for abstract mathematics in Linear or Modern Algebra classes. Linear Algebra introduces the student to formal proofs. Then, Modern Algebra uses formal proofs to gain insight into group, ring, and field theories.

Group theory is concerned with both finite and infinite group structures. The new student to algebra often does not have the necessary concrete examples at his fingertips to drive home the ideas presented. This may cause students a great deal of frustration and affect their confidence. The program and data base described in this report are designed to alleviate this frustration. The program makes available to the instructor of undergraduate students the forty non-isomorphic groups of order three to sixteen, and the capability to generate any group up to order one hundred.

This report describes the mathematics behind the program and how the program functions. Chapter I discusses the basic mathematical concepts used and how they are translated into a programming language. Chapter II describes testing results using the forty non-isomorphic groups of order three to sixteen. Chapter III then builds upon the previous chapters, presenting several examples an instructor could use to focus and clarify the ideas presented in the class and text.

Chapter I

To write a program that generates the elements and subgroups of a group, a mathematical scheme to generate the elements must be picked. Several such schemes exist, for example: power notation, matrix representation, prime power decomposition, integers under addition modulo some n, and permutations. The authors chose the permutation scheme because of its general applicability. We know from Cayley's theorem that every group is isomorphic to a group of permutations.

Next the authors had to decide what format the permutations would be represented in - either standard notation

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 2 & 4 & 3 & 1 & 6 \end{pmatrix}$$

or Cyclic notation (1 5)(2)(3 4)(6). Since every permutation in standard notation may be written in cyclic notation as above, and cyclic notation requires less computer storage space, cyclic notation was picked.

Cyclic notation by its very nature is a wrap-around notation. This causes some problems since a permutation written in cyclic notation is not unique. For example, (1 2 3 4), (2 3 4 1), and (3 4 1 2) all represent the same permutation. This problem is compounded when a permutation requires more than one cycle to represent it. For example, the permutation (1 2 3 4)(5 6 7 8) has sixteen possible different representations in cycle format. Also, an element of a cycle may be repeated in

following cycles, or singleton cycles may be present. All of these problems are easily solved by writing the cycles of a permutation in a special way. This special format is called the canonical form of a permutation. It is unique, and its existence stems from the following important facts:

1. Since singleton cycles neither add nor detract from a permutation represented in cyclic notation, they may be dropped.

2. Every permutation of a finite set is a product of disjoint cycles [1].

In Knuth [2] an algorithm was found to generate the canonical form of a permutation. A collection of cycles that represents a permutation is first written as a product of disjoint cycles. Next, each cycle is ordered by finding the smallest element and putting it first. Then all the cycles are ordered according to their first elements, highest to lowest. Any singleton cycles are dropped during this process. An example is the following, assuming a product of disjoint cycles:

permutation: (4 1 2 3)(8 6 7 5)(9)
first cycle becomes (1 2 3 4)
second cycle becomes (5 8 6 7)
third cycle is dropped

permutation in canonical form: (5 8 6 7)(1 2 3 4).

The next problem to arise prior to programming was the programming language. The authors wanted a portable, executable code, i.e., portable without recompiling for different personal computers (pcs) using different versions of MS-DOS. We chose the "C" language. The particular compiler is Turbo C [3]. The executable code works on various pcs made by Zenith and IBM.

These machines use MS-DOS version 2.0 through 3.3 with different designs of hardware, i.e., processors 8086, 80286, and 80386.

Implementation of the above decisions into code could now proceed. For speed and efficiency, the code and corresponding data space was left in memory until the program completely generated the elements and subgroups of a group. At that time, the data was written to disk under filenames generated by the computer using information that the user supplied. Also, the user was given the option to print the data. These design decisions drove the physical limitations of the program.

An early concern was how best to store the permutation data. We decided that the user would only see nonnegative numbers within the permutations, i.e., (1 2 3 4). We also decided to take advantage of C's extensive character manipulation library. Therefore, all numerical values entered or generated are converted to characters. This saves space, since numerical values require two (2) bytes, and characters require only one (1) byte.

The above design decisions limited the number of numerical values within a permutation to 256, that is, the numbers 0 through 255. However, zero (0) could not be used since it is the terminating character of the character string in C. Also, the numerical values 128 through 255, when converted to characters, have the first bit of the character byte turned on. This is significant because the first bit is propagated through the second byte when converting back to numerical values, yielding

negative numbers for printout. Thus, the numerical values 1 through 127 are the only values that qualify.

Next, the authors decided on the number of elements and subgroups that the system could generate and place in storage. If all the 127 numbers are used, the largest permutation possible takes 190 characters to express. Consider for example, the problem of storing the elements of the dihedral groups D(127). Since the order of D(127) is 254, we needed a storage matrix of size 48k (254 x 190), where k stands for kilobytes.. Considering other matrices we also needed to use and the limits on data storage for any module of a C program, we decided to limit the number of elements to 200. The final size of the element matrix is 203 by 155, which uses 32k of storage. The extra three rows are used for system space.

Each row of the subgroup matrix must contain at least the numeric names of all the elements possible in the group. This value is 203. The number of possible subgroups capable of being produced by the system is limited by available space in the common storage area. The common storage area allows only 64k. Since other variables also need to be kepted in common storage, we gave the subgroup matrix dimensions of 273 by 203, requiring 56k.

Except for some other minor considerations, the system was ready to be written. The authors decided that the user should have a definitive reason to do everything. This means that the system would not only be menu driven, but that every action would

require at least two key strokes; that is, an option keystroke and pressing ENTER to confirm the option. This allows the user to correct a mistake before the system starts working.

To start the system, the user loads the program in the current drive (drive A or B), and then types in **ABSALGS.** The first menu that the user sees when running the system is the Permutation Program Main Menu noted below.

PERMUTATION PROGRAM MAIN MENU

Option 1 - Group Generation

Option 2 - Print a Generated Group

Option 3 - Exit Program

Enter Option and Press Enter:

This menu allows the user to choose group generation, a print function, or to exit. Let us say the user chooses to generate a group. The menu then displayed is the Group Generation Menu.

GROUP GENERATION MENU

Option 1 for DIHEDRAL GROUPS

Option 2 for CYCLIC GROUPS

Option 3 for ABELIAN GROUPS

Option 4 for UNKNOWN GROUPS

Option 5 for EXIT PROGRAM

Enter Option and press ENTER:

If the user chooses options 1 or 2 the following directions are respectively displayed.

DIRECTIONS:

Enter the number of the Dihedral Group

after the 'D' and press ENTER

To exit just press ENTER

D

DIRECTIONS:

Enter the number of the Cyclic Group

after the 'C' and press ENTER

To exit just press ENTER

C

Enter the number of the group you wish generated, press "ENTER" and the system will display the generating elements. When the system completes the group it will write it to the same drive that was used to start the system. The system will then give the user the option to print the group either to screen or printer. If this option is chosen, the system will perform the requested task, and then ask the user to press "ENTER" to continue. This action will return the user to the directions of either option 1 or 2 respectively. The directions may be exited by pressing "ENTER" without entering a numerical value. This action returns the user to the Group Generation Menu.

Options 3 and 4 of the Group Generation Menu are also easy

to use. Suppose you choose option 4. The following directions are presented:

DIRECTIONS:

> When prompted with a '*', enter a permutation
> in cycle notation. Use '()' as delimiters.
> Enter only integer values (i) in the range
> 0 < i < 128. Place a space between each i,
> For Example: (1 34 5) (23 127 2).
>
> A PERMUTATION MAY ONLY BE 80 CHARACTERS LONG.
> NOTE: A GROUP MAY HAVE A TOTAL OF 128 ELEMENTS.
>
> Press ENTER when the permutation is complete.
> To end entry press ENTER with a null input.

"To end entry press ENTER with a null input.", allows you to either exit the option before entering any permutations, or start the group generation process after one or more permutations are entered.

Suppose you enter the following permutation:

(1 3 6 7)(2 3 4)(1 5 4)(6 1 4 5)(2 7 6 1 5).

Using this permutation, the following demonstrates the systems internal formatting capabilities. Two lines describe the entered permutation. The first line is in ASCII notation. The second line is the equivalent hexadecimal notation. The above permutation converts in the following way:

```
(  1  3  6  7  )  (  2  3  4  )  (  1  5  4  )
28 31 33 36 37 29 28 32 33 34 29 28 31 35 34 29

(  6  1  4  5  )  (  2  7  6  1  5  )  0
28 36 31 34 35 29 28 32 37 36 31 35 29 00
```

The terminating zero is added by the standard C programming library input function "get string" (gets( )). Now the internal formatting begins. First the right parentheses are dropped.

```
(  1  3  6  7  (  2  3  4  (  1  5  4
28 31 33 36 37 28 32 33 34 28 31 35 34

(  6  1  4  5  (  2  7  6  1  5  0
28 36 31 34 35 28 32 37 36 31 35 00
```

Second, all left parentheses are converted to 255.

```
255 1  3  6  7  255 2  3  4  255 1  5  4
ff  31 33 36 37 ff  32 33 34 ff  31 35 34

255 6  1  4  5  255 2  7  6  1  5  0
ff  36 31 34 35 ff  32 37 36 31 35 00
```

Next, the value of each integer is converted from a string character to a numeric value, insuring each value is between 0 and 128 exclusive. It is then stored in a single character byte, which has a new hexadecimal notation.

```
255 1  3  6  7  255 2  3  4  255 1  5  4
ff  01 03 06 07 ff  02 03 04 ff  01 05 04

255 6  1  4  5  255 2  7  6  1  5  0
ff  06 01 04 05 ff  02 07 06 01 05 00
```

The above new permutation representation is the format that is passed from the Internal Formatter to the Multiplier. The Multiplier reduces the permutation to a product of disjoint cycles. The algorithm for the Multiplier is in Knuth [2].

To illustrate the steps of the Multiplier consider the permutation used above in the demonstration of the Internal Formatter function. For simplification, we select a starting point that demonstrates all the checks of the Multiplier. Assume

the following conditions: a "U" shows which elements in the collection of cycles have already been traced through the cycles and recorded in a new permutation called the answer permutation. Note: the 255 in the answer permutation is a left parenthesis, not an element that was traced through the cycles and recorded.

```
     U                        U      U     U
 255 1  3  6  7  255 2  3  4  255 1  5  4
ff  01 03 06 07 ff  02 03 04 ff  01 05 04

        U  U                     U
 255 6  1  4  5  255  2  7  6  1  5  0
 ff  06 01 04 05 ff   02 07 06 01 05 00
```

Answer permutation is

```
 255 1  4
 ff  01 04
```

Holding a pointer at the first four in the permutation being multiplied and going from left to right, one can find the element that the four is mapped onto. Scanning the permutation, we find that the first element after the four is a left parenthesis, or 255. Thus in this cycle, the four maps onto the first element in the cycle, or two. Starting with the two, scan the permutation for the next occurrence of two. It occurs in the last cycle. In the last cycle, two is mapped onto seven. Starting with the seven, scan the permutation for the next occurrence of the seven without going past the zero at the end of the permutation. Since another seven does not occur before the end of the permutation, seven is where the four maps to in the answer permutation. Mark

all occurrences of seven with a "U". What we have now is:

```
     U         U             U       U      U
255  1   3   6   7   255  2   3   4   255  1   5   4
ff   01  03  06  07  ff   02  03  04  ff   01  05  04

         U   U              U      U
255  6   1   4   5   255   2   7   6   1   5   0
ff   06  01  04  05  ff    02  07  06  01  05  00
```

and the answer permutation is now:

```
255  1   4   7
ff   01  04  07
```

Holding a pointer at the first seven in the permutation being multiplied and going from left to right, one can find the element the seven is mapped onto. Scanning the permutation, we find that the first element after the seven is a left parenthesis, or 255. In this cycle, the seven maps onto the first element in the cycle, or one. Starting with the one, we scan the permutation for the next occurrence of one. It occurs in the third cycle. In the third cycle, one is mapped onto five. Starting with the five, scan the permutation for the next occurrence of five. It occurs in the fourth cycle. In the fourth cycle, the element following the five is a left parenthesis, or 255, therefore the five maps onto the first element in the fourth cycle, or six. Starting with the six, we scan the permutation for the next occurrence of six. It occurs in the fifth cycle. In the fifth cycle, the six maps onto one. Starting with the one, scan the permutation without going past the zero at the end of the permutation. Since another one does not occur before the end of the permutation, one is where the seven will map onto in the answer permutation. But one has

already been used, therefore a cycle is closed in the answer permutation by putting a 255 (left parenthesis) as a cycle delimiter. We now have:

```
     U        U            U      U     U
255  1  3  6  7  255  2  3  4  255  1  5  4
ff   01 03 06 07 ff   02 03 04 ff   01 05 04

        U  U             U     U
255  6  1  4  5  255  2  7  6  1  5  0
ff   06 01 04 05 ff   02 07 06 01 05 00
```

and the answer permutation is:

```
255  1  4  7  255
ff   01 04 07 ff
```

Going from left to right, we find the first element not used, that is, the first element that does not have a "U" above it, and start the process all over again. When all elements have been used, the Multiplier terminates by putting a terminating zero (0) in place of the trailing 255. The answer permutation is:

```
255  1  4  7  255  3  5  2  255  6  0
ff   01 04 07 ff   03 05 02 ff   06 00
```

The input permutation passed to the Multiplier had five cycles and twenty-five characters. After completion of the Multiplier, the input permutation was reduced to three cycles and eleven characters.

Notice that the last cycle is a singleton cycle. A subroutine called Singleton Remover was developed to remove these unnecessary and space wasting singletons. Passing through Singleton Remover, our permutation is reduced to two cycles and

nine characters as follows:

```
255 1  4  7  255 3  5  2  0
ff  01 04 07 ff  03 05 02 00
```

The above is our original permutation represented in its most compact form. However, it is not unique. For instance the following two permutations also represent the same permutation:

```
255 7  1  4  255 5  2  3  0
ff  07 01 04 ff  05 02 03 00

255 4  7  1  255 2  3  5  0
ff  04 07 01 ff  02 03 05 00
```

The canonical form discussed earlier must now be used. A separate program was developed to perform this action. The final form of our original permutation is

```
255 2  3  5  255 1  4  7  0
ff  02 03 05 ff  01 04 07 00
```

The final action in the internal formatting process is to insure that the permutation, now in canonical form, does not exist on the element table. If it does exist, an appropriate message is returned to the user. If it does not exist, then the system echoes this permutation to the user and configures itself to accept another permutation. It is interesting to note that the entire internal formatting process for the largest possible permutation entered requires less than two seconds.

The system allows the user to enter up to seven different permutations as group generating elements. Once the user decides on the number of permutations to input and does so, he then presses "ENTER" with a null input. The system will then generate

the group, that is, all the elements and subgroups that are possible from the entered permutations.

When the system completes the group generation process it will prompt the user for an identifying name. This name may be up to seven characters long. It will be concatenated with E.DAT for the elements and S.DAT for the subgroups. The files are created and the data recorded.

With one small exception, the system then follows the same procedure that was explained for options 1 and 2 of the Group Generation Menu. When the system requests you to press "ENTER" to continue, the system returns to the Group Generation Menu.

Below is the group generated by

(1 3 6 7)(2 3 4)(1 5 4)(6 1 4 5)(2 7 6 1 5)

The elements are the following:

e0 is the identity element

e1 = (2 3 5) (1 4 7)

e2 = (2 5 3) (1 7 4)

The subgroups are the following:

S0 is the null subgroup

S1 = (e0 e1 e2)

The above was a very quick walk-through of the group generation options of the system. We will now investigate the print options.

The user starts the system and chooses the Print Menu

option. The system displays the Print Menu to the user.

PRINT MENU

Option 1 for DIHEDRAL GROUPS

Option 2 for CYCLIC GROUPS

Option 3 for ABELIAN GROUPS

Option 4 for UNKNOWN GROUPS

Option 5 EXIT PROGRAM

Enter Option and press ENTER:

If the user chooses either option 1 or 2 the following instructions are displayed respectively:

DIRECTIONS:

Enter the number of the Dihedral Group

after the 'D' and press ENTER

To exit just press ENTER

D

DIRECTIONS:

Enter the number of the Cyclic Group

after the 'C' and press ENTER

To exit just press ENTER

C

The user must enter the number of a group for which data exists. This may be one of twenty groups that are dihedral or cyclic of the forty non-isomorphic groups in the data base. It may also be the number of a dihedral or cyclic group that the user generated under the Group Generation option of the Main Menu.

If the data does not exist the system will immediately terminate and put the user at the DOS prompt. Once the system reads the data off the disk the following self-explanatory instructions are displayed:

SELECT OUTPUT METHOD

p for PRINTER

s for SCREEN

Enter choice and press ENTER:

Suppose the user was in option 1 and wished to print to the screen Dihedral Group 3, the following is what the user would see:

The elements are the following:

```
e0 is the identity element
e1 = ( 1 2 3 )
e2 = ( 2 3 )
e3 = ( 1 3 2 )
e4 = ( 1 3 )
e5 = ( 1 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = ( e0 e2 )
S2 = ( e0 e4 )
S3 = ( e0 e5 )
S4 = ( e0 e1 e3 )
S5 = ( e0 e1 e2 e3 e4 e5 )
```

Press ENTER to continue:

Once the data is written either to the screen or the printer, the system will request the user to press ENTER to continue. This action will display the directions for option 1, if the group last printed was a dihedral group, or 2, if the

group last printed was a cyclic group. From here the user may print another dihedral or cyclic group or exit to the Print Menu.

Suppose our user exits to the Print Menu and then chooses option 3 (abelian option) or 4 (unknown option). The following directions will be presented respectively:

DIRECTIONS:

Enter the first seven characters of
the Abelian group file name after
the '*' and press ENTER.
To exit just press ENTER.
*

DIRECTIONS:

Enter the first seven characters of
the unknown group file name after
the '*' and press ENTER.
To exit just pres ENTER.
*

At this juncture the user must know the filename. Under option 4, unknown group, it may be any name that exists, either generated by the user using the Group Generation option of the Main Menu or one of the forty groups that are in the system's data base. For option 3, abelian group, only the filename of an abelian group that exists should be entered. It may be either user generated, or one of the forty groups in the system data base that is abelian and non-cyclic. Up to seven characters may be typed in. The E.DAT or S.DAT that is attached to all data files when they are created is not entered, they are generated by

the computer. From this point the same directions appear as in option 1 or 2 with one small difference. When the user is asked to press "ENTER" to continue, the system returns to the Print Menu.

For a demonstration, suppose the user was in the Unknown Group Option and wished to print the group structure known as the Klein 4-group. The filenames for this group are KLEIN4E.DAT and KLEIN4S.DAT. The user would enter klein4 in lowercase letters. By following the simple directions, the following would be printed to screen:

The elements are the following:

```
e0 is the identity element
e1 = ( 3 4 ) ( 1 2 )
e2 = ( 2 3 ) ( 1 4 )
e3 = ( 2 4 ) ( 1 3 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = ( e0 e1 )
S2 = ( e0 e2 )
S3 = ( e0 e3 )
S4 = ( e0 e1 e2 e3 )
```

Press ENTER to continue:

When the user presses "ENTER" to continue, the system returns him to option 4 (unknown option).

This chapter has highlighted the major functions and capabilities of the abstract algebra programming system ABSALGS. We have included two appendices to assist those readers wishing to delve deeper into the programming itself. Appendix B simply lists the names of all the programs within the system with a

short description of each. Appendix C contains the actual programs.

Chapter II

We tested the algebra programming system ABSALGS by developing a data base of the forty non-isomorphic groups of order three to sixteen, hereafter referred to as the System Data Base. In what follows, each of the four generating options are addressed in a similar fashion. First, the type of group or groups are formally defined. Second, if possible, a practical interpretation of the definition is given with an example. Third, a table or explanation of generating times is presented. Finally, we offer a table or explanation of the number of subgroups of each group. The reader should also note that appendix A contains all the data in the System Data Base, ordered on the size of the group within each type of group.

To test the system fully, every option of the group generation process needs to be exercised. Scanning the group generation menu there are four selections to test, that is, dihedral, cyclic, abelian, and unknown.

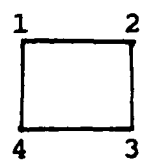
The Dihedral Option:

The dihedral groups are defined formally in the following manner: For an integer n, an element of the natural numbers, such that two is less than or equal to n, let D(n) equal the set G of elements X and Y, such that, X^^n equals 1, Y^^2 equals 1, and (XY)^^2 = 1, that is,

$$D(n) = G\{ X, Y/ X^{\wedge\wedge}n = 1, Y^{\wedge\wedge}2 = 1, (XY)^{\wedge\wedge}2 = 1\}.$$

Although this is a good abstract definition which, with enough multiplications, will generate a group table, a more practical definition of dihedral groups is possible. Indeed, the nth dihedral group D(n) may be thought of as the group of symmetries of the regular n-gon. For example, the dihedral group D(4) may be interpreted as the symmetries of the square. Here the symmetries refers to those rotations, mirror images in perpendicular bisectors of sides, and diagonal flips that bring the figure back onto itself.

When generating the permutations for D(n), it is convenient to think of a n-gon with a natural number associated with each corner, that is, the point of intersection of two sides. The figure below will be an aid in the comprehension of permutation generation for D(4).



Assume that the square in the above figure is in its identity position; then one clockwise rotation would produce (1 2 3 4) as a permutation in cycle notation. That is to say, one is mapped onto two's position, two is mapped onto three's position, three is mapped onto four's position, and four is mapped onto one's position. One more clockwise rotation would produce (1 3)(2 4) as a permutation. The next two clockwise rotations would produce (1 4 3 2) and (1)(2)(3)(4) respectively.

Again, if it is assumed that the above figure is in its identity position, then one mirror image in the perpendicular bisector of sides 1-2 and 3-4 would produce the permutation (1 2)(3 4). One more mirror image in the perpendicular bisector of the sides 1-2 and 3-4 would return the square to the identity position. Similarly, a diagonal flip using the diagonal that joins 2 to 4 would produce the permutation (1 3). One more diagonal flip using the same diagonal would result in the square returning to its identity position. Continuing in this manner, a total of eight permutations may be generated.

This method illustrates the way a human might attempt to find all the elements of a dihedral group. For a program on a computer to generate the elements a more general method is needed. The following generates the dihedral group D(n) on a computer using cyclic notation:

(1 2 3 4 5 . . . n) and

(1 n)(2 (n - 1))(3 (n - 2)) . . . (m (m + 1))

for n even, and by the permutations

(1 2 3 4 5 . . . n) and

(2 n)(3 (n - 1))(4 (n - 2)) . . . (m (m + 1))

for n odd.

The dihedral section of the program will only work for n greater than two. It is interesting to note that a dihedral group has twice as many elements as its associated value of n, for instance D(8) has sixteen elements. Thus it is easy to see that the dihedral groups in System Data Base are D(3), D(4),

D(5), D(6), D(7), and D(8). Below is a chart indicating the amount of computer time in seconds it took to generate each dihedral group.

DIHEDRAL GROUP	TIME
D(3)	1.76
D(4)	3.92
D(5)	5.99
D(6)	22.92
D(7)	20.27
D(8)	63.04

The following is a table of the number of subgroups by order for each of the dihedral groups. It does not include the group itself.

DIHEDRAL GROUP	SUBGROUP ORDER								TOTAL
	1	2	3	4	5	6	7	8	
D(3)	1	3	1						5
D(4)	1	5		3					8
D(5)	1	5			1				7
D(6)	1	7	1	3		3			15
D(7)	1	7					1		9
D(8)	1	9		5				3	18

This concludes our discussion of the dihedral groups. We now turn our attention to the cyclic groups.

The Cyclic Option:

The cyclic groups are formally defined by: For an integer n, an element of the natural numbers, such that 2 is less than or equal to n, let C(n) equal the set G of elements X, such that X^^n equals 1, that is,

$$C(n) = G\{X/\ X^{\wedge\wedge}n = 1\}.$$

Again this is a good abstract definition. However, there is a more practical interpretation. You may already have noticed that the above definition is the first part of the definition for dihedral groups. We may think of a cyclic group as nothing more than the rotations of the regular n-gon. For example, C(4) has elements (1 2 3 4), (1 3)(2 4), (1 4 3 2), and (1)(2)(3)(4). These are the first four elements of D(4).

The above method represents one way a person might generate a cyclic group. However, for a computer, all we need is one generating element in permutation form. The cyclic group C(n) may be generated on a computer with the permutation

(1 2 3 4 . . . . n).

As you may have guessed, a cyclic group has as many elements as its associated n. For example, C(5) has five elements. For the purpose of the program, n must be greater than two to use the cyclic option of the menu. Fourteen groups in the System Data Base are cyclic groups. C(16) took approximately five seconds to produce, and is the largest cyclic group generated for our test.

C(3) through C(15) each required less than five seconds to produce.

The cyclic groups have one subgroup for each integer value that divides the order of the group. This means that C(16) has five subgroups: the identity, the group itself, the subgroup of order two, the subgroup of order four, and the subgroup of order eight. Similarly, the subgroups of the other cyclic groups from order three to sixteen are easily produced.

We next turn our attention to the abelian groups, which are closely related to the cyclic groups.

The Abelian Option:

By the Fundamental Theorem of Finitely Generated Abelian Groups, every finitely generated abelian group G is isomorphic to a direct product of cyclic groups of the form

1. Z[p(1)^^r(1)] x Z[p(2)^^r(2)] x ... x Z[p(n)^^r(n)] x Z x...x Z

where the p(i) are primes, not necessarily distinct, and also of the form

2. Z[m(1)] x Z[m(2)] x ... x Z[m(n)] x Z x ... x Z

where m(i) divides m(i+1) [1].

There are eight abelian groups in the above formats that are part of the System Data Base. The following table gives the

traditional name, the name given as a filename in the System Data Base, and the time in seconds it took to generate each group.

TRADITIONAL	FILENAME	TIME
Z(2)xZ(2)xZ(2)	a2a2a2	12.74
Z(2)xZ(4)	a4xa2	3.10
Z(3)xZ(3)	a3xa3	2.97
Z(2)xZ(6)	a6xa2	14.98
Z(2)xZ(8)	a2x8	31.45
Z(4)xZ(4)	a4x4	34.85
Z(2)xZ(2)xZ(4)	a2x2x4	150.28
Z(2)xZ(2)xZ(2)xZ(2)	a22x22	609.58

NOTE: For the rest of this paper the filenames will be used to reference the different groups.

The following table gives us an idea how the subgroups break down for the eight abelian groups. It does not include the group itself.

GROUP NAME	SUBGROUP ORDER						TOTAL
	1	2	3	4	6	8	
a2a2a2	1	7		7			15
a4xa2	1	3		3			7
a3xa3	1		4				5
a6xa2	1	3	1	1	3		9
a2x8	1	3		3		3	10
a4x4	1	3		7		3	14
a2x2x4	1	7		11		7	26
a22x22	1	15		35		15	66

The Unknown Option:

For this option, the filenames used help identify the group structure. In some cases, the group structure is quite well-known, even to the casual algebra student. One easy example being the Klein 4-group. In other cases, the group structure and generating elements are harder to identify. The groups will be listed in group order from lowest to highest. Where there are several groups of the same order, the authors randomly chose their listing.

Besides C(4), the Klein 4-group is the only other order four group known to exist. It is isomorphic to the dihedral group D(2), but cannot be formed in the same manner as other dihedral groups.

The Quaternion group of order eight forms a skew field under addition and multiplication. The group (q4) is defined by:
q4 = { X, Y / X^^4 = 1, Y^^2 = X^^2, YX = X^^3Y }.

The subgroup of the symmetric group on four letters is called the alternating group (alt4) on four letters. The number of elements is four factorial divided by two.

The group s3xz2 is the group formed by a direct product of the symmetric group on three letters, which is isomorphic to the dihedral group D(3), and Z(2), the integers under addition modulo two.

The semidihedral group sd2z8z2 which is isomorphic to one semidirect product of Z(8) X Z(2), is defined in the following manner:
sd2z8z2 = G{ X, Y / X^^8 = 1, Y^^2 = 1, and YX = (X^^3)Y }.

The group sd1z8z2 is isomorphic to a different semidirect product of Z(8) X Z(2). It is defined in the following manner:
sd1z8z2 = G{ X, Y / X^^8 = 1, Y^^2 = 1, and YX = (X^^5)Y }.

The generalized quaternions of order sixteen (q8) are defined by:
q8 = G{ X, Y/ X^^8 = 1, Y^^2 = X^^4, YX = (X^^7)Y }.

The direct product of the quaternions of order eight and the integers under addition modulo two, that is, Q(4) X Z(2), is a group of order sixteen. It is named q4xz2.

The direct product of the dihedral group D(4) and the integers under addition modulo two, that is, D(4) X Z(2), is named d4xz2.

The group q4xz4 is isomorphic to a subgroup of the direct product of the quaternions of order eight, and the integers under addition modulo four, that is, Q(4) X Z(4). It is defined by:

q4xz4 = { X, Y/ X^^4 = 1, Y^^4 = 1, YX = (X^^3)Y }.

The group wpd4z4 is isomorphic to a subgroup of the wreath product of the dihedral group D(4) and the integers under addition modulo four. It is defined in the following manner:

wpd4z4 = { X, Y, Z/ X^^4 = 1, Y^^2 = 1, Z^^2 = 1,
ZYZ(X^^2)Y = 1, YXYX^^-1 = 1, and ZXZX^^-1 = 1}.

The group dpd4d4 is isomorphic to a subgroup of the direct product of the dihedral group D(4) and D(4). It is defined in the following manner:

dpd4d4 = { X, Y/ X^^4 = 1, Y^^4 = 1, (XY)^^2 = 1,
((X^^-1)Y)^^2 = 1 }.

The table below gives the run times in seconds for each of the unknown groups.

GROUP NAME	TIME
klein4	.81
q4	4.04
alt4	9.34
s3xz2	26.78
sd2z8z2	122.23
sd1z8z2	69.59
q8	88.13
q4xz2	188.52
d4xz2	187.59
q4xz4	111.98
wpd4z4	525.62
dpd4d4	163.63

The following table gives the subgroup structure of the unknown groups. It does not include the group itself.

GROUP NAME	SUBGROUP ORDER						TOTAL
	1	2	3	4	6	8	
klein4	1	3					4
q4	1	1		3			5
alt4	1	3	4	1			9
s3xz2	1	7	1	3	3		15
sd2z8z2	1	5		5		3	14
sd1z8z2	1	3	3			3	10
q8	1	1		5		3	10
q4xz2	1	3		7		7	18
d4xz2	1	11		15		7	34
q4xz4	1	3		7		3	14
wpd4z4	1	7		7		7	22
dpd4d4	1	7		9		3	20

Generating the last twelve groups in our System Data Base in the unknown option completed the formal testing of the system. Other tests were performed, such as: generating the dihedral groups up to order 94; however, this is beyond the scope of this paper. If the reader wishes to investigate these findings, please refer to the Air Force Technical Report "A Number-Theoretic Approach to Subgroups of Dihedral Groups" by the same authors [4].

Chapter III

Examples are an indispensable part of a mathematician's research. Only after a thorough examination of numerous examples, should a mathematician attempt to formulate their common properties into a theorem. Then and only then does the mathematician undertake to prove the theorem.

An instructor teaching group theory is always searching for examples to help clarify and justify the explanation of theorems and definitions, and the student, at any level, always needs examples to elucidate the subject at hand.

Another role for an example is its use as a counterexample. Many apparent theorems or conjectures have been proved wrong by a simple counterexample. For instance, a well-known theorem states that every subgroup of an abelian group is a normal subgroup. However, its converse, if every subgroup of a group is normal, then the group is abelian, is not true. The quaternion group of order eight has all subgroups normal but is definitely not abelian. Thus, by a simple counterexample a possible theorem is disproved.

From the above, it is easy to see why examples play such an important role in the study of group theory. To this end, Weinstein [5] has written a book devoted entirely to examples in group theory. His book is quite comprehensive, providing examples to illustrate various group-theoretical concepts, and presenting some very important counterexamples.

In this chapter, the authors develop several examples from the data generated by the abstract algebra programming system presented in chapter two. Since, these examples are just the tip of the iceberg, instructors should use the system to develop more in-depth ideas and examples.

Isomorphism is a fundamental, but crucial idea of abstract algebra. The beginning student generally finds the concept of isomorphism difficult at best. To strengthen their understanding of isomorphism, an instructor may wish to have students develop both an example and a counterexample. An exercise given to a student using this program might be to generate Z(5)xZ(2) and determine if an isomorphic mapping is possible with C(10). Also, using the data that already exists, a student could demonstrate why q8 and sd1z8z2 or q4xz4 and sd2z8z2 are not isomorphic.

Another concept that could possibly be reinforced by this system is coset theory. As an exercise, a student could be asked to pick any of the order sixteen groups, determine a right coset, and then prove that it is also a left coset. This exercise requires a host of activities that this system can perform, such as developing all the elements and subgroups of a group.

The idea of cosets leads naturally to the notion of a factor group. Fraleigh [1] defines a factor group in the following way:

If N is a normal subgroup of a group G, the group cosets of N under the induced operation is the factor group of G modulo N,

and is denoted G/N. The cosets are the residue classes of G modulo N.

An easy exercise to assign a student would be to find the normal subgroups of a group. A team project might be to develop the programs to print out the normal subgroups of the groups in the System Data Base (this requires only three short modules of fifty to sixty lines each [6]) .

A fourth example for the use of this program in the classroom are the Sylow Theorems. One must remember that the Sylow Theorems deal with finite groups. The first Sylow Theorem states that for any prime dividing the order of the group there exist a subgroup of that order. Since we have a data base of forty non-isomorphic groups it would be an easy task for the student to find the subgroups that have a prime order. Then using the third Sylow Theorem, show how many Sylow p-subgroups exist of that order.

Finally, given this programming system, a student might well venture out into the realms of finite group theory and discover a host of interesting topics. Hopefully he will ask questions of his instructor and start down the long road of self-study that leads to mathematical maturity.

BIBLIOGRAPHY

1. Fraleigh, J.B., A First Course in Abstract Algebra, Third Edition, Addison-Wesley Publishing Company, Inc., 1982.

2. Knuth, D.E., The Art of Computer Programming Volume I. Fundamental Algorithms. Addison-Wesley Publishing Company Inc., 1969.

3. Borland, Turbo C, Borland International Inc., 1988.

4. Jensen, D.W. & Keane, M.K., A Number-Theoretic Approach to Subgroups of Dihedral Groups, USAFA Technical Report, 1990

5. Weinstein, M., Examples of Groups, Polygonol Publishing House, 1977

6. Keane, M.K., The Personal Computer as an Aid in The Study of Group Theory, Microfilms International, 1985.

APPENDIX A

DIHEDRAL GROUP D(3)

The ELEMENTS are the following:

e0 is the identity element
e1 = (1 2 3)
e2 = (2 3)
e3 = (1 3 2)
e4 = (1 3)
e5 = (1 2)

The subgroups are the following:

S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e4 }
S3 = { e0 e5 }
S4 = { e0 e1 e3 }
S5 = { e0 e1 e2 e3 e4 e5 }

DIHEDRAL GROUP D(4)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 )
e2 = ( 2 3 )( 1 4 )
e3 = ( 2 4 )( 1 3 )
e4 = ( 1 4 3 2 )
e5 = ( 1 3 )
e6 = ( 2 4 )
e7 = ( 3 4 )( 1 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e3 }
S3 = { e0 e5 }
S4 = { e0 e6 }
S5 = { e0 e7 }
S6 = { e0 e1 e3 e4 }
S7 = { e0 e2 e3 e7 }
S8 = { e0 e3 e5 e6 }
S9 = { e0 e1 e2 e3 e4 e5 e6 e7 }
```

DIHEDRAL GROUP D(5)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 )
e2 = ( 3 4 )( 2 5 )
e3 = ( 1 3 5 2 4 )
e4 = ( 1 4 2 5 3 )
e5 = ( 1 5 4 3 2 )
e6 = ( 2 4 )( 1 5 )
e7 = ( 3 5 )( 1 2 )
e8 = ( 2 3 )( 1 4 )
e9 = ( 4 5 )( 1 3 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e6 }
S3 = { e0 e7 }
S4 = { e0 e8 }
S5 = { e0 e9 }
S6 = { e0 e1 e3 e4 e5 }
S7 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 }
```

DIHEDRAL GROUP D(6)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 )
e2 = ( 3 4 )( 2 5 )( 1 6 )
e3 = ( 2 4 6 )( 1 3 5 )
e4 = ( 3 6 )( 2 5 )( 1 4 )
e5 = ( 2 6 4 )( 1 5 3 )
e6 = ( 1 6 5 4 3 2 )
e7 = ( 2 4 )( 1 5 )
e8 = ( 3 5 )( 2 6 )
e9 = ( 5 6 )( 2 3 )( 1 4 )
e10 = ( 4 5 )( 3 6 )( 1 2 )
e11 = ( 4 6 )( 1 3 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e4 }
S3 = { e0 e7 }
S4 = { e0 e8 }
S5 = { e0 e9 }
S6 = { e0 e10 }
S7 = { e0 e11 }
S8 = { e0 e3 e5 }
S9 = { e0 e2 e4 e11 }
S10 = { e0 e4 e7 e10 }
S11 = { e0 e4 e8 e9 }
S12 = { e0 e1 e3 e4 e5 e6 }
S13 = { e0 e2 e3 e5 e9 e10 }
S14 = { e0 e3 e5 e7 e8 e11 }
S15 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 }
```

DIHEDRAL GROUP D(7)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 7 )
e2 = ( 4 5 )( 3 6 )( 2 7 )
e3 = ( 1 3 5 7 2 4 6 )
e4 = ( 1 4 7 3 6 2 5 )
e5 = ( 1 5 2 6 3 7 4 )
e6 = ( 1 6 4 2 7 5 3 )
e7 = ( 1 7 6 5 4 3 2 )
e8 = ( 3 5 )( 2 6 )( 1 7 )
e9 = ( 4 6 )( 3 7 )( 1 2 )
e10 = ( 3 4 )( 2 5 )( 1 6 )
e11 = ( 5 6 )( 4 7 )( 1 3 )
e12 = ( 6 7 )( 2 4 )( 1 5 )
e13 = ( 5 7 )( 2 3 )( 1 4 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e8 }
S3 = { e0 e9 }
S4 = { e0 e10 }
S5 = { e0 e11 }
S6 = { e0 e12 }
S7 = { e0 e13 }
S8 = { e0 e1 e3 e4 e5 e6 e7 }
S9 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 }
```

DIHEDRAL GROUP D(8)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 7 8 )
e2 = ( 4 5 )( 3 6 )( 2 7 )( 1 8 )
e3 = ( 2 4 6 8 )( 1 3 5 7 )
e4 = ( 1 4 7 2 5 8 3 6 )
e5 = ( 4 8 )( 3 7 )( 2 6 )( 1 5 )
e6 = ( 1 6 3 8 5 2 7 4 )
e7 = ( 2 8 6 4 )( 1 7 5 3 )
e8 = ( 1 8 7 6 5 4 3 2 )
e9 = ( 3 5 )( 2 6 )( 1 7 )
e10 = ( 4 6 )( 3 7 )( 2 8 )
e11 = ( 7 8 )( 3 4 )( 2 5 )( 1 6 )
e12 = ( 5 6 )( 4 7 )( 3 8 )( 1 2 )
e13 = ( 6 8 )( 2 4 )( 1 5 )
e14 = ( 5 7 )( 4 8 )( 1 3 )
e15 = ( 6 7 )( 5 8 )( 2 3 )( 1 4 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e5 }
S3 = { e0 e9 }
S4 = { e0 e10 }
S5 = { e0 e11 }
S6 = { e0 e12 }
S7 = { e0 e13 }
S8 = { e0 e14 }
S9 = { e0 e15 }
S10 = { e0 e3 e5 e7 }
S11 = { e0 e2 e5 e15 }
S12 = { e0 e5 e9 e14 }
S13 = { e0 e5 e10 e13 }
S14 = { e0 e5 e11 e12 }
S15 = { e0 e1 e3 e4 e5 e6 e7 e8 }
S16 = { e0 e2 e3 e5 e7 e11 e12 e15 }
S17 = { e0 e3 e5 e7 e9 e10 e13 e14 }
S18 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }
```

CYCLIC GROUP C(3)

The ELEMENTS are the following:

e0 is the identity element
e1 = (1 2 3)
e2 = (1 3 2)

The subgroups are the following:

S0 is the null subgroup
S1 = { e0 e1 e2 }

CYCLIC GROUP C(4)

The ELEMENTS are the following:

e0 is the identity element
e1 = (1 2 3 4)
e2 = (2 4)(1 3)
e3 = (1 4 3 2)

The subgroups are the following:

S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e1 e2 e3 }

CYCLIC GROUP C(5)

The ELEMENTS are the following:

e0 is the identity element
e1 = (1 2 3 4 5)
e2 = (1 3 5 2 4)
e3 = (1 4 2 5 3)
e4 = (1 5 4 3 2)

The subgroups are the following:

S0 is the null subgroup
S1 = { e0 e1 e2 e3 e4 }

CYCLIC GROUP C(6)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 )
e2 = ( 2 4 6 )( 1 3 5 )
e3 = ( 3 6 )( 2 5 )( 1 4 )
e4 = ( 2 6 4 )( 1 5 3 )
e5 = ( 1 6 5 4 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e3 }
S2 = { e0 e2 e4 }
S3 = { e0 e1 e2 e3 e4 e5 }
```

CYCLIC GROUP C(7)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 7 )
e2 = ( 1 3 5 7 2 4 6 )
e3 = ( 1 4 7 3 6 2 5 )
e4 = ( 1 5 2 6 3 7 4 )
e5 = ( 1 6 4 2 7 5 3 )
e6 = ( 1 7 6 5 4 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e1 e2 e3 e4 e5 e6 }
```

CYCLIC GROUP C(8)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 7 8 )
e2 = ( 2 4 6 8 )( 1 3 5 7 )
e3 = ( 1 4 7 2 5 8 3 6 )
e4 = ( 4 8 )( 3 7 )( 2 6 )( 1 5 )
e5 = ( 1 6 3 8 5 2 7 4 )
e6 = ( 2 8 6 4 )( 1 7 5 3 )
e7 = ( 1 8 7 6 5 4 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e4 }
S2 = { e0 e2 e4 e6 }
S3 = { e0 e1 e2 e3 e4 e5 e6 e7 }
```

CYCLIC GROUP C(9)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 7 8 9 )
e2 = ( 1 3 5 7 9 2 4 6 8 )
e3 = ( 3 6 9 )( 2 5 8 )( 1 4 7 )
e4 = ( 1 5 9 4 8 3 7 2 6 )
e5 = ( 1 6 2 7 3 8 4 9 5 )
e6 = ( 3 9 6 )( 2 8 5 )( 1 7 4 )
e7 = ( 1 8 6 4 2 9 7 5 3 )
e8 = ( 1 9 8 7 6 5 4 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e3 e6 }
S2 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 }
```

CYCLIC GROUP C(10)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 7 8 9 10 )
e2 = ( 2 4 6 8 10 )( 1 3 5 7 9 )
e3 = ( 1 4 7 10 3 6 9 2 5 8 )
e4 = ( 2 6 10 4 8 )( 1 5 9 3 7 )
e5 = ( 5 10 )( 4 9 )( 3 8 )( 2 7 )( 1 6 )
e6 = ( 2 8 4 10 6 )( 1 7 3 9 5 )
e7 = ( 1 8 5 2 9 6 3 10 7 4 )
e8 = ( 2 10 8 6 4 )( 1 9 7 5 3 )
e9 = ( 1 10 9 8 7 6 5 4 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e5 }
S2 = { e0 e2 e4 e6 e8 }
S3 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 }
```

CYCLIC GROUP C(11)

The ELEMENTS are the following:

e0 is the identity element
e1 = (1 2 3 4 5 6 7 8 9 10 11)
e2 = (1 3 5 7 9 11 2 4 6 8 10)
e3 = (1 4 7 10 2 5 8 11 3 6 9)
e4 = (1 5 9 2 6 10 3 7 11 4 8)
e5 = (1 6 11 5 10 4 9 3 8 2 7)
e6 = (1 7 2 8 3 9 4 10 5 11 6)
e7 = (1 8 4 11 7 3 10 6 2 9 5)
e8 = (1 9 6 3 11 8 5 2 10 7 4)
e9 = (1 10 8 6 4 2 11 9 7 5 3)
e10 = (1 11 10 9 8 7 6 5 4 3 2)

The subgroups are the following:

S0 is the null subgroup
S1 = (e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10)

CYCLIC GROUP C(12)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 7 8 9 10 11 12 )
e2 = ( 2 4 6 8 10 12 )( 1 3 5 7 9 11 )
e3 = ( 3 6 9 12 )( 2 5 8 11 )( 1 4 7 10 )
e4 = ( 4 8 12 )( 3 7 11 )( 2 6 10 )( 1 5 9 )
e5 = ( 1 6 11 4 9 2 7 12 5 10 3 8 )
e6 = ( 6 12 )( 5 11 )( 4 10 )( 3 9 )( 2 8 )( 1 7 )
e7 = ( 1 8 3 10 5 12 7 2 9 4 11 6 )
e8 = ( 4 12 8 )( 3 11 7 )( 2 10 6 )( 1 9 5 )
e9 = ( 3 12 9 6 )( 2 11 8 5 )( 1 10 7 4 )
e10 = ( 2 12 10 8 6 4 )( 1 11 9 7 5 3 )
e11 = ( 1 12 11 10 9 8 7 6 5 4 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e6 }
S2 = { e0 e4 e8 }
S3 = { e0 e3 e6 e9 }
S4 = { e0 e2 e4 e6 e8 e10 }
S5 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 }
```

CYCLIC GROUP C(13)

The ELEMENTS are the following:

e0 is the identity element
e1 = (1 2 3 4 5 6 7 8 9 10 11 12 13)
e2 = (1 3 5 7 9 11 13 2 4 6 8 10 12)
e3 = (1 4 7 10 13 3 6 9 12 2 5 8 11)
e4 = (1 5 9 13 4 8 12 3 7 11 2 6 10)
e5 = (1 6 11 3 8 13 5 10 2 7 12 4 9)
e6 = (1 7 13 6 12 5 11 4 10 3 9 2 8)
e7 = (1 8 2 9 3 10 4 11 5 12 6 13 7)
e8 = (1 9 4 12 7 2 10 5 13 8 3 11 6)
e9 = (1 10 6 2 11 7 3 12 8 4 13 9 5)
e10 = (1 11 8 5 2 12 9 6 3 13 10 7 4)
e11 = (1 12 10 8 6 4 2 13 11 9 7 5 3)
e12 = (1 13 12 11 10 9 8 7 6 5 4 3 2)

The subgroups are the following:

S0 is the null subgroup
S1 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 }

CYCLIC GROUP C(14)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 7 8 9 10 11 12 13 14 )
e2 = ( 2 4 6 8 10 12 14 )( 1 3 5 7 9 11 13 )
e3 = ( 1 4 7 10 13 2 5 8 11 14 3 6 9 12 )
e4 = ( 2 6 10 14 4 8 12 )( 1 5 9 13 3 7 11 )
e5 = ( 1 6 11 2 7 12 3 8 13 4 9 14 5 10 )
e6 = ( 2 8 14 6 12 4 10 )( 1 7 13 5 11 3 9 )
e7 = ( 7 14 )( 6 13 )( 5 12 )( 4 11 )( 3 10 )( 2 9 )( 1 8 )
e8 = ( 2 10 4 12 6 14 8 )( 1 9 3 11 5 13 7 )
e9 = ( 1 10 5 14 9 4 13 8 3 12 7 2 11 6 )
e10 = ( 2 12 8 4 14 10 6 )( 1 11 7 3 13 9 5 )
e11 = ( 1 12 9 6 3 14 11 8 5 2 13 10 7 4 )
e12 = ( 2 14 12 10 8 6 4 )( 1 13 11 9 7 5 3 )
e13 = ( 1 14 13 12 11 10 9 8 7 6 5 4 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e7 }
S2 = { e0 e2 e4 e6 e8 e10 e12 }
S3 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 }
```

CYCLIC GROUP C(15)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 )
e2 = ( 1 3 5 7 9 11 13 15 2 4 6 8 10 12 14 )
e3 = ( 3 6 9 12 15 )( 2 5 8 11 14 )( 1 4 7 10 13 )
e4 = ( 1 5 9 13 2 6 10 14 3 7 11 15 4 8 12 )
e5 = ( 5 10 15 )( 4 9 14 )( 3 8 13 )( 2 7 12 )( 1 6 11 )
e6 = ( 3 9 15 6 12 )( 2 8 14 5 11 )( 1 7 13 4 10 )
e7 = ( 1 8 15 7 14 6 13 5 12 4 11 3 10 2 9 )
e8 = ( 1 9 2 10 3 11 4 12 5 13 6 14 7 15 8 )
e9 = ( 3 12 6 15 9 )( 2 11 5 14 8 )( 1 10 4 13 7 )
e10 = ( 5 15 10 )( 4 14 9 )( 3 13 8 )( 2 12 7 )( 1 11 6 )
e11 = ( 1 12 8 4 15 11 7 3 14 10 6 2 13 9 5 )
e12 = ( 3 15 12 9 6 )( 2 14 11 8 5 )( 1 13 10 7 4 )
e13 = ( 1 14 12 10 8 6 4 2 15 13 11 9 7 5 3 )
e14 = ( 1 15 14 13 12 11 10 9 8 7 6 5 4 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e5 e10 }
S2 = { e0 e3 e6 e9 e12 }
S3 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14 }
```

CYCLIC GROUP C(16)

The ELEMENTS are the following:

e0 is the identity element
e1 = (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16)
e2 = (2 4 6 8 10 12 14 16)(1 3 5 7 9 11 13 15)
e3 = (1 4 7 10 13 16 3 6 9 12 15 2 5 8 11 14)
e4 = (4 8 12 16)(3 7 11 15)(2 6 10 14)(1 5 9 13)
e5 = (1 6 11 16 5 10 15 4 9 14 3 8 13 2 7 12)
e6 = (2 8 14 4 10 16 6 12)(1 7 13 3 9 15 5 11)
e7 = (1 8 15 6 13 4 11 2 9 16 7 14 5 12 3 10)
e8 = (8 16)(7 15)(6 14)(5 13)(4 12)(3 11)(2 10)(1 9)
e9 = (1 10 3 12 5 14 7 16 9 2 11 4 13 6 15 8)
e10 = (2 12 6 16 10 4 14 8)(1 11 5 15 9 3 13 7)
e11 = (1 12 7 2 13 8 3 14 9 4 15 10 5 16 11 6)
e12 = (4 16 12 8)(3 15 11 7)(2 14 10 6)(1 13 9 5)
e13 = (1 14 11 8 5 2 15 12 9 6 3 16 13 10 7 4)
e14 = (2 16 14 12 10 8 6 4)(1 15 13 11 9 7 5 3)
e15 = (1 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2)

The subgroups are the following:

S0 is the null subgroup
S1 = { e0 e8 }
S2 = { e0 e4 e8 e12 }
S3 = { e0 e2 e4 e6 e8 e10 e12 e14 }
S4 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }

ABELIAN GROUP (a4xa2)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 )
e2 = ( 5 6 )
e3 = ( 2 4 )( 1 3 )
e4 = ( 1 4 3 2 )
e5 = ( 5 6 )( 1 2 3 4 )
e6 = ( 5 6 )( 2 4 )( 1 3 )
e7 = ( 5 6 )( 1 4 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e3 }
S3 = { e0 e6 }
S4 = { e0 e1 e3 e4 }
S5 = { e0 e3 e5 e7 }
S6 = { e0 e2 e3 e6 }
S7 = { e0 e1 e2 e3 e4 e5 e6 e7 }
```

ABELIAN GROUP (a2a2a2)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 )
e2 = ( 3 4 )
e3 = ( 5 6 )
e4 = ( 3 4 )( 1 2 )
e5 = ( 5 6 )( 1 2 )
e6 = ( 5 6 )( 3 4 )
e7 = ( 5 6 )( 3 4 )( 1 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e1 }
S2 = { e0 e2 }
S3 = { e0 e3 }
S4 = { e0 e4 }
S5 = { e0 e5 }
S6 = { e0 e6 }
S7 = { e0 e7 }
S8 = { e0 e1 e2 e4 }
S9 = { e0 e1 e3 e5 }
S10 = { e0 e2 e3 e6 }
S11 = { e0 e2 e5 e7 }
S12 = { e0 e3 e4 e7 }
S13 = { e0 e4 e5 e6 }
S14 = { e0 e1 e6 e7 }
S15 = { e0 e1 e2 e3 e4 e5 e6 e7 }
```

ABELIAN GROUP (a3xa3)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 )
e2 = ( 4 5 6 )
e3 = ( 1 3 2 )
e4 = ( 4 6 5 )
e5 = ( 4 5 6 )( 1 2 3 )
e6 = ( 4 6 5 )( 1 2 3 )
e7 = ( 4 5 6 )( 1 3 2 )
e8 = ( 4 6 5 )( 1 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e1 e3 }
S2 = { e0 e2 e4 }
S3 = { e0 e5 e8 }
S4 = { e0 e6 e7 }
S5 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 }
```

ABELIAN GROUP (a6xa2)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 )
e2 = ( 7 8 )
e3 = ( 2 4 6 )( 1 3 5 )
e4 = ( 3 6 )( 2 5 )( 1 4 )
e5 = ( 2 6 4 )( 1 5 3 )
e6 = ( 1 6 5 4 3 2 )
e7 = ( 7 8 )( 1 2 3 4 5 6 )
e8 = ( 7 8 )( 2 4 6 )( 1 3 5 )
e9 = ( 7 8 )( 3 6 )( 2 5 )( 1 4 )
e10 = ( 7 8 )( 2 6 4 )( 1 5 3 )
e11 = ( 7 8 )( 1 6 5 4 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e4 }
S3 = { e0 e9 }
S4 = { e0 e3 e5 }
S5 = { e0 e2 e4 e9 }
S6 = { e0 e1 e3 e4 e5 e6 }
S7 = { e0 e3 e5 e7 e9 e11 }
S8 = { e0 e2 e3 e5 e8 e10 }
S9 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 }
```

ABELIAN GROUP (a2x8)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 5 6 7 8 )
e2 = ( 9 10 )
e3 = ( 2 4 6 8 )( 1 3 5 7 )
e4 = ( 1 4 7 2 5 8 3 6 )
e5 = ( 4 8 )( 3 7 )( 2 6 )( 1 5 )
e6 = ( 1 6 3 8 5 2 7 4 )
e7 = ( 2 8 6 4 )( 1 7 5 3 )
e8 = ( 1 8 7 6 5 4 3 2 )
e9 = ( 9 10 )( 1 2 3 4 5 6 7 8 )
e10 = ( 9 10 )( 2 4 6 8 )( 1 3 5 7 )
e11 = ( 9 10 )( 1 4 7 2 5 8 3 6 )
e12 = ( 9 10 )( 4 8 )( 3 7 )( 2 6 )( 1 5 )
e13 = ( 9 10 )( 1 6 3 8 5 2 7 4 )
e14 = ( 9 10 )( 2 8 6 4 )( 1 7 5 3 )
e15 = ( 9 10 )( 1 8 7 6 5 4 3 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e5 }
S3 = { e0 e12 }
S4 = { e0 e3 e5 e7 }
S5 = { e0 e5 e10 e14 }
S6 = { e0 e2 e5 e12 }
S7 = { e0 e1 e3 e4 e5 e6 e7 e8 }
S8 = { e0 e3 e5 e7 e9 e11 e13 e15 }
S9 = { e0 e2 e3 e5 e7 e10 e12 e14 }
S10 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }
```

ABELIAN GROUP (a4x4)

The ELEMENTS are the following:

e0 is the identity element
e1 = (1 2 3 4)
e2 = (5 6 7 8)
e3 = (2 4)(1 3)
e4 = (1 4 3 2)
e5 = (6 8)(5 7)
e6 = (5 8 7 6)
e7 = (5 6 7 8)(1 2 3 4)
e8 = (6 8)(5 7)(1 2 3 4)
e9 = (5 8 7 6)(1 2 3 4)
e10 = (5 6 7 8)(2 4)(1 3)
e11 = (6 8)(5 7)(2 4)(1 3)
e12 = (5 8 7 6)(2 4)(1 3)
e13 = (5 6 7 8)(1 4 3 2)
e14 = (6 8)(5 7)(1 4 3 2)
e15 = (5 8 7 6)(1 4 3 2)

The subgroups are the following:

S0 is the null subgroup
S1 = { e0 e3 }
S2 = { e0 e5 }
S3 = { e0 e11 }
S4 = { e0 e1 e3 e4 }
S5 = { e0 e2 e5 e6 }
S6 = { e0 e7 e11 e15 }
S7 = { e0 e3 e8 e14 }
S8 = { e0 e9 e11 e13 }
S9 = { e0 e5 e10 e12 }
S10 = { e0 e3 e5 e11 }
S11 = { e0 e1 e3 e4 e5 e8 e11 e14 }
S12 = { e0 e2 e3 e5 e6 e10 e11 e12 }
S13 = { e0 e3 e5 e7 e9 e11 e13 e15 }
S14 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }

ABELIAN GROUP (a2x2x4)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 )
e2 = ( 3 4 )
e3 = ( 5 6 7 8 )
e4 = ( 6 8 )( 5 7 )
e5 = ( 5 8 7 6 )
e6 = ( 3 4 )( 1 2 )
e7 = ( 5 6 7 8 )( 1 2 )
e8 = ( 6 8 )( 5 7 )( 1 2 )
e9 = ( 5 8 7 6 )( 1 2 )
e10 = ( 5 6 7 8 )( 3 4 )
e11 = ( 6 8 )( 5 7 )( 3 4 )
e12 = ( 5 8 7 6 )( 3 4 )
e13 = ( 5 6 7 8 )( 3 4 )( 1 2 )
e14 = ( 6 8 )( 5 7 )( 3 4 )( 1 2 )
e15 = ( 5 8 7 6 )( 3 4 )( 1 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e1 }
S2 = { e0 e2 }
S3 = { e0 e4 }
S4 = { e0 e6 }
S5 = { e0 e8 }
S6 = { e0 e11 }
S7 = { e0 e14 }
S8 = { e0 e3 e4 e5 }
S9 = { e0 e1 e2 e6 }
S10 = { e0 e4 e7 e9 }
S11 = { e0 e1 e4 e8 }
S12 = { e0 e4 e10 e12 }
S13 = { e0 e4 e13 e15 }
S14 = { e0 e2 e4 e11 }
S15 = { e0 e2 e8 e14 }
S16 = { e0 e4 e6 e14 }
S17 = { e0 e6 e8 e11 }
S18 = { e0 e1 e11 e14 }
S19 = { e0 e1 e3 e4 e5 e7 e8 e9 }
S20 = { e0 e2 e3 e4 e5 e10 e11 e12 }
S21 = { e0 e2 e4 e7 e9 e11 e13 e15 }
S22 = { e0 e1 e2 e4 e6 e8 e11 e14 }
S23 = { e0 e3 e4 e5 e6 e13 e14 e15 }
S24 = { e0 e4 e6 e7 e9 e10 e12 e14 }
S25 = { e0 e1 e4 e8 e10 e12 e13 e15 }
S26 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }
```

ABELIAN GROUP (a22x22)

The ELEMENTS are the following:

e0 is the identity element
e1 = (1 2)
e2 = (3 4)
e3 = (5 6)
e4 = (7 8)
e5 = (3 4)(1 2)
e6 = (5 6)(1 2)
e7 = (7 8)(1 2)
e8 = (5 6)(3 4)
e9 = (5 6)(3 4)(1 2)
e10 = (7 8)(5 6)
e11 = (7 8)(5 6)(1 2)
e12 = (7 8)(3 4)
e13 = (7 8)(3 4)(1 2)
e14 = (7 8)(5 6)(3 4)
e15 = (7 8)(5 6)(3 4)(1 2)

The subgroups are the following:

S0 is the null subgroup
S1 = { e0 e1 }
S2 = { e0 e2 }
S3 = { e0 e3 }
S4 = { e0 e4 }
S5 = { e0 e5 }
S6 = { e0 e6 }
S7 = { e0 e7 }
S8 = { e0 e8 }
S9 = { e0 e9 }
S10 = { e0 e10 }
S11 = { e0 e11 }
S12 = { e0 e12 }
S13 = { e0 e13 }
S14 = { e0 e14 }
S15 = { e0 e15 }
S16 = { e0 e1 e2 e5 }
S17 = { e0 e1 e3 e6 }
S18 = { e0 e1 e4 e7 }
S19 = { e0 e1 e10 e11 }
S20 = { e0 e2 e3 e8 }
S21 = { e0 e2 e4 e12 }
S22 = { e0 e2 e6 e9 }
S23 = { e0 e2 e7 e13 }
S24 = { e0 e2 e10 e14 }
S25 = { e0 e2 e11 e15 }
S26 = { e0 e3 e4 e10 }
S27 = { e0 e3 e5 e9 }
S28 = { e0 e3 e7 e11 }
S29 = { e0 e4 e5 e13 }

S30 = { e0 e4 e6 e11 }
S31 = { e0 e4 e8 e14 }
S32 = { e0 e4 e9 e15 }
S33 = { e0 e5 e6 e8 }
S34 = { e0 e5 e7 e12 }
S35 = { e0 e5 e10 e15 }
S36 = { e0 e5 e11 e14 }
S37 = { e0 e6 e7 e10 }
S38 = { e0 e7 e8 e15 }
S39 = { e0 e7 e9 e14 }
S40 = { e0 e1 e8 e9 }
S41 = { e0 e8 e10 e12 }
S42 = { e0 e8 e11 e13 }
S43 = { e0 e9 e10 e13 }
S44 = { e0 e9 e11 e12 }
S45 = { e0 e1 e12 e13 }
S46 = { e0 e3 e12 e14 }
S47 = { e0 e6 e12 e15 }
S48 = { e0 e3 e13 e15 }
S49 = { e0 e6 e13 e14 }
S50 = { e0 e1 e14 e15 }
S51 = { e0 e1 e2 e3 e5 e6 e8 e9 }
S52 = { e0 e1 e3 e4 e6 e7 e10 e11 }
S53 = { e0 e1 e2 e4 e5 e7 e12 e13 }
S54 = { e0 e1 e2 e5 e10 e11 e14 e15 }
S55 = { e0 e1 e4 e7 e8 e9 e14 e15 }
S56 = { e0 e1 e3 e6 e12 e13 e14 e15 }
S57 = { e0 e1 e8 e9 e10 e11 e12 e13 }
S58 = { e0 e2 e3 e4 e8 e10 e12 e14 }
S59 = { e0 e2 e3 e7 e8 e11 e13 e15 }
S60 = { e0 e2 e4 e6 e9 e11 e12 e15 }
S61 = { e0 e2 e6 e7 e9 e10 e13 e14 }
S62 = { e0 e3 e4 e5 e9 e10 e13 e15 }
S63 = { e0 e3 e5 e7 e9 e11 e12 e14 }
S64 = { e0 e4 e5 e6 e8 e11 e13 e14 }
S65 = { e0 e5 e6 e7 e8 e10 e12 e15 }
S66 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }

UNKNOWN GROUP (klein4)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 3 4 )( 1 2 )
e2 = ( 2 3 )( 1 4 )
e3 = ( 2 4 )( 1 3 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e1 }
S2 = { e0 e2 }
S3 = { e0 e3 }
S4 = { e0 e1 e2 e3 }
```

UNKNOWN GROUP (q4)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 5 8 7 6 )( 1 4 3 2 )
e2 = ( 2 5 4 7 )( 1 6 3 8 )
e3 = ( 6 8 )( 5 7 )( 2 4 )( 1 3 )
e4 = ( 5 6 7 8 )( 1 2 3 4 )
e5 = ( 2 7 4 5 )( 1 8 3 6 )
e6 = ( 2 6 4 8 )( 1 7 3 5 )
e7 = ( 2 8 4 6 )( 1 5 3 7 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e3 }
S2 = { e0 e1 e3 e4 }
S3 = { e0 e2 e3 e5 }
S4 = { e0 e3 e6 e7 }
S5 = { e0 e1 e2 e3 e4 e5 e6 e7 }
```

UNKNOWN GROUP (alt4)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 )
e2 = ( 2 3 4 )
e3 = ( 2 3 )( 1 4 )
e4 = ( 1 3 2 )
e5 = ( 2 4 3 )
e6 = ( 2 4 )( 1 3 )
e7 = ( 3 4 )( 1 2 )
e8 = ( 1 4 3 )
e9 = ( 1 2 4 )
e10 = ( 1 4 2 )
e11 = ( 1 3 4 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e3 }
S2 = { e0 e6 }
S3 = { e0 e7 }
S4 = { e0 e1 e4 }
S5 = { e0 e2 e5 }
S6 = { e0 e8 e11 }
S7 = { e0 e9 e10 }
S8 = { e0 e3 e6 e7 }
S9 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 }
```

UNKNOWN GROUP (s3xz2)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 )
e2 = ( 1 3 )
e3 = ( 4 5 )
e4 = ( 1 3 2 )
e5 = ( 1 2 )
e6 = ( 2 3 )
e7 = ( 4 5 )( 1 2 3 )
e8 = ( 4 5 )( 1 3 2 )
e9 = ( 4 5 )( 1 3 )
e10 = ( 4 5 )( 2 3 )
e11 = ( 4 5 )( 1 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e3 }
S3 = { e0 e5 }
S4 = { e0 e6 }
S5 = { e0 e9 }
S6 = { e0 e10 }
S7 = { e0 e11 }
S8 = { e0 e1 e4 }
S9 = { e0 e2 e3 e9 }
S10 = { e0 e3 e5 e11 }
S11 = { e0 e3 e6 e10 }
S12 = { e0 e1 e2 e4 e5 e6 }
S13 = { e0 e1 e3 e4 e7 e8 }
S14 = { e0 e1 e4 e9 e10 e11 }
S15 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 }
```

UNKNOWN GROUP (sd2z8z2)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 9 16 15 14 13 12 11 10 )( 1 8 7 6 5 4 3 2 )
e2 = ( 8 12 )( 7 9 )( 6 14 )( 5 11 )( 4 16 )( 3 13 )( 2 10 )( 1 15 )
e3 = ( 10 16 14 12 )( 9 15 13 11 )( 2 8 6 4 )( 1 7 5 3 )
e4 = ( 9 14 11 16 13 10 15 12 )( 1 6 3 8 5 2 7 4 )
e5 = ( 12 16 )( 11 15 )( 10 14 )( 9 13 )( 4 8 )( 3 7 )( 2 6 )( 1 5 )
e6 = ( 9 12 15 10 13 16 11 14 )( 1 4 7 2 5 8 3 6 )
e7 = ( 10 12 14 16 )( 9 11 13 15 )( 2 4 6 8 )( 1 3 5 7 )
e8 = ( 9 10 11 12 13 14 15 16 )( 1 2 3 4 5 6 7 8 )
e9 = ( 4 13 8 9 )( 3 10 7 14 )( 2 15 6 11 )( 1 12 5 16 )
e10 = ( 4 15 8 11 )( 3 12 7 16 )( 2 9 6 13 )( 1 14 5 10 )
e11 = ( 8 14 )( 7 11 )( 6 16 )( 5 13 )( 4 10 )( 3 15 )( 2 12 )( 1 9 )
e12 = ( 8 16 )( 7 13 )( 6 10 )( 5 15 )( 4 12 )( 3 9 )( 2 14 )( 1 11 )
e13 = ( 8 10 )( 7 15 )( 6 12 )( 5 9 )( 4 14 )( 3 11 )( 2 16 )( 1 13 )
e14 = ( 4 9 8 13 )( 3 14 7 10 )( 2 11 6 15 )( 1 16 5 12 )
e15 = ( 4 11 8 15 )( 3 16 7 12 )( 2 13 6 9 )( 1 10 5 14 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e5 }
S3 = { e0 e11 }
S4 = { e0 e12 }
S5 = { e0 e13 }
S6 = { e0 e3 e5 e7 }
S7 = { e0 e5 e9 e14 }
S8 = { e0 e5 e10 e15 }
S9 = { e0 e2 e5 e12 }
S10 = { e0 e5 e11 e13 }
S11 = { e0 e1 e3 e4 e5 e6 e7 e8 }
S12 = { e0 e2 e3 e5 e7 e11 e12 e13 }
S13 = { e0 e3 e5 e7 e9 e10 e14 e15 }
S14 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }
```

UNKNOWN GROUP (sd1z8z2)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 9 16 15 14 13 12 11 10 )( 1 8 7 6 5 4 3 2 )
e2 = ( 8 16 )( 7 11 )( 6 14 )( 5 9 )( 4 12 )( 3 15 )( 2 10 )( 1 13 )
e3 = ( 10 16 14 12 )( 9 15 13 11 )( 2 8 6 4 )( 1 7 5 3 )
e4 = ( 9 14 11 16 13 10 15 12 )( 1 6 3 8 5 2 7 4 )
e5 = ( 12 16 )( 11 15 )( 10 14 )( 9 13 )( 4 8 )( 3 7 )( 2 6 )( 1 5 )
e6 = ( 9 12 15 10 13 16 11 14 )( 1 4 7 2 5 8 3 6 )
e7 = ( 10 12 14 16 )( 9 11 13 15 )( 2 4 6 8 )( 1 3 5 7 )
e8 = ( 9 10 11 12 13 14 15 16 )( 1 2 3 4 5 6 7 8 )
e9 = ( 2 13 4 15 6 9 8 11 )( 1 16 3 10 5 12 7 14 )
e10 = ( 2 9 4 11 6 13 8 15 )( 1 12 3 14 5 16 7 10 )
e11 = ( 4 10 8 14 )( 3 13 7 9 )( 2 16 6 12 )( 1 11 5 15 )
e12 = ( 4 14 8 10 )( 3 9 7 13 )( 2 12 6 16 )( 1 15 5 11 )
e13 = ( 2 11 8 9 6 15 4 13 )( 1 14 7 12 5 10 3 16 )
e14 = ( 2 15 8 13 6 11 4 9 )( 1 10 7 16 5 14 3 12 )
e15 = ( 8 12 )( 7 15 )( 6 10 )( 5 13 )( 4 16 )( 3 11 )( 2 14 )( 1 9 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e5 }
S3 = { e0 e15 }
S4 = { e0 e3 e5 e7 }
S5 = { e0 e5 e11 e12 }
S6 = { e0 e2 e5 e15 }
S7 = { e0 e1 e3 e4 e5 e6 e7 e8 }
S8 = { e0 e3 e5 e7 e9 e10 e13 e14 }
S9 = { e0 e2 e3 e5 e7 e11 e12 e15 }
S10 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }
```

UNKNOWN GROUP (q8)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 9 16 15 14 13 12 11 10 )( 1 8 7 6 5 4 3 2 )
e2 = ( 4 15 8 11 )( 3 16 7 12 )( 2 9 6 13 )( 1 10 5 14 )
e3 = ( 10 16 14 12 )( 9 15 13 11 )( 2 8 6 4 )( 1 7 5 3 )
e4 = ( 9 14 11 16 13 10 15 12 )( 1 6 3 8 5 2 7 4 )
e5 = ( 12 16 )( 11 15 )( 10 14 )( 9 13 )( 4 8 )( 3 7 )( 2 6 )( 1 5 )
e6 = ( 9 12 15 10 13 16 11 14 )( 1 4 7 2 5 8 3 6 )
e7 = ( 10 12 14 16 )( 9 11 13 15 )( 2 4 6 8 )( 1 3 5 7 )
e8 = ( 9 10 11 12 13 14 15 16 )( 1 2 3 4 5 6 7 8 )
e9 = ( 4 11 8 15 )( 3 12 7 16 )( 2 13 6 9 )( 1 14 5 10 )
e10 = ( 4 16 8 12 )( 3 9 7 13 )( 2 10 6 14 )( 1 11 5 15 )
e11 = ( 4 14 8 10 )( 3 15 7 11 )( 2 16 6 12 )( 1 9 5 13 )
e12 = ( 4 12 8 16 )( 3 13 7 9 )( 2 14 6 10 )( 1 15 5 11 )
e13 = ( 4 10 8 14 )( 3 11 7 15 )( 2 12 6 16 )( 1 13 5 9 )
e14 = ( 4 9 8 13 )( 3 10 7 14 )( 2 11 6 15 )( 1 12 5 16 )
e15 = ( 4 13 8 9 )( 3 14 7 10 )( 2 15 6 11 )( 1 16 5 12 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e5 }
S2 = { e0 e2 e5 e9 }
S3 = { e0 e3 e5 e7 }
S4 = { e0 e5 e10 e12 }
S5 = { e0 e5 e11 e13 }
S6 = { e0 e5 e14 e15 }
S7 = { e0 e1 e3 e4 e5 e6 e7 e8 }
S8 = { e0 e2 e3 e5 e7 e9 e14 e15 }
S9 = { e0 e3 e5 e7 e10 e11 e12 e13 }
S10 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }
```

UNKNOWN GROUP (q4xz2)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 5 8 7 6 )( 1 4 3 2 )
e2 = ( 2 5 4 7 )( 1 6 3 8 )
e3 = ( 9 10 )
e4 = ( 6 8 )( 5 7 )( 2 4 )( 1 3 )
e5 = ( 5 6 7 8 )( 1 2 3 4 )
e6 = ( 2 7 4 5 )( 1 8 3 6 )
e7 = ( 2 6 4 8 )( 1 7 3 5 )
e8 = ( 2 8 4 6 )( 1 5 3 7 )
e9 = ( 9 10 )( 5 8 7 6 )( 1 4 3 2 )
e10 = ( 9 10 )( 6 8 )( 5 7 )( 2 4 )( 1 3 )
e11 = ( 9 10 )( 5 6 7 8 )( 1 2 3 4 )
e12 = ( 9 10 )( 2 5 4 7 )( 1 6 3 8 )
e13 = ( 9 10 )( 2 8 4 6 )( 1 5 3 7 )
e14 = ( 9 10 )( 2 6 4 8 )( 1 7 3 5 )
e15 = ( 9 10 )( 2 7 4 5 )( 1 8 3 6 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e3 }
S2 = { e0 e4 }
S3 = { e0 e10 }
S4 = { e0 e1 e4 e5 }
S5 = { e0 e2 e4 e6 }
S6 = { e0 e4 e7 e8 }
S7 = { e0 e4 e9 e11 }
S8 = { e0 e4 e12 e15 }
S9 = { e0 e4 e13 e14 }
S10 = { e0 e3 e4 e10 }
S11 = { e0 e1 e2 e4 e5 e6 e7 e8 }
S12 = { e0 e1 e3 e4 e5 e9 e10 e11 }
S13 = { e0 e2 e3 e4 e6 e10 e12 e15 }
S14 = { e0 e2 e4 e6 e9 e11 e13 e14 }
S15 = { e0 e3 e4 e7 e8 e10 e13 e14 }
S16 = { e0 e4 e7 e8 e9 e11 e12 e15 }
S17 = { e0 e1 e4 e5 e12 e13 e14 e15 }
S18 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }
```

UNKNOWN GROUP (d4xz2)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 1 2 3 4 )
e2 = ( 2 3 )( 1 4 )
e3 = ( 5 6 )
e4 = ( 2 4 )( 1 3 )
e5 = ( 1 4 3 2 )
e6 = ( 1 3 )
e7 = ( 2 4 )
e8 = ( 3 4 )( 1 2 )
e9 = ( 5 6 )( 1 2 3 4 )
e10 = ( 5 6 )( 2 4 )( 1 3 )
e11 = ( 5 6 )( 1 4 3 2 )
e12 = ( 5 6 )( 2 3 )( 1 4 )
e13 = ( 5 6 )( 2 4 )
e14 = ( 5 6 )( 1 3 )
e15 = ( 5 6 )( 3 4 )( 1 2 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e3 }
S3 = { e0 e4 }
S4 = { e0 e6 }
S5 = { e0 e7 }
S6 = { e0 e8 }
S7 = { e0 e10 }
S8 = { e0 e12 }
S9 = { e0 e13 }
S10 = { e0 e14 }
S11 = { e0 e15 }
S12 = { e0 e1 e4 e5 }
S13 = { e0 e4 e9 e11 }
S14 = { e0 e2 e3 e12 }
S15 = { e0 e2 e4 e8 }
S16 = { e0 e2 e10 e15 }
S17 = { e0 e3 e4 e10 }
S18 = { e0 e3 e6 e14 }
S19 = { e0 e3 e7 e13 }
S20 = { e0 e3 e8 e15 }
S21 = { e0 e4 e6 e7 }
S22 = { e0 e6 e10 e13 }
S23 = { e0 e7 e10 e14 }
S24 = { e0 e8 e10 e12 }
S25 = { e0 e4 e12 e15 }
S26 = { e0 e4 e13 e14 }
S27 = { e0 e1 e2 e4 e5 e6 e7 e8 }
S28 = { e0 e1 e3 e4 e5 e9 e10 e11 }
S29 = { e0 e2 e4 e8 e9 e11 e13 e14 }
S30 = { e0 e4 e6 e7 e9 e11 e12 e15 }
S31 = { e0 e1 e4 e5 e12 e13 e14 e15 }
S32 = { e0 e2 e3 e4 e8 e10 e12 e15 }
S33 = { e0 e3 e4 e6 e7 e10 e13 e14 }
S34 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }
```

UNKNOWN GROUP (q4xz4)

The ELEMENTS are the following:

e0 is the identity element
e1 = (7 16 15 14)(6 13 12 11)(5 10 9 8)(1 4 3 2)
e2 = (4 10 13 16)(3 5 12 7)(2 8 11 14)(1 9 6 15)
e3 = (14 16)(11 13)(8 10)(7 15)(6 12)(5 9)(2 4)(1 3)
e4 = (7 14 15 16)(6 11 12 13)(5 8 9 10)(1 2 3 4)
e5 = (10 16)(9 15)(8 14)(5 7)(4 13)(3 12)(2 11)(1 6)
e6 = (4 16 13 10)(3 7 12 5)(2 14 11 8)(1 15 6 9)
e7 = (4 5 13 7)(3 8 12 14)(2 9 11 15)(1 10 6 16)
e8 = (4 9 13 15)(3 10 12 16)(2 5 11 7)(1 8 6 14)
e9 = (7 10 15 8)(5 16 9 14)(2 6 4 12)(1 13 3 11)
e10 = (4 7 13 5)(3 14 12 8)(2 15 11 9)(1 16 6 10)
e11 = (4 15 13 9)(3 16 12 10)(2 7 11 5)(1 14 6 8)
e12 = (4 8 13 14)(3 9 12 15)(2 10 11 16)(1 5 6 7)
e13 = (10 14)(8 16)(7 9)(5 15)(4 11)(3 6)(2 13)(1 12)
e14 = (4 14 13 8)(3 15 12 9)(2 16 11 10)(1 7 6 5)
e15 = (7 8 15 10)(5 14 9 16)(2 12 4 6)(1 11 3 13)

The subgroups are the following:

S0 is the null subgroup
S1 = { e0 e3 }
S2 = { e0 e5 }
S3 = { e0 e13 }
S4 = { e0 e1 e3 e4 }
S5 = { e0 e2 e5 e6 }
S6 = { e0 e5 e7 e10 }
S7 = { e0 e5 e8 e11 }
S8 = { e0 e3 e9 e15 }
S9 = { e0 e5 e12 e14 }
S10 = { e0 e3 e5 e13 }
S11 = { e0 e1 e3 e4 e5 e9 e13 e15 }
S12 = { e0 e2 e3 e5 e6 e12 e13 e14 }
S13 = { e0 e3 e5 e7 e8 e10 e11 e13 }
S14 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }

UNKNOWN GROUP (wpd4z4)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 13 16 15 14 )( 6 12 10 8 )( 5 11 9 7 )( 1 4 3 2 )
e2 = ( 7 12 11 8 )( 5 10 9 6 )( 2 14 4 16 )( 1 13 3 15 )
e3 = ( 7 13 11 15 )( 5 16 9 14 )( 2 10 4 6 )( 1 8 3 12 )
e4 = ( 14 16 )( 13 15 )( 8 12 )( 7 11 )( 6 10 )( 5 9 )( 2 4 )( 1 3 )
e5 = ( 13 14 15 16 )( 6 8 10 12 )( 5 7 9 11 )( 1 2 3 4 )
e6 = ( 7 8 11 12 )( 5 6 9 10 )( 2 16 4 14 )( 1 15 3 13 )
e7 = ( 7 15 11 13 )( 5 14 9 16 )( 2 6 4 10 )( 1 12 3 8 )
e8 = ( 9 12 )( 7 10 )( 6 11 )( 5 8 )( 4 15 )( 3 14 )( 2 13 )( 1 16 )
e9 = ( 10 11 )( 8 9 )( 6 7 )( 5 12 )( 4 13 )( 3 16 )( 2 15 )( 1 14 )
e10 = ( 11 14 )( 9 13 )( 7 16 )( 5 15 )( 4 12 )( 3 10 )( 2 8 )( 1 6 )
e11 = ( 11 16 )( 9 15 )( 7 14 )( 5 13 )( 4 8 )( 3 6 )( 2 12 )( 1 10 )
e12 = ( 8 13 12 15 )( 6 16 10 14 )( 2 5 4 9 )( 1 11 3 7 )
e13 = ( 8 15 12 13 )( 6 14 10 16 )( 2 9 4 5 )( 1 7 3 11 )
e14 = ( 12 14 )( 10 13 )( 8 16 )( 6 15 )( 4 7 )( 3 5 )( 2 11 )( 1 9 )
e15 = ( 12 16 )( 10 15 )( 8 14 )( 6 13 )( 4 11 )( 3 9 )( 2 7 )( 1 5 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e4 }
S2 = { e0 e8 }
S3 = { e0 e9 }
S4 = { e0 e10 }
S5 = { e0 e11 }
S6 = { e0 e14 }
S7 = { e0 e15 }
S8 = { e0 e1 e4 e5 }
S9 = { e0 e2 e4 e6 }
S10 = { e0 e3 e4 e7 }
S11 = { e0 e4 e12 e13 }
S12 = { e0 e4 e8 e9 }
S13 = { e0 e4 e10 e11 }
S14 = { e0 e4 e14 e15 }
S15 = { e0 e1 e2 e4 e5 e6 e8 e9 }
S16 = { e0 e1 e3 e4 e5 e7 e10 e11 }
S17 = { e0 e2 e3 e4 e6 e7 e12 e13 }
S18 = { e0 e2 e4 e6 e10 e11 e14 e15 }
S19 = { e0 e3 e4 e7 e8 e9 e14 e15 }
S20 = { e0 e4 e8 e9 e10 e11 e12 e13 }
S21 = { e0 e1 e4 e5 e12 e13 e14 e15 }
S22 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }
```

UNKNOWN GROUP (dpd4d4)

The ELEMENTS are the following:

```
e0 is the identity element
e1 = ( 7 14 15 16 )( 6 11 12 13 )( 5 8 9 10 )( 1 2 3 4 )
e2 = ( 12 16 )( 9 13 )( 6 14 )( 5 11 )( 4 15 )( 3 10 )( 2 7 )( 1 8 )
e3 = ( 14 16 )( 11 13 )( 8 10 )( 7 15 )( 6 12 )( 5 9 )( 2 4 )( 1 3 )
e4 = ( 7 16 15 14 )( 6 13 12 11 )( 5 10 9 8 )( 1 4 3 2 )
e5 = ( 4 8 13 14 )( 3 15 12 9 )( 2 10 11 16 )( 1 7 6 5 )
e6 = ( 4 16 13 10 )( 3 5 12 7 )( 2 14 11 8 )( 1 9 6 15 )
e7 = ( 12 14 )( 9 11 )( 6 16 )( 5 13 )( 4 7 )( 3 8 )( 2 15 )( 1 10 )
e8 = ( 13 15 )( 10 12 )( 7 11 )( 6 8 )( 4 9 )( 3 16 )( 2 5 )( 1 14 )
e9 = ( 4 10 13 16 )( 3 7 12 5 )( 2 8 11 14 )( 1 15 6 9 )
e10 = ( 4 14 13 8 )( 3 9 12 15 )( 2 16 11 10 )( 1 5 6 7 )
e11 = ( 11 15 )( 8 12 )( 7 13 )( 6 10 )( 4 5 )( 3 14 )( 2 9 )( 1 16 )
e12 = ( 7 10 15 8 )( 5 16 9 14 )( 2 6 4 12 )( 1 13 3 11 )
e13 = ( 10 16 )( 9 15 )( 8 14 )( 5 7 )( 4 13 )( 3 12 )( 2 11 )( 1 6 )
e14 = ( 7 8 15 10 )( 5 14 9 16 )( 2 12 4 6 )( 1 11 3 13 )
e15 = ( 10 14 )( 8 16 )( 7 9 )( 5 15 )( 4 11 )( 3 6 )( 2 13 )( 1 12 )
```

The subgroups are the following:

```
S0 is the null subgroup
S1 = { e0 e2 }
S2 = { e0 e3 }
S3 = { e0 e7 }
S4 = { e0 e8 }
S5 = { e0 e11 }
S6 = { e0 e13 }
S7 = { e0 e15 }
S8 = { e0 e1 e3 e4 }
S9 = { e0 e5 e10 e13 }
S10 = { e0 e6 e9 e13 }
S11 = { e0 e3 e12 e14 }
S12 = { e0 e2 e3 e7 }
S13 = { e0 e2 e8 e13 }
S14 = { e0 e3 e8 e11 }
S15 = { e0 e2 e11 e15 }
S16 = { e0 e3 e13 e15 }
S17 = { e0 e3 e5 e6 e9 e10 e13 e15 }
S18 = { e0 e1 e3 e4 e12 e13 e14 e15 }
S19 = { e0 e2 e3 e7 e8 e11 e13 e15 }
S20 = { e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14
e15 }
```

APPENDIX B

ABSALGMM.C - The main menu for the Abstract Algebra Programming System.

ABSALGMP.C - This program is the print menu for the Abstract Algebra Programming System.

ABSALGMG.C - This program is the Group Generation menu for the Abstract Algebra Programming System.

ABSALGLDG.C - This program produces the generators for dihedral groups for the Abstract Algebra Programming System.

ABSALGCG.C - This program produces the generators for cyclic groups for the Abstract Algebra Programming System.

ABSALGAG.C - This program accepts permutations from a user for the abelian group generation for the Abstract Algebra Programming System.

ABSALGUG.C - This program accepts permutations from a user for the unknown group generation for the Abstract Algebra Programming System.

ABSALGIN.C - This program initializes the external arrays.

ABSALGCC.C - Canonical Formatter Program for the Abstract Algebra Programming System.

ABSALGPO.C - This program prints out the generating elements produced by the different generating programs - Groupd, Groupc, Groupa, and Groupg.

ABSALGIF.C - Internal Formatter for the Abstract Algebra Programming System.

ABSALGFR.C - File read data program for the Abstract Algebra Programming System.

ABSALGPS.C - Write data to either the printer or screen for the Abstract Algebra Programming System.

ABSALGOS.C - Orders the subgroup data from lowest order to highest order for the Abstract Algebra Programming System.

ABSALGDW.C - Dihedral write data program for the Abstract Algebra Programming System.

ABSALGCW.C - Cyclic write data program for the Abstract Algebra Programming System.

ABSALGAW.C - Abelian write data program for the Abstract Algebra Programming System.

ABSALGUW.C - Unknown write data program for the Abstract Algebra Programming System.

ABSALGGD.C - Group Generator Driver Program for the Abstract Algebra Programming System.

ABSALGAN.C - Basic Permutation Raised to a Power Program for the Abstract Algebra Programming System.

ABSALGSS.C - Subgroup Sorter Program for the Abstract Algebra Programming System.

ABSALGAS.C - Adds generated subgroups to the list of subgroups for the Abstract Algebra Programming System.

ABSALGKG.C - Check Subgroups for Existence Program for the Abstract Algebra Programming System.

ABSALGAB.C - Multiplier of two permutations for the Abstract Algebra Programming System.

ABSALGPM.C - (Disjoint) Basic Permutation Multiplier for the Abstract Algebra Programming System

ABSALGSI.C - Singleton Search and Deletion Program for the Abstract Algebra Program System.

ABSALGKE.C - Element Checker for the Abstract Algebra Programming System.

APPENDIX C

```
/*

ABSALGMM.C -  The main menu for the Abstract Algebra Programming System

This program presents the user with a menu.  The program then passes
control to that system.

This program defines the global variables  subsets, internal_form.

*/

char subsets[273][203], internal_form[155];

void group_generation();
void print_menu();

main()
        {

        char option[2], ch;

        do
                {
                option[0] = '\0';
                clrscr();
                printf("\n\n\n\n\n\n\n\n");
                printf("%53s", "PERMUTATION PROGRAM MAIN MENU");
                printf("\n\n\n");
                printf("%49s", "Option 1 - Group Generation");
                printf("\n\n");
                printf("%56s", "Option 2 - Print a Generated Group");
                printf("\n\n");
                printf("%45s", "Option 3 - Exit Program");
                printf("\n\n\n");
                printf("%37s", "Enter Option and Press Enter:  ");
                gets(option);

                switch(*option)
                        {

                        case '1':
                        group_generation();
                        break;

                        case '2':
                        print_menu();
                        break;

                        case '3':
                        option[0] = '\0';
```

```
                break;

                default:
                break;
                }
        }
while(*option);
clrscr();
}
```

```
/*

ABSALGMP.C -  This program is the print menu for the Abstract Algebra
programming system.

This presents the user with a Print Menu.  The user choses the type of
group they want printed.  The program then passes control to that system.

*/

extern char subsets[273][203], internal_form[155];

#include "string.h"

void file_read(char elements[][], char *, char *, char abelian[], char unknown[]);

void print_menu()
        {

        char choice[3], dihedral[4], cyclic[4];
        char elements[203][155], unknown[8], abelian[8];
        int i, j, n;

        /*

        The following while loop will execute as long as choice
        does not return a NULL value, i.e. '\0'

        */

        do
                {

                clrscr();
                printf("\n\n\n\n\n");
                printf("%40s", "PRINT MENU");
                printf("\n\n\n");
                printf("%50s", "Option 1 for DIHEDRAL GROUPS");
                printf("\n\n");
                printf("%48s", "Option 2 for CYCLIC GROUPS");
                printf("\n\n");
                printf("%49s", "Option 3 for ABELIAN GROUPS");
                printf("\n\n");
                printf("%49s", "Option 4 for UNKNOWN GROUPS");
                printf("\n\n");
                printf("%43s", "Option 5 EXIT PROGRAM");
                printf("\n\n\n");
                printf("%37s", "Enter Option and press ENTER:  ");
                gets(choice);
```

```
switch(*choice)     /* case statement */
        {

        case '1':
        do
                {
                clrscr();
                dihedral[0] = '\0';
                cyclic[0] = '\0';
                abelian[u] = '\0';
                unknown[0] = '\0';
                printf("\n\n\n\n\n\n\n");
                printf("%25s", "DIRECTIONS:");
                printf("\n\n\n");
                printf("%60s", "Enter the number of the Dihedral Group");
                printf("\n\n");
                printf("%51s", "after the 'D' and press ENTER");
                printf("\n\n");
                printf("%46s", "To exit just press ENTER");
                printf("\n\n");
                printf("%24s", "D ");
                gets(dihedral);
                if(*dihedral)
                        {
                        file_read(elements, dihedral, cyclic, abelian, unknown);
                        }
                }
        while(*dihedral);
        break;


        case '2':
        do
                {
                clrscr();
                dihedral[0] = '\0';
                cyclic[0] = '\0';
                abelian[0] = '\0';
                unknown[0] = '\0';
                printf("\n\n\n\n\n\n\n");
                printf("%25s", "DIRECTIONS:");
                printf("\n\n\n");
                printf("%58s", "Enter the number of the Cyclic Group");
                printf("\n\n");
                printf("%51s", "after the 'C' and press ENTER");
                printf("\n\n");
                printf("%46s", "To exit just press ENTER");
                printf("\n\n");
                printf("%24s", "C ");
                gets(cyclic);
```

```
        if(*cyclic)
                {
                file_read(elements, dihedral, cyclic, abelian, unknown);
                }
        }
while(*cyclic);
break;

case '3':
do
        {
        clrscr();
        dihedral[0] = '\0';
        cyclic[0] = '\0';
        abelian[0] = '\0';
        unknown[0] = '\0';
        printf("\n\n\n\n\n\n\n");
        printf("%25s", "DIRECTIONS:");
        printf("\n\n\n");
        printf("%60s", "Enter the first seven characters of");
        printf("\n\n");
        printf("%58s", "the Abelian group file name after");
        printf("\n\n");
        printf("%49s", "the '*' and press ENTER.");
        printf("\n\n");
        printf("%50s", "To exit just press ENTER.");
        printf("\n\n");
        printf("%27s", "* ");
        gets(abelian);
        if(*abelian)
                {
                file_read(elements, dihedral, cyclic, abelian, unknown);
                }
        }
while(*abelian);
break;

case '4':
do
        {
        clrscr();
        dihedral[0] = '\0';
        cyclic[0] = '\0';
        abelian[0] = '\0';
        unknown[0] = '\0';
        printf("\n\n\n\n\n\n\n");
        printf("%25s", "DIRECTIONS:");
        printf("\n\n\n");
        printf("%60s", "Enter the first seven characters of");
        printf("\n\n");
```

```
                        printf("%58s", "the Unknown group file name after");
                        printf("\n\n");
                        printf("%49s", "the '*' and press ENTER.");
                        printf("\n\n");
                        printf("%50s", "To exit just press ENTER.");
                        printf("\n\n");
                        printf("%27s", "* ");
                        gets(unknown);
                        if(*unknown)
                                {
                                file_read(elements, dihedral, cyclic, abelian, unknown);
                                }
                        }
                while(*unknown);
                break;

                case '5':
                printf("exit");
                choice[0] = '\0';
                break;

                default:
                break;

                }
        }
while(*choice);
}
```

```
/*

ABSALGMG.C -  This program is the Group Generation menu for the Abstract
Algebra programming system.

This presents the user with a Menu.  The user choses the type of group
they want generated.  The program then passes control to that system.

*/

extern char subsets[273][203], internal_form[155];

void dihedral_generation(char elements[][], char *);
void cyclic_generation(char elements[][], char *);
void unknown_generation(char elements[][]);
void abelian_generation(char elements[][]);

void group_generation()
        {

        char choice[3], dihedral[4], cyclic[4], abelian[4];
        char elements[203][155];
        int i, j, n;

        /*

        The following while loop will execute as long as choice
        does not return a NULL value, i.e. '\0'

        */

        do
                {

                clrscr();
                printf("\n\n\n\n\n");
                printf("%45s", "GROUP GENERATION MENU");
                printf("\n\n\n");
                printf("%50s", "Option 1 for DIHEDRAL GROUPS");
                printf("\n\n");
                printf("%48s", "Option 2 for CYCLIC GROUPS");
                printf("\n\n");
                printf("%49s", "Option 3 for ABELIAN GROUPS");
                printf("\n\n");
                printf("%49s", "Option 4 for UNKNOWN GROUPS");
                printf("\n\n");
                printf("%43s", "Option 5 EXIT PROGRAM");
                printf("\n\n\n");
                printf("%37s", "Enter Option and press ENTER:  ");
                gets(choice);
```

```
switch(*choice)     /* case statement */
        {

        case '1':
        do
                {
                clrscr();
                dihedral[0] = '\0';
                printf("\n\n\n\n\n\n\n");
                printf("%25s", "DIRECTIONS:");
                printf("\n\n\n");
                printf("%60s", "Enter the number of the Dihedral Group");
                printf("\n\n");
                printf("%51s", "after the 'D' and press ENTER");
                printf("\n\n");
                printf("%46s", "To exit just press ENTER");
                printf("\n\n");
                printf("%24s", "D ");
                gets(dihedral);
                if(*dihedral)
                        {
                        dihedral_generation(elements, dihedral);
                        }
                }
        while(*dihedral);
        break;


        case '2':
        do
                {
                clrscr();
                cyclic[0] = '\0';
                printf("\n\n\n\n\n\n\n");
                printf("%25s", "DIRECTIONS:");
                printf("\n\n\n");
                printf("%58s", "Enter the number of the Cyclic Group");
                printf("\n\n");
                printf("%51s", "after the 'C' and press ENTER");
                printf("\n\n");
                printf("%46s", "To exit just press ENTER");
                printf("\n\n");
                printf("%24s", "C ");
                gets(cyclic);
                if(*cyclic)
                        {
                        cyclic_generation(elements, cyclic);
                        }
                }
```

```
                    while(*cyclic);
                    break;

                    case '3':
                    abelian_generation(elements);
                    break;

                    case '4':
                    unknown_generation(elements);
                    break;

                    case '5':
                    choice[0] = '\0';
                    break;

                    default:
                    break;

                    }
            }
while(*choice);
}
```

```
/*

ABSALGDG.C  - This program produces the generators for dihedral groups for
Abstract Algebra Programming System

*/

extern char subsets[273][203], internal_form[155];

#include "stdlib.h"
div_t x;


void initialize(char elements[][]);
void permutation_out(char elements[][]);
void dihedral_file_write(char elements[][], char *);
void canonical_formatter();
void work_driver(char elements[][], int *);
void screen_printer_write(char elements[][], char *, char *, char *, char *);

void dihedral_generation(elements, dihedral)
char elements[203][155], dihedral[4];

        {
        char return_char[3], ddummy1[4], ddummy2[8], ddummy3[8];
        int i, j, n, m, dihedral_length, dihedral_value, order;
        int dmax = 101, dmin = 2, left_paren = -1;


        initialize(elements);
        order = 0;
        i = 0;  j = 1;
        dihedral_length = strlen(dihedral);
        if(dihedral_length != 0);
                {
                clrscr();
                sscanf(&dihedral[i], "%d", &dihedral_value);
                if(dihedral_value > dmin && dihedral_value < dmax)
                        {
                        elements[0][0] = dihedral_value;
                        elements[0][1] = '\0';
                        x = div(dihedral_value, 2);

                        /* building the first generator */

                        internal_form[i] = left_paren;
                        ++i;
                        while(i <= dihedral_value)
                                {
                                internal_form[i] = i;
```

```
        ++i;
        }
internal_form[i] = '\0';
strcpy(elements[j], internal_form);
internal_form[0] = '\0';
++j; i = 0;
internal_form[i] = left_paren;
++i;

/* building the second generator */

if(x.rem == 0)     /* dihedral_value is even */
        {
        m = 1;
        n = dihedral_value;
        while(m < n)
                {
                internal_form[i] = m;
                ++i; ++m;
                internal_form[i] = n;
                ++i; --n;
                internal_form[i] = left_paren;
                ++i;
                }
        --i;
        internal_form[i] = '\0';
        }
else              /* dihedral_value is odd */
        {
        m = 2;
        n = dihedral_value;
        while(m < n)
                {
                internal_form[i] = m;
                ++i; ++m;
                internal_form[i] = n;
                ++i; --n;
                internal_form[i] = left_paren;
                ++i;
                }
        --i;
        internal_form[i] = '\0';
        }
canonical_formatter();
strcpy(elements[j], internal_form);
internal_form[0] = '\0';
printf("\n\n\nThe DIHEDRAL GROUP GENERATORS are:\n\n");
permutation_out(elements);
order = 3;
work_driver(elements, &order);
```

```
                dihedral_file_write(elements, dihedral);
                clrscr();
                return_char[0] = '\0';
                printf("\n\n\nTo view the elements and subgroups\n\n");
                printf("Enter any alpha-key and press ENTER");
                printf("\n\nElse press ENTER to continue\n\n");
                gets(return_char);
                if(return_char[0] != '\0')
                        {
                        ddummy1[0] = '\0';
                        ddummy2[0] = '\0';
                        ddummy3[0] = '\0';
                        screen_printer_write(elements, dihedral, ddummy1, ddummy2, ddummy3);
                        }
                }
        else
                {
                printf("\n\n\nThe entered value of %d is either < 3 or > 100",
                                dihedral_value);
                printf("\n\nPress ENTER to continue");
                gets(return_char);
                }
        }
}
```

```
/*

ABSALGCG.C  - This program produces the generators for cyclic groups for
Abstract Algebra Programming System

*/

extern char subsets[273][203], internal_form[155];



void initialize(char elements[][]);
void permutation_out(char elements[][]);
void cyclic_file_write(char elements[][], char *);
void canonical_formatter();
void work_driver(char elements[][], int *);
void screen_printer_write(char elements[][], char *, char *, char *, char *);

void cyclic_generation(elements, cyclic)
char elements[203][155], cyclic[4];

        {
        char return_char[2], cdummy1[4], cdummy2[8], cdummy3[8];
        int i, j, n, m, cyclic_length, cyclic_value, order;
        int dmax = 101, dmin = 2, left_paren = -1;


        initialize(elements);
        order = 0;
        i = 0;  j = 1;
        cyclic_length = strlen(cyclic);
        if(cyclic_length != 0);
                {
                clrscr();
                sscanf(&cyclic[i], "%d", &cyclic_value);
                if(cyclic_value > dmin && cyclic_value < dmax)
                        {
                        elements[0][0] = cyclic_value;
                        elements[0][1] = '\0';

                        /* building the only generator */

                        internal_form[i] = left_paren;
                        ++i;
                        while(i <= cyclic_value)
                                {
                                internal_form[i] = i;
                                ++i;
                                }
                        internal_form[i] = '\0';
```

```
                strcpy(elements[j], internal_form);
                internal_form[0] = '\0';
                i = 0;
                printf("\n\n\nThe CYCLIC GROUP GENERATOR is:\n\n");
                permutation_out(elements);
                order = 2;
                work_driver(elements, &order);
                cyclic_file_write(elements, cyclic);
                clrscr();
                return_char[0] = '\0';
                printf("\n\n\nTo view the elements and subgroups\n\n");
                printf("Enter any alpha-key and press ENTER");
                printf("\n\nElse press ENTER to continue\n\n");
                gets(return_char);
                if(return_char[0] != '\0')
                        {
                        cdummy1[0] = '\0';
                        cdummy2[0] = '\0';
                        cdummy3[0] = '\0';
                        screen_printer_write(elements, cdummy1, cyclic, cdummy2, cdummy3);
                        }
                }
        else
                {
                printf("\n\n\nThe entered value of %d is either < 3 or > 100",
                                cyclic_value);
                printf("\n\nPress ENTER to continue");
                gets(return_char);
                }
        }
}
```

```
/*

ABSALGAG.C - This program accepts permutations from a user for the
abelian group generation for the Abstract Algebra Programming System.

*/

extern char subsets[273][203], internal_form[155];

void initialize(char elements[][]);
int internal_format(char permu_in[]);
void permutation_out(char elements[][]);
void check_elements(char elements[][], int *, int *);
void work_driver(char elements[][], int *);
void abelian_file_write(char elements[][], char abelian_file_name[]);
void screen_printer_write(char elements[][], char *, char *, char *, char *);

void abelian_generation(elements)
char elements[203][155];

        {
        char permu_in[81], return_char[2], abelian_file_name[8];
        char adummy1[4], adummy2[4], adummy3[8];
        int i, good_bad, order, new_order, element_number;
        int number_of_elements_input;

        order = 1; new_order = 1;
        number_of_elements_input = 7;
        initialize(elements);
        clrscr();
        printf("\n\n");
        printf("%25s", "DIRECTIONS:");
        printf("\n\n\n");
        printf("%67s", "When prompted with a '*', enter a permutation");
        printf("\n");
        printf("%65s", "in cycle notation.  Use '()' as delimiters.");
        printf("\n");
        printf("%65s", "Enter only interger values (i) in the range");
        printf("\n");
        printf("%65s", "0 < i < 128.  Place a space between each i,");
        printf("\n");
        printf("%61s", "For Example: (1 34 5)(23 127 2).");
        printf("\n\n");
        printf("%67s", "A PERMUTATION MAY ONLY BE 80 CHARACTERS LONG.");
        printf("\n");
        printf("%69s", "NOTE: A GROUP MAY HAVE A TOTAL OF 128 ELEMENTS.");
        printf("\n\n");
        printf("%67s", "Press ENTER when the permutation is complete.");
        printf("\n");
        printf("%65s", "To end entry press ENTER with a null input.");
```

```
        printf("\n\n");
        putch('*');
        gets(permu_in);
        i = 0;
        while(*permu_in)
                {
                if(number_of_elements_input != 0)
                        {
                        if(permu_in[i] == '(')
                                {
                                good_bad = internal_format(permu_in);
                                if(good_bad)
                                        {
                                        check_elements(elements, &new_order, &element_number);
                                        if(order != new_order)
                                                {
                                                --number_of_elements_input;
                                                order = new_order;
                                                clrscr();
                                                }
                                        else
                                                {
                                                printf("\n\n");
                                                printf("%69s", "The above permutation has already been en
tered.");
                                                printf("\n\n");
                                                printf("%62s", "Press ENTER to reenter your permutation."
);
                                                gets(return_char);
                                                }
                                        }
                                }
                        else
                                {
                                printf("\n\n");
                                printf("%63s", "The first element of entry must be a '('.");
                                printf("\n\n");
                                printf("%62s", "Press ENTER to reenter your permutation.");
                                gets(return_char);
                                }
                        clrscr();
                        printf("This is the canonical form ");
                        printf("of the permutations you entered\n");
                        permutation_out(elements);
                        if(number_of_elements_input != 0)
                                {
                                printf("\n");
                                printf("%25s", "DIRECTIONS:");
                                printf("\n\n");
                                printf("%67s", "When prompted with a '*', enter a permutation");
```

```
                        printf("\n");
                        printf("%65s", "in cycle notation.  Use '()' as delimiters.");
                        printf("\n");
                        printf("%65s", "Enter only interger values (i) in the range");
                        printf("\n");
                        printf("%65s", "0 < i < 128.  Place a space between each i,");
                        printf("\n");
                        printf("%61s", "For Example: (1 34 5)(23 127 2).");
                        printf("\n\n");
                        printf("%67s", "A PERMUTATION MAY ONLY BE 80 CHARACTERS LONG.");
                        printf("\n");
                        printf("%69s", "NOTE: A GROUP MAY HAVE A TOTAL OF 128 ELEMENTS.");
                        printf("\n\n");
                        printf("%67s", "Press ENTER when the permutation is complete.");
                        printf("\n");
                        printf("%65s", "To end entry press ENTER with a null input.");
                        printf("\n\n");
                        putch('*');
                        gets(permu_in);
                        }
                else
                        {
                        printf("\n\n");
                        printf("%65s", "You have entered the max number of permutations.");
                        printf("\n\n");
                        printf("%53s", "Press ENTER to generated the group. ");
                        gets(return_char);
                        permu_in[0] = '\0';
                        }
                }
        else
                {
                printf("\n\n");
                printf("%67s", "You may only enter a total of 7 permutations.");
                printf("\n\n");
                printf("%62s", "Press ENTER for the program to continue.");
                gets(return_char);
                permu_in[0] = '\0';
                }
        }
if(number_of_elements_input < 7)
        {
        work_driver(elements, &order);
        clrscr();
        printf("\n\n\n\n\n");
        printf("At the *, please enter up to seven (7) characters for a filename.\n\n");
        printf("NOTE:  DO NOT ENTER THE PERIOD AND THREE CHARACTER EXTENSION.\n\n");
        printf("%25s", "* ");
        gets(abelian_file_name);
        abelian_file_write(elements, abelian_file_name);
```

```
        clrscr();
        return_char[0] = '\0';
        printf("\n\n\nTo view the elements and subgroups\n\n");
        printf("Enter any alpha-key and press ENTER");
        printf("\n\nElse press ENTER to continue\n\n");
        gets(return_char);
        if(return_char[0] != '\0')
                {
                adummy1[0] = '\0';
                adummy2[0] = '\0';
                adummy3[0] = '\0';
                screen_printer_write(elements, adummy1, adummy2, abelian_file_name, adummy3);
                }
        }
}
```

```
/*

ABSALGUG.C - This program accepts permutations from a user for the
unknown group generation for the Abstract Algebra Programming System.

*/

extern char subsets[273][203], internal_form[155];

void initialize(char elements[][]);
int internal_format(char permu_in[]);
void permutation_out(char elements[][]);
void check_elements(char elements[][], int *, int *);
void work_driver(char elements[][], int *);
void unknown_file_write(char elements[][], char unknown_file_name[]);
void screen_printer_write(char elements[][], char *, char *, char *, char *);

void unknown_generation(elements)
char elements[203][155];

        {
        char permu_in[81], return_char[2], unknown_file_name[8];
        char udummy1[4], udummy2[4], udummy3[8];
        int i, good_bad, order, new_order, element_number;
        int number_of_elements_input;

        order = 1; new_order = 1;
        number_of_elements_input = 7;
        initialize(elements);
        clrscr();
        printf("\n\n");
        printf("%25s", "DIRECTIONS:");
        printf("\n\n\n");
        printf("%67s", "When prompted with a '*', enter a permutation");
        printf("\n");
        printf("%65s", "in cycle notation.  Use '()' as delimiters.");
        printf("\n");
        printf("%65s", "Enter only interger values (i) in the range");
        printf("\n");
        printf("%65s", "0 < i < 128.  Place a space between each i,");
        printf("\n");
        printf("%61s", "For Example: (1 34 5)(23 127 2).");
        printf("\n\n");
        printf("%67s", "A PERMUTATION MAY ONLY BE 80 CHARACTERS LONG.");
        printf("\n");
        printf("%69s", "NOTE: A GROUP MAY HAVE A TOTAL OF 128 ELEMENTS.");
        printf("\n\n");
        printf("%67s", "Press ENTER when the permutation is complete.");
        printf("\n");
        printf("%65s", "To end entry press ENTER with a null input.");
```

```
        printf("\n\n");
        putch('*');
        gets(permu_in);
        i = 0;
        while(*permu_in)
                {
                if(number_of_elements_input != 0)
                        {
                        if(permu_in[i] == '(')
                                {
                                good_bad = internal_format(permu_in);
                                if(good_bad)
                                        {
                                        check_elements(elements, &new_order, &element_number);
                                        if(order != new_order)
                                                {
                                                --number_of_elements_input;
                                                order = new_order;
                                                clrscr();
                                                }
                                        else
                                                {
                                                printf("\n\n");
                                                printf("%69s", "The above permutation has already been en
tered.");
                                                printf("\n\n");
                                                printf("%62s", "Press ENTER to reenter your permutation."
);
                                                gets(return_char);
                                                }
                                        }
                                }
                        else
                                {
                                printf("\n\n");
                                printf("%63s", "The first element of entry must be a '('.");
                                printf("\n\n");
                                printf("%62s", "Press ENTER to reenter your permutation.");
                                gets(return_char);
                                }
                        clrscr();
                        printf("This is the canonical form ");
                        printf("of the permutations you entered\n");
                        permutation_out(elements);
                        if(number_of_elements_input != 0)
                                {
                                printf("\n");
                                printf("%25s", "DIRECTIONS:");
                                printf("\n\n");
                                printf("%67s", "When prompted with a '*', enter a permutation");
```

```
                        printf("\n");
                        printf("%65s", "in cycle notation.  Use '()' as delimiters.");
                        printf("\n");
                        printf("%65s", "Enter only interger values (i) in the range");
                        printf("\n");
                        printf("%65s", "0 < i < 128.  Place a space between each i,");
                        printf("\n");
                        printf("%61s", "For Example: (1 34 5)(23 127 2).");
                        printf("\n\n");
                        printf("%67s", "A PERMUTATION MAY ONLY BE 80 CHARACTERS LONG.");
                        printf("\n");
                        printf("%69s", "NOTE: A GROUP MAY HAVE A TOTAL OF 128 ELEMENTS.");
                        printf("\n\n");
                        printf("%67s", "Press ENTER when the permutation is complete.");
                        printf("\n");
                        printf("%65s", "To end entry press ENTER with a null input.");
                        printf("\n\n");
                        putch('*');
                        gets(permu_in);
                        }
                else
                        {
                        printf("\n\n");
                        printf("%65s", "You have entered the max number of permutations.");
                        printf("\n\n");
                        printf("%53s", "Press ENTER to generated the group. ");
                        gets(return_char);
                        permu_in[0] = '\0';
                        }
                }
        else
                {
                printf("\n\n");
                printf("%67s", "You may only enter a total of 7 permutations.");
                printf("\n\n");
                printf("%62s", "Press ENTER for the program to continue.");
                gets(return_char);
                permu_in[0] = '\0';
                }
        }
if(number_of_elements_input < 7)
        {
        work_driver(elements, &order);
        clrscr();
        printf("\n\n\n\n\n");
        printf("At the *, please enter up to seven (7) characters for a filename.\n\n");
        printf("NOTE:  DO NOT ENTER THE PERIOD AND THREE CHARACTER EXTENSION.\n\n");
        printf("%25s", "* ");
        gets(unknown_file_name);
        unknown_file_write(elements, unknown_file_name);
```

```
        clrscr();
        return_char[0] = '\0';
        printf("\n\n\nTo view the elements and subgroups\n\n");
        printf("Enter any alpha-key and press ENTER");
        printf("\n\nElse press ENTER to continue\n\n");
        gets(return_char);
        if(return_char[0] != '\0')
                {
                udummy1[0] = '\0';
                udummy2[0] = '\0';
                udummy3[0] = '\0';
                screen_printer_write(elements, udummy1, udummy2, udummy3, unknown_file_name);
                }
        }
}
```

```
/*

ABSALGIN.C - This program initializes the external arrays.

*/

extern char subsets[273][203], internal_form[155];

void initialize(elements)
char elements[203][155];
        {

        int i;

        i = 0;
        internal_form[i] = '\0';
        while(i < 203)
                {
                elements[i][0] = '\0';
                ++i;
                }
        i = 0;
        while(i < 273)
                {
                subsets[i][0] = '\0';
                ++i;
                }
        }
```

```
/*

ABSALGCC.C - Canonical Formatter Program for the Abstract Algebra Programming
System

This function puts into canonical form any permutation passed to
it.  It breaks down into three separate parts:

Part 1:  Separates the given permutation into individual cycles.

Part 2:  Works on each individual cycle, placing the smallest
                value first.

Part 3:  Combines the individual cycles into one permutation.
                Placing the cycle with the largest first value first
                then each cycle in decreasing order by the value of
                the first element.

This function is called after every permutation multiplication.

*/

extern char subsets[273][203], internal_form[155];

void  canonical_formatter()


        {
        char temp_form[155], canonical_form[53][155], first_element;
        int i, j, m, n, l, depth_of_canonical_form;
        int length_of_internal_form, hold, once_through;

        j = 0; n = 0; m = 0; temp_form[0] = '\0';            /* start Part 1 */
        length_of_internal_form = strlen(internal_form);
        while(j < length_of_internal_form)
                {
                canonical_form[m][n] = internal_form[j];
                ++j;
                while(internal_form[j] != internal_form[0] && internal_form[j] != '\0')
                        {
                        ++n;
                        canonical_form[m][n] = internal_form[j];
                        ++j;
                        }
                ++n;
                canonical_form[m][n] = '\0';
                ++m; n = 0;
                }
        depth_of_canonical_form = m - 1;                     /* end of Part 1 */
```

```
m = 0; n = 1;                                        /* start Part 2 */
while(m <= depth_of_canonical_form)
        {
        first_element = canonical_form[m][n];
        hold = n; ++n;
        while(canonical_form[m][n] != '\0')
                {
                if(first_element > canonical_form[m][n])
                        {
                        first_element = canonical_form[m][n];
                        hold = n;
                        }
                ++n;
                }
        i= 0;
        temp_form[i] = internal_form[0];
        ++i; n = hold;
        temp_form[i] = canonical_form[m][n];
        ++n; ++i;
        while(canonical_form[m][n] != canonical_form[m][hold])
                {
                if(canonical_form[m][n] != '\0')
                        {
                        temp_form[i] = canonical_form[m][n];
                        ++i; ++n;
                        }
                else
                        {
                        n = 1;
                        }
                }
        temp_form[i] = '\0';
        strcpy(canonical_form[m], temp_form);
        ++m; n = 1;
        }                                                /* end of Part 2 */


n = 1; once_through = 1; l = 0;                      /* start Part 3 */
while(depth_of_canonical_form >= 0)
        {
        m = 0;
        first_element = canonical_form[m][n];
        hold = m; ++m;
        while(m <= depth_of_canonical_form)
                {
                if(first_element < canonical_form[m][n])
                        {
                        first_element = canonical_form[m][n];
                        hold = m;
```

```
                }
            ++m;
            }
        i = hold; m = i + 1; j = 0;
        if(once_through == 1)
            {
            while(canonical_form[i][j] != '\0')
                {
                temp_form[l] = canonical_form[i][j];
                ++l; ++j;
                }
            --once_through;
            }
        else
            {
            while(canonical_form[i][j] != '\0')
                {
                temp_form[l] = canonical_form[i][j];
                ++l; ++j;
                }
            }
        while(m <= depth_of_canonical_form)
            {
            strcpy(canonical_form[i], canonical_form[m]);
            ++i; ++m;
            }
        --depth_of_canonical_form;
        }                                           /* end of Part 3 */

    temp_form[l] = '\0';
    strcpy(internal_form, temp_form);
    }
```

```
/*

ABSALGPO.C - This program printsout the generating elements produced by
the different generating programs - Groupd, Groupc, Groupa, and Groupg

*/

extern char subsets[273][203], internal_form[155];

void permutation_out(elements)
char elements[203][155];
        {

        int i, j;

        i = 1;
        j = 1;
        while(elements[j][0] != '\0')
                {
                strcpy(internal_form, elements[j]);
                printf("( ");
                while(internal_form[i] != '\0')
                        {
                        if(internal_form[i] == internal_form[0])
                                {
                                printf(")( ");
                                ++i;
                                }
                        else
                                {
                                printf("%d ", internal_form[i]);
                                ++i;
                                }
                        }
                printf(")\n");
                ++j; i = 1;
                }
        }
```

```
/*
ABSALGIF.C - Internal Formatter for the Abstract Algebra Programming
System.

This function accepts a permutation from the Driver and converts it into the
internal format that the rest of the program can then use.  This routine
calls SINGLETON SEARCH, CANONICAL FORMATTER, and DISJOINT to perform
its tasks.

*/

extern char subsets[273][203], internal_form[155];

#include "ctype.h"

void singleton_search();
void disjoint(char if_work_form[]);
void canonical_formatter();

int internal_format(permu_in)
char permu_in[80];

        {
        char return_char[2], if_work_form[311];
        int i, j, int_val, n, digit_test, space_test, good_bad;

        i = 0; j = 0; good_bad = 1;
        n = strlen(permu_in);
        if_work_form[0] = '\0';
        while(i < n)
                {
                digit_test = isdigit(permu_in[i]);
                if(digit_test)
                        {
                        sscanf(&permu_in[i], "%d", &int_val);
                        if(int_val > 127)
                                {
                                clrscr();
                                printf("\n\n\n%d is greater than 127, ", int_val);
                                printf("the maximum allowed value.\n\n");
                                printf("Please press ENTER to continue ");
                                gets(return_char);
                                good_bad = 0;
                                break;
                                }
                        else
                                {
                                if(int_val == 0)
                                        {
                                        clrscr();
```

```
                                printf("\n\n\n0 by itself is not an allowed input value.\n\n");
                                printf("Please press ENTER to continue ");
                                gets(return_char);
                                good_bad = 0;
                                break;
                                }
                        }
                internal_form[j] = int_val;
                ++i; ++j;
                digit_test = isdigit(permu_in[i]);
                while(digit_test)
                        {
                        ++i;
                        digit_test = isdigit(permu_in[i]);
                        }
                }
        else
                {
                space_test = isspace(permu_in[i]);
                if(space_test)
                        ++i;
                else
                        {
                        if(permu_in[i] == '(')
                                {
                                internal_form[j] = -1;
                                ++i; ++j;
                                }
                        else
                                {
                                if(permu_in[i] == ')')
                                        ++i;
                                else
                                        {
                                        clrscr();
                                        printf("\n\n\nIntegers, spaces, and '()' are the ");
                                        printf("only legal characters.\n\n");
                                        printf("Please press ENTER to continue ");
                                        gets(return_char);
                                        good_bad = 0;
                                        break;
                                        }
                                }
                        }
                }
        }
internal_form[j] = '\0';
if(good_bad)
        {
        singleton_search();
```

```
        n = strlen(internal_form);
        if(n)
                {
                strcpy(if_work_form, internal_form);
                disjoint(if_work_form);
                j = 0;
                while(if_work_form[j] != '\0')
                        {
                        internal_form[j] = if_work_form[j];
                        ++j;
                        }
                internal_form[j] = '\0';
                singleton_search();
                n = strlen(internal_form);
                if(n)
                        canonical_formatter();
                else
                        {
                        clrscr();
                        printf("\n\n\nThe previous permutation equated to all singletons.");
                        printf("Please press ENTER to continue ");
                        gets(return_char);
                        good_bad = 0;
                        }
                }
        else
                {
                clrscr();
                printf("\n\n\nThe previous permutation equated to all singletons.\n\n");
                printf("Please press ENTER to continue ");
                gets(return_char);
                good_bad = 0;
                }
        }
return(good_bad);
}
```

```
/*

ABSALGFR.C -  File read data program for the Abstract Algebra Programming
System

This program reads the data created and written to a file by the
different Group Generation Programs.

*/

extern char subsets[273][203], internal_form[155];

#include "stdio.h"

void screen_printer_write(char elements[][], char *, char *, char *, char *);

void file_read(elements, dihedral, cyclic, abelian, unknown)
char elements[203][155], dihedral[4], cyclic[4], abelian[8], unknown[8];

        {
        int i, j, int_value, efile_value, sfile_value, dihedral_value, cyclic_value;
        int char_count, line_count, high_count, length_count, count_array[273];
        char element_form[155], subgroup_form[203], output_select[3];
        char element_file[15], subgroup_file[15], return_char[2];
        char subgroup_sort[273][203];
        FILE *ep;
        FILE *sp;
        FILE *prn_ptr;

        /* the following area is the filename construction area */

        i = 0;
        if(dihedral[0] != '\0')
                {
                element_file[0] = 'D';
                element_file[1] = 'E';
                element_file[2] = '\0';
                subgroup_file[0] = 'D';
                subgroup_file[1] = 'S';
                subgroup_file[2] = '\0';
                strcat(element_file, dihedral);
                strcat(subgroup_file, dihedral);
                }
        else
                {
                if(cyclic[0] != '\0')
                        {
                        element_file[0] = 'C';
                        element_file[1] = 'E';
                        element_file[2] = '\0';
```

```
                subgroup_file[0] = 'C';
                subgroup_file[1] = 'S';
                subgroup_file[2] = '\0';
                strcat(element_file, cyclic);
                strcat(subgroup_file, cyclic);
                }
        else
                {
                if(abelian[0] != '\0')
                        {
                        element_file[0] = '\0';
                        subgroup_file[0] = '\0';
                        strcpy(element_file, abelian);
                        strcpy(subgroup_file, abelian);
                        strcat(element_file, "E");
                        strcat(subgroup_file, "S");
                        }
                else
                        {
                        element_file[0] = '\0';
                        subgroup_file[0] = '\0';
                        strcpy(element_file, unknown);
                        strcpy(subgroup_file, unknown);
                        strcat(element_file, "E");
                        strcat(subgroup_file, "S");
                        }
                }
        }
strcat(element_file, ".DAT");
strcat(subgroup_file, ".DAT");

/* the following area opens the files */

if((ep = fopen(element_file, "rt")) == NULL)
        {
        printf("cannot open element file\n");
        exit(1);
        }
if((sp = fopen(subgroup_file, "rt")) == NULL)
        {
        printf("cannot open subgroup file\n");
        exit(1);
        }

/* the following area reads the element file */

i = 0; j = 1;
efile_value = fread(element_form, 1, 155, ep);
while(efile_value)
        {
```

```
        while(element_form[i] != '\0')
                {
                int_value = element_form[i];
                if(int_value == -2)
                        {
                        elements[j][i] = 13;
                        ++i;
                        }
                else
                        {
                        if(int_value == -3)
                                {
                                elements[j][i] = 26;
                                ++i;
                                }
                        else
                                {
                                elements[j][i] = element_form[i];
                                ++i;
                                }
                        }
                }
        elements[j][i] = '\0';
        efile_value = fread(element_form, 1, 155, ep);
        ++j; i = 0;
        }
fclose(ep);
elements[j][0] = '\0';

/* the following area reads the subgroup file */

i = 0; j = 1;
sfile_value = fread(subgroup_form, 1, 203, sp);
while(sfile_value)
        {
        while(subgroup_form[i] != '\0')
                {
                int_value = subgroup_form[i];
                if(int_value == -2)
                        {
                        subsets[j][i] = 13;
                        ++i;
                        }
                else
                        {
                        if(int_value == -3)
                                {
                                subsets[j][i] = 26;
                                ++i;
                                }
```

```
                        else
                                {
                                if(int_value == -4)
                                        {
                                        subsets[j][i] = 115;
                                        ++i;
                                        }
                                else
                                        {
                                        subsets[j][i] = subgroup_form[i];
                                        ++i;
                                        }
                                }
                        }
                }
        subsets[j][i] = '\0';
        sfile_value = fread(subgroup_form, 1, 203, sp);
        ++j; i = 0;
        }
fclose(sp);
subsets[j][0] = '\0';

screen_printer_write(elements, dihedral, cyclic, abelian, unknown);

}
```

```
/*

ABSALGPS.C -  Write data to either the printer or screen for the Abstract
Algebra Programming System

This program gets the data created and/or written to a file by the
different Group Generation Programs and wrtes to the appropriate output.

*/

extern char subsets[273][203], internal_form[155];

#include "stdio.h"

void order_subgroups(int count_array[], int *);

void screen_printer_write(elements, dihedral, cyclic, abelian, unknown)
char elements[203][155], dihedral[4], cyclic[4], abelian[8], unknown[8];

        {
        int i, j, int_value, dihedral_value, cyclic_value, count, subgroup_number;
        int char_count, line_count, high_count, count_array[273];
        char output_select[3], return_char[2];
        FILE *prn_ptr;


        high_count = 0;
        order_subgroups(count_array, &high_count);
        count = 1;

        /* the following area determines if output
        goes either to screen or to printer */

        clrscr();
        printf("\n\n\n\n\n\n\n");
        printf("%50s", "SELECT OUTPUT METHOD");
        printf("\n\n\n");
        printf("%45s", "p for PRINTER");
        printf("\n\n");
        printf("%44s", "s for SCREEN");
        printf("\n\n\n");
        printf("%40s", "Enter choice and press ENTER: ");
        gets(output_select);

        /* the following area is for screen output */

        if(output_select[0] == 's')
                {
                clrscr();
                printf("The elements are the following:\n\n");
```

```
printf("e0 is the identity element\n");
j = 1; i = 1;
while(elements[j][0] != '\0')
        {
        strcpy(internal_form, elements[j]);
        printf("e%d = ( ", j);
        while(internal_form[i] != '\0')
                {
                if(internal_form[i] == internal_form[0])
                        {
                        printf(")( ");
                        ++i;
                        }
                else
                        {
                        printf("%d ", internal_form[i]);
                        ++i;
                        }
                }
        printf(")\n");
        ++j; i = 1;
        }
printf("\nThe subgroups are the following:\n\n");
printf("S0 is the null subgroup\n");
j = 1;  subgroup_number = 1;
while(count <= high_count)
        {
        while(subsets[j][0] != '\0')
                {
                if(count == count_array[j])
                        {
                        i = 0;
                        printf("S%d = { e0 ", subgroup_number);
                        while(subsets[j][i] != '\0')
                                {
                                printf("e%d ", subsets[j][i]);
                                ++i;
                                }
                        printf("}\n");
                        ++j; ++subgroup_number;
                        }
                else
                        {
                        ++j;
                        }
                }
        j = 1;
        ++count;
        }
printf("\n\nPress ENTER to continue: ");
```

```
        gets(return_char);
        }

/* the following area is for printer output */

if(output_select[0] == 'p')
        {
        i = 0;
        clrscr();
        printf("\n\n\n\n\n\n\n\n");
        printf("%44s", "PRINTING");
        prn_ptr = fopen("PRN", "wt");
        if(dihedral[0] != '\0')
                {
                sscanf(&dihedral[i], "%d", &dihedral_value);
                fprintf(prn_ptr, "\n\nDIHEDRAL GROUP D(%d)\n\n\n", dihedral_value);
                }
        else
                {
                if(cyclic[0] != '\0')
                        {
                        sscanf(&cyclic[i], "%d", &cyclic_value);
                        fprintf(prn_ptr, "\n\nCYCLIC GROUP C(%d)\n\n\n", cyclic_value);
                        }
                else
                        {
                        if(abelian[0] != '\0')
                                {
                                fprintf(prn_ptr, "\n\nABELIAN GROUP (%s)\n\n\n", abelian);
                                }
                        else
                                {
                                fprintf(prn_ptr, "\n\nUNKNOWN GROUP (%s)\n\n\n", unknown);
                                }
                        }
                }
        fputs("The ELEMENTS are the following:\n\n", prn_ptr);
        fputs("e0 is the identity element\n", prn_ptr);
        j = 1; line_count = 9;
        while(elements[j][0] != '\0')
                {
                if(line_count > 60)
                        {
                        fprintf(prn_ptr, "\n\n\n\n\r\n\n\n");
                        line_count = 3;
                        }
                fprintf(prn_ptr, "e%d = ( ", j);
                i = 1; char_count = 9;
                while(elements[j][i] != '\0')
                        {
```

```
                if(char_count > 80)
                        {
                        fprintf(prn_ptr, "\n");
                        char_count = 0;
                        ++line_count;
                        }
                if(line_count > 60)
                        {
                        fprintf(prn_ptr, "\n\n\n\n\n\n\n\n\n");
                        line_count = 3;
                        }
                if(elements[j][i] == elements[j][0])
                        {
                        fprintf(prn_ptr, ")( ");
                        ++i; char_count = char_count + 3;
                        }
                else
                        {
                        fprintf(prn_ptr, "%d ", elements[j][i]);
                        ++i; char_count = char_count + 3;
                        }
                }
        fprintf(prn_ptr, ")\n");
        ++j;
        ++line_count;
        }
if(line_count <= 61)
        {
        while(line_count < 70)
                {
                fprintf(prn_ptr, "\n");
                ++line_count;
                }
        }
fputs("The subgroups are the following:\n\n", prn_ptr);
fputs("S0 is the null subgroup\n", prn_ptr);
j = 1; line_count = 7; subgroup_number = 1;
while(count <= high_count)
        {
        while(subsets[j][0] != '\0')
                {
                if(count == count_array[j])
                        {
                        i = 0;
                        if(line_count > 60)
                                {
                                fprintf(prn_ptr, "\n\n\n\n\n\n\n\n\n");
                                line_count = 3;
                                }
                        fprintf(prn_ptr, "S%d = { e0 ", subgroup_number);
```

```
                                        char_count = 12;
                                        while(subsets[j][i] != '\0')
                                                {
                                                if(char_count > 80)
                                                        {
                                                        fprintf(prn_ptr, "\n");
                                                        char_count = 0;
                                                        ++line_count;
                                                        }
                                                if(line_count > 60)
                                                        {
                                                        fprintf(prn_ptr, "\n\n\n\n\n\n\n\n");
                                                        line_count = 3;
                                                        }
                                                fprintf(prn_ptr, "e%d ", subsets[j][i]);
                                                ++i; char_count = char_count + 5;
                                                }
                                        fprintf(prn_ptr, "}\n");
                                        ++j; ++line_count; ++subgroup_number;
                                        }
                                else
                                        {
                                        ++j;
                                        }
                                }
                        j = 1;
                        ++count;
                        }
                if(line_count <= 61)
                        {
                        while(line_count < 67)
                                {
                                fprintf(prn_ptr, "\n");
                                ++line_count;
                                }
                        }
                fclose(prn_ptr);
                printf("\n\nPress ENTER to continue: ");
                gets(return_char);
                }
        }
```

```
/*

ABSALGOS.C -  Orders the subgroup data from lowest order to highest order
for Abstract Algebra Programming System

*/

extern char subsets[273][203], internal_form[155];


void order_subgroups(count_array, high_count)
int count_array[273], *high_count;

        {
        int i, j;
        int length_count;


        /* the following area sorts the subgroup file */

        i = 0;
        while(i < 273)
                {
                count_array[i] = 0;
                ++i;
                }
        length_count = 0;
        j = 1;
        while(subsets[j][0] != '\0')
                {
                length_count = strlen(subsets[j]);
                if((length_count > *high_count)
                        {
                        *high_count = length_count;
                        }
                count_array[j] = length_count;
                ++j;
                }
        }
```

```
/*

ABSALGDW.C -  Dihedral write data program for the Abstract Algebra Programming
System

This program writes the data created by the Dihedral Group Generation Program.

*/

extern char subsets[273][203], internal_form[155];

#include "stdio.h"


void dihedral_file_write(elements, dihedral)
char elements[203][155], dihedral[4];

        {
        int i, j, int_value, efile_value, sfile_value;
        char element_form[155], subgroup_form[203];
        char dihedral_element[12], dihedral_subgroup[12];
        FILE *ep;
        FILE *sp;

        /* the following area is the filename construction area */

        i = 0;
        dihedral_element[0] = 'D';
        dihedral_element[1] = 'E';
        dihedral_element[2] = '\0';
        dihedral_subgroup[0] = 'D';
        dihedral_subgroup[1] = 'S';
        dihedral_subgroup[2] = '\0';
        strcat(dihedral_element, dihedral);
        strcat(dihedral_subgroup, dihedral);
        strcat(dihedral_element, ".DAT");
        strcat(dihedral_subgroup, ".DAT");

        /* the following area opens the files */

        if((ep = fopen(dihedral_element, "wt")) == NULL)
                {
                printf("cannot open element file\n");
                exit(1);
                }
        if((sp = fopen(dihedral_subgroup, "wt")) == NULL)
                {
                printf("cannot open subgroup file\n");
                exit(1);
                }
```

```
/* the following area writes the element file */

i = 0; j = 1;
while(elements[j][0] != '\0')
        {
        while(i < 155)
                {
                element_form[i] = '\0';
                ++i;
                }
        i = 0;
        while(elements[j][i] != '\0')
                {
                int_value = elements[j][i];
                if(int_value == 13)
                        {
                        element_form[i] = -2;
                        ++i;
                        }
                else
                        {
                        if(int_value == 26)
                                {
                                element_form[i] = -3;
                                ++i;
                                }
                        else
                                {
                                element_form[i] = elements[j][i];
                                ++i;
                                }
                        }
                }
        element_form[i] = '\0';
        efile_value = fwrite(element_form, 155, 1, ep);
        ++j; i = 0;
        }
fclose(ep);

/* the following area writes the subgroup file */

i = 0; j = 1;
while(subsets[j][0] != '\0')
        {
        while(i < 203)
                {
                subgroup_form[i] = '\0';
                ++i;
                }
```

```
        i = 0;
        while(subsets[j][i] != '\0')
                {
                int_value = subsets[j][i];
                if(int_value == 13)
                        {
                        subgroup_form[i] = -2;
                        ++i;
                        }
                else
                        {
                        if(int_value == 26)
                                {
                                subgroup_form[i] = -3;
                                ++i;
                                }
                        else
                                {
                                if(int_value == 115)
                                        {
                                        subgroup_form[i] = -4;
                                        ++i;
                                        }
                                else
                                        {
                                        subgroup_form[i] = subsets[j][i];
                                        ++i;
                                        }
                                }
                        }
                }
        subgroup_form[i] = '\0';
        sfile_value = fwrite(subgroup_form, 203, 1, sp);
        ++j; i = 0;
        }
fclose(sp);
}
```

```
/*

ABSALGCW.C -  Cyclic write data program for the Abstract Algebra Programming
System

This program writes the data created by the Cyclic Group Generation Program.

*/

extern char subsets[273][203], internal_form[155];

#include "stdio.h"


void cyclic_file_write(elements, cyclic)
char elements[203][155], cyclic[4];

        {
        int i, j, int_value, efile_value, sfile_value;
        char element_form[155], subgroup_form[203];
        char cyclic_element[12], cyclic_subgroup[12];
        FILE *ep;
        FILE *sp;

        /* the following area is the filename construction area */

        i = 0;
        cyclic_element[0] = 'C';
        cyclic_element[1] = 'E';
        cyclic_element[2] = '\0';
        cyclic_subgroup[0] = 'C';
        cyclic_subgroup[1] = 'S';
        cyclic_subgroup[2] = '\0';
        strcat(cyclic_element, cyclic);
        strcat(cyclic_subgroup, cyclic);
        strcat(cyclic_element, ".DAT");
        strcat(cyclic_subgroup, ".DAT");

        /* the following area opens the files */

        if((ep = fopen(cyclic_element, "wt")) == NULL)
                {
                printf("cannot open element file\n");
                exit(1);
                }
        if((sp = fopen(cyclic_subgroup, "wt")) == NULL)
                {
                printf("cannot open subgroup file\n");
                exit(1);
                }
```

```
/* the following area writes the element file */

i = 0; j = 1;
while(elements[j][0] != '\0')
        {
        while(i < 155)
                {
                element_form[i] = '\0';
                ++i;
                }
        i = 0;
        while(elements[j][i] != '\0')
                {
                int_value = elements[j][i];
                if(int_value == 13)
                        {
                        element_form[i] = -2;
                        ++i;
                        }
                else
                        {
                        if(int_value == 26)
                                {
                                element_form[i] = -3;
                                ++i;
                                }
                        else
                                {
                                element_form[i] = elements[j][i];
                                ++i;
                                }
                        }
                }
        element_form[i] = '\0';
        efile_value = fwrite(element_form, 155, 1, ep);
        ++j; i = 0;
        }
fclose(ep);

/* the following area writes the subgroup file */

i = 0; j = 1;
while(subsets[j][0] != '\0')
        {
        while(i < 203)
                {
                subgroup_form[i] = '\0';
                ++i;
                }
```

```
        i = 0;
        while(subsets[j][i] != '\0')
                {
                int_value = subsets[j][i];
                if(int_value == 13)
                        {
                        subgroup_form[i] = -2;
                        ++i;
                        }
                else
                        {
                        if(int_value == 26)
                                {
                                subgroup_form[i] = -3;
                                ++i;
                                }
                        else
                                {
                                if(int_value == 115)
                                        {
                                        subgroup_form[i] = -4;
                                        ++i;
                                        }
                                else
                                        {
                                        subgroup_form[i] = subsets[j][i];
                                        ++i;
                                        }
                                }
                        }
                }
        subgroup_form[i] = '\0';
        sfile_value = fwrite(subgroup_form, 203, 1, sp);
        ++j; i = 0;
        }
fclose(sp);
}
```

```
/*

ABSALGAW.C -  Abelian write data program for the Abstract Algebra Programming
System

This program writes the data created by the Abelian Group Generation Program.

*/

extern char subsets[273][203], internal_form[155];

#include "stdio.h"


void abelian_file_write(elements, abelian_file_name)
char elements[203][155], abelian_file_name[8];

        {
        int i, j, int_value, efile_value, sfile_value;
        char element_form[155], subgroup_form[203];
        char abelian_element[15], abelian_subgroup[15];
        FILE *ep;
        FILE *sp;

        /* the following area is the filename construction area */

        i = 0;
        abelian_element[0] = '\0';
        abelian_subgroup[0] = '\0';
        strcpy(abelian_element, abelian_file_name);
        strcpy(abelian_subgroup, abelian_file_name);
        strcat(abelian_element, "E.DAT");
        strcat(abelian_subgroup, "S.DAT");

        /* the following area opens the files */

        if((ep = fopen(abelian_element, "wt")) == NULL)
                {
                printf("cannot open element file\n");
                exit(1);
                }
        if((sp = fopen(abelian_subgroup, "wt")) == NULL)
                {
                printf("cannot open subgroup file\n");
                exit(1);
                }

        /* the following area writes the element file */

        i = 0; j = 1;
```

```
while(elements[j][0] != '\0')
        {
        while(i < 155)
                {
                element_form[i] = '\0';
                ++i;
                }
        i = 0;
        while(elements[j][i] != '\0')
                {
                int_value = elements[j][i];
                if(int_value == 13)
                        {
                        element_form[i] = -2;
                        ++i;
                        }
                else
                        {
                        if(int_value == 26)
                                {
                                element_form[i] = -3;
                                ++i;
                                }
                        else
                                {
                                element_form[i] = elements[j][i];
                                ++i;
                                }
                        }
                }
        element_form[i] = '\0';
        efile_value = fwrite(element_form, 155, 1, ep);
        ++j; i = 0;
        }
fclose(ep);

/* the following area writes the subgroup file */

i = 0; j = 1;
while(subsets[j][0] != '\0')
        {
        while(i < 203)
                {
                subgroup_form[i] = '\0';
                ++i;
                }
        i = 0;
        while(subsets[j][i] != '\0')
                {
                int_value = subsets[j][i];
```

```
                if(int_value == 13)
                        {
                        subgroup_form[i] = -2;
                        ++i;
                        }
                else
                        {
                        if(int_value == 26)
                                {
                                subgroup_form[i] = -3;
                                ++i;
                                }
                        else
                                {
                                if(int_value == 115)
                                        {
                                        subgroup_form[i] = -4;
                                        ++i;
                                        }
                                else
                                        {
                                        subgroup_form[i] = subsets[j][i];
                                        ++i;
                                        }
                                }
                        }
                }
        subgroup_form[i] = '\0';
        sfile_value = fwrite(subgroup_form, 203, 1, sp);
        ++j; i = 0;
        }
fclose(sp);
}
```

```
/*

ABSALGUW.C -  Unknown write data program for the Abstract Algebra Programming
System

This program writes the data created by the Unknown Group Generation Program.

*/

extern char subsets[273][203], internal_form[155];

#include "stdio.h"


void unknown_file_write(elements, unknown_file_name)
char elements[203][155], unknown_file_name[8];

        {
        int i, j, int_value, efile_value, sfile_value;
        char element_form[155], subgroup_form[203];
        char unknown_element[15], unknown_subgroup[15];
        FILE *ep;
        FILE *sp;

        /* the following area is the filename construction area */

        i = 0;
        unknown_element[0] = '\0';
        unknown_subgroup[0] = '\0';
        strcpy(unknown_element, unknown_file_name);
        strcpy(unknown_subgroup, unknown_file_name);
        strcat(unknown_element, "E.DAT");
        strcat(unknown_subgroup, "S.DAT");

        /* the following area opens the files */

        if((ep = fopen(unknown_element, "wt")) == NULL)
                {
                printf("cannot open element file\n");
                exit(1);
                }
        if((sp = fopen(unknown_subgroup, "wt")) == NULL)
                {
                printf("cannot open subgroup file\n");
                exit(1);
                }

        /* the following area writes the element file */

        i = 0; j = 1;
```

```
while(elements[j][0] != '\0')
        {
        while(i < 155)
                {
                element_form[i] = '\0';
                ++i;
                }
        i = 0;
        while(elements[j][i] != '\0')
                {
                int_value = elements[j][i];
                if(int_value == 13)
                        {
                        element_form[i] = -2;
                        ++i;
                        }
                else
                        {
                        if(int_value == 26)
                                {
                                element_form[i] = -3;
                                ++i;
                                }
                        else
                                {
                                element_form[i] = elements[j][i];
                                ++i;
                                }
                        }
                }
        element_form[i] = '\0';
        efile_value = fwrite(element_form, 155, 1, ep);
        ++j; i = 0;
        }
fclose(ep);

/* the following area writes the subgroup file */

i = 0; j = 1;
while(subsets[j][0] != '\0')
        {
        while(i < 203)
                {
                subgroup_form[i] = '\0';
                ++i;
                }
        i = 0;
        while(subsets[j][i] != '\0')
                {
                int_value = subsets[j][i];
```

```
                if(int_value == 13)
                        {
                        subgroup_form[i] = -2;
                        ++i;
                        }
                else
                        {
                        if(int_value == 26)
                                {
                                subgroup_form[i] = -3;
                                ++i;
                                }
                        else
                                {
                                if(int_value == 115)
                                        {
                                        subgroup_form[i] = -4;
                                        ++i;
                                        }
                                else
                                        {
                                        subgroup_form[i] = subsets[j][i];
                                        ++i;
                                        }
                                }
                        }
                }
        subgroup_form[i] = '\0';
        sfile_value = fwrite(subgroup_form, 203, 1, sp);
        ++j; i = 0;
        }
fclose(sp);
}
```

```
/*

ABSALGGD.C - Group Generator Driver Program for Abstract Algebra
Programming System

This function calls all the functions necessary to generate a group.

*/

extern char subsets[273][203], internal_form[155];

void a_to_the_n(char elements[][], char *, int *, char *);
void subgroup_sort(char *);
void add_subgroups(char *, int *);
char check_subgroup(char *);
void a_times_b(char elements[][], char *, int *, char *);

work_driver(elements, order)
char elements[203][155];
int *order;

        {
        char save_set[407], set[407], work_form[311];
        char element_used_table[203][2];
        int  new_order, i, j, length_of_subset, compare_sets, m, n;
        int  new_num_sets, no_new_elements, do_not_have_subgroup, length;
        int  num_subsets, l, length_of_set;

        i = 0;
        while(i < 203)
                {
                element_used_table[i][0] = '\0';
                element_used_table[i][1] = '\0';
                ++i;
                }
        work_form[0] = '\0';
        save_set[0] = '\0';
        set[0] = '\0';
        new_order = *order;
        num_subsets = 0;
        new_num_sets = num_subsets;
        j = 1; i = 0; set[i] = '\0';
        while(j < *order)  /*  This sets up the base set of elements. */
                {
                set[i] = j;
                strcpy(internal_form, elements[j]);
                a_to_the_n(elements, set, &new_order, work_form);
                element_used_table[j][0] = 'x';
                subgroup_sort(set);
                add_subgroups(set, &new_num_sets);
```

```
        num_subsets = new_num_sets;
        i = 0; ++j;
        *order = new_order;
        set[i] = '\0';
        }
j = 1; i = 1;
while(*order <= 201)
        {
        no_new_elements = 1;
        length_of_subset = strlen(subsets[i]);
        length = length_of_subset + 1;
        if(length == *order)
                {
                ++i;
                length_of_subset = strlen(subsets[i]);
                }
        while(length_of_subset)
                {
                strcpy(set, subsets[i]);
                strcpy(save_set, subsets[i]);
                length_of_subset = strlen(subsets[j]);
                length = length_of_subset + 1;
                if(length == *order)
                        {
                        ++j;
                        length_of_subset = strlen(subsets[j]);
                        }
                while(length_of_subset)
                        {
                        l = 0;
                        length_of_set = strlen(set);
                        while(subsets[j][l] != '\0')
                                {
                                set[length_of_set] = subsets[j][l];
                                ++l; ++length_of_set;
                                }
                        set[length_of_set] = '\0';
                        subgroup_sort(set);
                        compare_sets = strcmp(set, save_set);
                        if(compare_sets)
                                {
                                do_not_have_subgroup = check_subgroup(set);
                                if(do_not_have_subgroup)
                                        {
                                        a_times_b(elements, set, &new_order, work_form);
                                        if(*order == new_order)
                                                {
                                                subgroup_sort(set);
                                                add_subgroups(set, &new_num_sets);
                                                if(new_num_sets == num_subsets)
```

```
                        {
                        strcpy(set, save_set);
                        ++j;
                        length_of_subset = strlen(subsets[j]);
                        if(length == *order)
                                {
                                ++j;
                                length_of_subset = strlen(subsets[j]);
                                }
                        }
                else
                        {
                        num_subsets = new_num_sets;
                        length_of_subset = 0; no_new_elements = 0;
                        }
                }
        else
                {
                subgroup_sort(set);
                add_subgroups(set, &new_num_sets);
                num_subsets = new_num_sets;
                length_of_subset = 0; no_new_elements = 0;
                *order = new_order;
                m = 1; n = 0; set[n] = '\0';
                while(m < *order)
                        {
                        if(element_used_table[m][0] != 'x')
                                {
                                set[n] = m;
                                strcpy(internal_form, elements[m]);
                                a_to_the_n(elements, set, &new_order,
                                                work_form);
                                element_used_table[m][0] = 'x';
                                subgroup_sort(set);
                                add_subgroups(set, &new_num_sets);
                                n = 0; ++m;
                                *order = new_order;
                                num_subsets = new_num_sets;
                                set[n] = '\0';
                                }
                        else
                                {
                                ++m;
                                }
                        }
                }
        }
else
        {
        ++j;
```

```
                                        length_of_subset = strlen(subsets[j]);
                                        if(length == *order)
                                                {
                                                ++j;
                                                length_of_subset = strlen(subsets[j]);
                                                }
                                        }
                                }
                        else
                                {
                                ++j;
                                length_of_subset = strlen(subsets[j]);
                                if(length == *order)
                                        {
                                        ++j;
                                        length_of_subset = strlen(subsets[j]);
                                        }
                                }
                        }
                if(no_new_elements)
                        {
                        ++i; j = 1;
                        length_of_subset = strlen(subsets[i]);
                        if(length_of_subset == *order)
                                {
                                ++i;
                                length_of_subset = strlen(subsets[i]);
                                }
                        }
                else
                        {
                        length_of_subset = 0;
                        }
                }
        if(no_new_elements)
                *order = 255;
        }
*order = new_order;
}
```

```
/*

ABSALGAN.C - Basic Premutation Raised to a Power Program for the Abstract
Algebra Programming System

The function receives a permutation and multiplies that permutation
by itself until the null string ('\0') is found.

*/

extern char subsets[273][203], internal_form[155];

void disjoint(char *);
void singleton_search();
void canonical_formatter();
void check_elements(char elements[][], int *, int *);

void a_to_the_n(elements, set, order, work_form)
char elements[203][155], set[407], work_form[311];
int *order;

        {
        char save_form[155];
        int k, element_number, i;
        int new_order, length_internal_form, length_work_form;


        save_form[0] = '\0';
        new_order = *order; k = 1;
        strcpy(save_form, internal_form);
        strcpy(work_form, internal_form);
        strcat(work_form, save_form);
        length_work_form = strlen(work_form);
        while(length_work_form)
                {
                disjoint(work_form);
                i = 0;
                while(work_form[i] != '\0')
                        {
                        internal_form[i] = work_form[i];
                        ++i;
                        }
                internal_form[i] = '\0';
                singleton_search();
                length_internal_form = strlen(internal_form);
                length_work_form = 0;
                if(length_internal_form)
                        {
                        canonical_formatter();
                        check_elements(elements, &new_order, &element_number);
```

```
                set[k] = element_number;
                ++k;
                strcpy(work_form, internal_form);
                strcat(work_form, save_form);
                length_work_form = strlen(work_form);
                }
        }
*order = new_order;
set[k] = '\0';
}
```

```
/*

ABSALGSS.C -  Subgroup Sorter Program for the Abstract Algebra Programming
System

This function recieves a subset and then sorts it from lowest to
highest.

*/

void subgroup_sort(set)
char set[407];

        {
        char temp_set[407], delete_set[407], element;
        int i, j, n, length_of_set, hold, second_hold;


        j = 0; i = 0; temp_set[0] = '\0'; delete_set[0] = '\0';
        element = set[i];
        hold = i; ++i; second_hold = 101;
        while(element != '\0')
                {
                while(set[i] != '\0')
                        {
                        if(element < set[i])
                                ++i;
                        else
                                {
                                if(element > set[i])
                                        {
                                        element = set[i];
                                        hold = i; ++i;
                                        }
                                else
                                        {
                                        second_hold = i; ++i;
                                        }
                                }
                        }
                temp_set[j] = element;
                i = 0; ++j; n = 0;
                length_of_set = strlen(set);
                while(i <= length_of_set)
                        {
                        if(i == hold || i == second_hold)
                                ++i;
                        else
                                {
                                delete_set[n] = set[i];
```

```
                        ++i; ++n;
                        }
                }
        strcpy(set, delete_set);
        i = 0;
        element = set[i];
        hold = i; ++i; second_hold = 101;
        }
temp_set[j] = '\0';
strcpy(set, temp_set);
}
```

```
/*

ABSALGAS.C - Adds generated Subgroups to the list of Subgroups for the
Abstract Algebra Programming System

This function searches all the known subsets that have been produced
to determine if the current subset is a new subset  If it is, then it
is added to the list and the list is returned to the calling function.

*/

extern char subsets[273][203], internal_form[155];

void add_subgroups(set, new_num_sets)
char set[407];
int *new_num_sets;


        {
        char temp_set[203];
        int i, j, new, already_have_it, length_of_subset;


        temp_set[0] = '\0';  i = 0;
        j = 1; already_have_it = 0;
        while(set[i] != '\0')
                {
                temp_set[i] = set[i];
                ++i;
                }
        temp_set[i] = '\0';
        length_of_subset = strlen(subsets[j]);
        while(length_of_subset)
                {
                new = strcmp(temp_set, subsets[j]);
                if(new)
                        {
                        ++j;
                        length_of_subset = strlen(subsets[j]);
                        }
                else
                        {
                        ++already_have_it;
                        length_of_subset = 0;
                        }
                }
        if(already_have_it)
                return;
        else
                {
```

```
        strcpy(subsets[j], temp_set);
        *new_num_sets = *new_num_sets + 1;
        }
    }
```

```
/*

ABSALGKG.C  -  Check Subgroups for Existence Program for the Abstract Algebra
Programming System

This function searches all the known subgroups that have been
produced to determine if the current subgroup is on the list.

*/

extern char subsets[273][203], internal_form[155];

char check_subgroup(set)
char  set[407];


        {
        char temp_set[203];
        int i, j, new, have_subgroup, length_of_subgroup;

        i = 0; temp_set[0] = '\0';
        while(set[i] != '\0')
                {
                temp_set[i] = set[i];
                ++i;
                }
        temp_set[i] = '\0';

        j = 1; have_subgroup = 1;
        length_of_subgroup = strlen(subsets[j]);
        while(length_of_subgroup)
                {
                new = strcmp(temp_set, subsets[j]);
                if(new)
                        {
                        ++j;
                        length_of_subgroup = strlen(subsets[j]);
                        }
                else
                        {
                        --have_subgroup;
                        length_of_subgroup = 0;
                        }
                }
        return have_subgroup;
        }
```

```
/*

ABSALGAB.C - Multiplier of two Permutations for Abstract Algebra
Programming System

This function receives a set of elements which is composed of 2
subgroups.  Its primary function is multiplying every element of
the set by every other element of the set both the left and the
right side.

*/

extern char subsets[273][203], internal_form[155];

void disjoint(char *);
void singleton_search();
void canonical_formatter();
void check_elements(char elements[][], int *, int *);
void subgroup_sort(char *);

void a_times_b(elements, set, order, work_form)
char elements[203][155], set[407], work_form[311];
int  *order;

        {
        int i, j, n, new_order, length_of_set, count, element, multiplier;
        int multiplican, length_of_internal_form, element_number;
        int save_length_of_set, do_not_have_it, m;


        save_length_of_set = strlen(set);
        length_of_set = save_length_of_set;
        new_order = *order;
        i = 0; j = i;

        while(i < length_of_set)
                {
                multiplier = set[i];
                multiplican = set[j];
                strcpy(work_form, elements[multiplier]);
                strcat(work_form, elements[multiplican]);
                while(j < length_of_set)
                        {
                        count = 2;
                        while(count)
                                {
                                disjoint(work_form);
                                m = 0;
                                while(work_form[m] != '\0')
                                        {
```

```
                        internal_form[m] = work_form[m];
                        ++m;
                        }
                internal_form[m] = '\0';
                singleton_search();
                length_of_internal_form = strlen(internal_form);
                if(length_of_internal_form)
                        {
                        --count;
                        canonical_formatter();
                        check_elements(elements, &new_order, &element_number);
                        n = 0; do_not_have_it = 1;
                        while(n < length_of_set)
                                {
                                element = set[n];
                                if(element_number == element)
                                        {
                                        n = length_of_set;
                                        do_not_have_it = 0;
                                        }
                                else
                                        ++n;
                                }
                        if(do_not_have_it)
                                {
                                set[n] = element_number;
                                element = 0; ++n;
                                set[n] = element;
                                length_of_set = strlen(set);
                                }
                        }
                else
                        count = 0;
                if(count)
                        {
                        strcpy(work_form, elements[multiplican]);
                        strcat(work_form, elements[multiplier]);
                        }
                }
        ++j;
        length_of_set = strlen(set);
        if(j < length_of_set)
                {
                strcpy(work_form, elements[multiplier]);
                multiplican = set[j];
                strcat(work_form, elements[multiplican]);
                }
        }
length_of_set = strlen(set);
if(save_length_of_set == length_of_set)
```

```
                {
                ++i; j = i;
                }
        else
                {
                subgroup_sort(set);
                i = 0; j = i;
                save_length_of_set = strlen(set);
                length_of_set = save_length_of_set;
                }
        }
*order = new_order;
}
```

```
/*

ABSALGPM.C - Basic Permutation Multiplier for Abstract Algebra
Programming System

This program is called by PROGRAM A_TO_THE_N and A_TIMES_B to perform
all the necessary multiplication of the permutation generated by the
program.

This program uses Algorithm A (Multiply permutations in cycle form)
from FUNDAMENTAL ALGORITHMS, page 162, by Knuth, 1969.

*/


void disjoint(work_form)
char work_form[311];

        {
        char temp_form[311], start, current;
        int i, j, m, already_used;
        int n, untagged, length_work_form;

        temp_form[0] = '\0';
        length_work_form = strlen(work_form);
        i = 0; j = 0;
        temp_form[i] = work_form[j];
        ++i; ++j;
        temp_form[i] = work_form[j];
        start = work_form[j];
        ++i; ++j;
        current = work_form[j];

        while(start != '\0')
                {
                if(current == work_form[0] || current == '\0')
                        {
                        --j;
                        current = work_form[j];
                        while(current != work_form[0])
                                {
                                --j;
                                current = work_form[j];
                                }
                        ++j;
                        current = work_form[j];
                        }
                ++j;
                while(j <= length_work_form)
                        {
```

```
        while(current != work_form[j])
                {
                if(work_form[j] == '\0')
                        break;
                ++j;
                }
        if(work_form[j] != '\0')
                {
                ++j;
                current = work_form[j];
                if(current == work_form[0] || current == '\0')
                        {
                        --j;
                        current = work_form[j];
                        while(current != work_form[0])
                                {
                                --j;
                                current = work_form[j];
                                }
                        ++j;
                        current = work_form[j];
                        }
                ++j;
                }
        else
                j = length_work_form + 1;
        }
if(current != start)
        {
        temp_form[i] = current;
        ++i; j = -1;
        }
else
        {
        j = 0;
        untagged = work_form[j];
        while(untagged != 0)
                {
                already_used = 0;
                m = 0;
                while(m < i)
                        {
                        if(temp_form[m] != work_form[j])
                                ++m;
                        else
                                {
                                already_used = 1;
                                m = i;
                                }
                        }
```

```
                        if(already_used)
                                {
                                ++j;
                                untagged = work_form[j];
                                }
                        else
                                break;
                        }
                if(untagged == 0)
                        {
                        temp_form[i] = '\0';
                        start = temp_form[i];
                        }
                else
                        {
                        temp_form[i] = work_form[0];
                        ++i;
                        start = work_form[j];
                        temp_form[i] = start;
                        ++j; ++i;
                        current = work_form[j];
                        }
                }
        }
work_form[0] = '\0';
strcpy(work_form, temp_form);
}
```

```
/*

ABSALGSI.C - Singleton Search and Deletion Program for the Abstract Algebra
Programming System

This function searches a permutation, which is in the internal format,
for singleton cycles.  When a singleton cycle is encountered it is
deleted.  This function is called after every permutation
multiplication.

*/

extern char subsets[273][203], internal_form[155];

void singleton_search()


        {
        char temp_form[155];
        int i, j, m, n;


        temp_form[0] = '\0';
        j = 0; i = 0; m = 0;
        n = strlen(internal_form);
        while(j < n)
                {
                if(internal_form[j] == internal_form[0])
                        {
                        i = j + 2;
                        if(internal_form[i] == internal_form[0] || internal_form[i] == '\0')
                                {
                                j = j + 2;
                                }
                        else
                                {
                                temp_form[m] = internal_form[j];
                                ++m; ++j;
                                }
                        }
                else
                        {
                        temp_form[m] = internal_form[j];
                        ++m; ++j;
                        }
                }
        temp_form[m] = internal_form[n];
        strcpy(internal_form, temp_form);
        }
```

```
/*

ABSALGKE.C - Element Checker for the Abstract Algebra Programming System

This function searches all the known elements that have been produced
to determine if the current element is a new element.  If it is then
it is added to the list and the list is returned to the calling
function.

*/

extern char subsets[273][203], internal_form[155];

void check_elements(elements, order, element_number)
char elements[203][155];
int *order, *element_number;


        {
        int j, length_of_element, new, already_have_it;


        j = 1; already_have_it = 0; *element_number = 0;
        length_of_element = strlen(elements[j]);
        while(length_of_element)
                {
                new = strcmp(internal_form, elements[j]);
                if(new)
                        {
                        ++j;
                        length_of_element = strlen(elements[j]);
                        }
                else
                        {
                        ++already_have_it;
                        *element_number = j;
                        length_of_element = 0;
                        }
                }
        if(already_have_it)
                return;
        else
                {
                strcpy(elements[j], internal_form);
                *element_number = j;
                *order = *order + 1;
                }
        }
```

[redacted]