Introduction
To
C
Programming

Presented by:
Jim Polzin

e-mail: james.polzin@normandale.edu
otto: http://otto.normandale.edu
# TABLE OF CONTENTS

**TABLE OF CONTENTS** ................................................................. 2

C OVERVIEW ......................................................................................... 4

BASIC C PROGRAM STRUCTURE .......................................................... 5

COMPILING AND LINKING ................................................................. 6

FUNDAMENTAL DATA TYPES ............................................................. 7

COMMENTS .......................................................................................... 8

IDENTIFIERS ....................................................................................... 9

KEYWORDS .......................................................................................... 10

BASIC INPUT AND OUTPUT ............................................................... 11

CONVERSION SPECIFIERS ................................................................. 12

ESCAPE SEQUENCES ....................................................................... 13

OPERATORS ......................................................................................... 17

OPERATOR PRECEDENCE ................................................................. 18

OPERATOR PRECEDENCE CHART ...................................................... 19

ARITHMETIC OPERATORS ................................................................. 20

INCREMENT ++/DECREMENT -- OPERATORS ...................................... 31

STRINGS ............................................................................................. 32

FUNCTIONS ....................................................................................... 40

LOGICAL, TRUE/FALSE VALUES ...................................................... 49

RELATIONAL OPERATORS ............................................................... 49

LOGICAL OPERATORS ..................................................................... 49

LOOPING ............................................................................................ 50

MATH LIBRARIES ............................................................................. 56

CONDITIONAL STATEMENTS ......................................................... 59

FUNCTIONS – THE DETAILS ........................................................... 68

POINTERS .......................................................................................... 78

ARRAYS .............................................................................................. 81

ARRAYS AND POINTERS .................................................................... 94

BASIC MULTI-DIMENSIONAL ARRAYS ........................................... 99

MULTIDIMENSIONAL ARRAYS AND POINTERS ............................. 101

COMMAND-LINE ARGUMENTS ........................................................ 112

OPENGL/GLUT..................................................................................... 114

MAKE UTILITY .................................................................................. 143

C STORAGE CLASSES ...................................................................... 145

DYNAMIC MEMORY ALLOCATION ............................................... 148

DYNAMIC MULTIDIMENSIONAL ARRAYS ..................................... 157

TEXT FILE I/O .................................................................................. 167

BINARY FILE I/O .............................................................................. 172

STRUCTURES ..................................................................................... 174

ENUMERATED TYPES ..................................................................... 197

---

Introduction to C Programming

---

Page: 2

Copyright © 2015 by James J. Polzin  All Rights Reserved

Printed: 6/13/15
UNIONS ................................................................. 202
TYPEDEF ........................................................... 205
FUNCTION POINTERS/QUICK SORT ...................... 207
BIT OPERATORS ..................................................... 214
PREPROCESSOR – MORE FEATURES ....................... 217
BUILDING LIBRARIES ........................................... 231
USING SEARCH PATHS .......................................... 236
WRITING TEXT WITH OPENGL/GLUT ..................... 238
LINKED LISTS ...................................................... 250
ADT – ABSTRACT DATA TYPE .............................. 270
INDEX ....................................................................... 281
C OVERVIEW

Goals

• speed
• portability
• allow access to features of the architecture
• speed

C

• fast executables
• allows high-level structure without losing access to machine features
• many popular languages such as C++, Java, Perl use C syntax/C as a basis
• generally a compiled language
• reasonably portable
• very available and popular
BASIC C PROGRAM STRUCTURE

- The function `main()` is found in every C program and is where every C program begins execution.
- C uses braces `{ }` to delimit the start/end of a function and to group sets of statements together into what C calls a block of code.
- Semicolons are used to terminate each C statement.
- Groups of instructions can be gathered together and named for ease of use and ease of programming. These “modules” are called functions in C.

Ex:

```c
/*     FILE: first.c     */
#include <stdio.h>
int main( )
{
    printf("Hello world! \n");
    return 0;
}

/*     OUTPUT: first.c     */
Hello world!
*/
```
COMPILING AND LINKING

- Producing an executable file from C source code involves a two step process, compiling and linking.
- The compiler translates the C code into machine code, the linker combines the new machine code with code for existing routines from available libraries and adds some startup code.
- The end result is a file full of machine instructions that are executable on the target machine.

Ex:

```
gcc first.c
  • compile and link to a.exe

gcc -c first.c
  • compile only, stop at object module

gcc -lm first.c
  • link in math libraries
```
FUNDAMENTAL DATA TYPES

- there are three basic types of data, integer, floating-point, and character
- character data type is really a small integer
- signed and unsigned integer types available

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1 byte</td>
<td>0 / -128</td>
<td>255 / 127</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>int</td>
<td>2,4 bytes</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>long</td>
<td>4 bytes</td>
<td>-2147483648</td>
<td>2147483647</td>
</tr>
<tr>
<td>long long</td>
<td>8 bytes</td>
<td>-9x10^18/2^{63}</td>
<td>9x10^18/2^{63}-1</td>
</tr>
<tr>
<td>float</td>
<td>4 bytes ~ 7 digits</td>
<td>+1.0x10^{-37}</td>
<td>+3.4x10^{-38}</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes ~ 14 digits</td>
<td>+1.0x10^{-307}</td>
<td>+1.8x10^{-308}</td>
</tr>
<tr>
<td>long double</td>
<td>12 bytes ~ 20 digits</td>
<td>+1.0x10^{-3931}</td>
<td>+1.0x10^{-3932}</td>
</tr>
</tbody>
</table>

Ex:

```c
/* FILE: unsigned.c */

/*
 Illustration of the unsigned keyword. Allows
 recovering the use of the lead bit for magnitude.
*/
#include <stdio.h>

int main() {
    unsigned int x;
    x = 3333222111;
    printf("Unsigned x = %u\n", x);
    printf("Signed x = %d\n", x);
    return 0;
}

/* OUTPUT: unsigned.c

    Unsigned x = 3333222111
    Signed x = -961745185 */
```
COMMENTS

- Block-style comments /* … */ Everything between the opening /* and the first */ is a comment.
- Comment-to-end-of-line: // Everything from the // to the end of the line is a comment.
- Nesting of block-style comments doesn’t work.
- Note: Some older compilers may not recognize the // comment indicator.

Ex:

```c
/* FILE: example.c */
#include <stdio.h>

/* C-style comments can span several lines
   ...where they are terminated by:
*/

int main( )
{
    printf("Here's a program\n");
    return 0;
}

/* OUTPUT: example.c
   Here's a program
*/
```
IDENTIFIERS

• C identifiers must follow these rules:
  • C is case sensitive
  • may consist of letters, digits, and the underscore
  • first character must be a letter, (could be underscore but this is discouraged)
  • no length limit but only the first 31-63 may be significant
KEYWORDS

auto     enum     restrict     unsigned
break    extern   return     void
case     float    short      volatile
char     for      signed     while
const    goto     sizeof     _Bool
continue if      static     _Complex
default  inline   struct     _Imaginary
do       int       switch
double   long      typedef
else      register  union
BASIC INPUT AND OUTPUT

- The basic I/O functions are printf( ) and scanf( ).
- printf( ) and scanf( ) are very generic. They always are processing text on one end. They get all their information about the data type they are to print or scan from the conversion specifiers.
- printf( ) always is producing text output from any number of internal data formats, i.e. int, float, char, double. The job of the conversion specifiers is to tell printf( ) how big a piece of data it's getting and how to interpret the internal representation.
- scanf( ) always receives a text representation of some data and must produce an internal representation. It is the conversion specifiers job to tell scanf( ) how to interpret the text and what internal representation to produce.
- **printf( ) tips and warnings:**
  - Make sure your conversion specifiers match your data values in number, type and order.
  - Use %f for both float and double.
  - Everything you put in the format string prints exactly as it appears, except conversion specifiers and escape sequences.
- **scanf( ) tips and warnings:**
  - Make sure your conversion specifiers match your data values in number, type and order.
  - As a general rule, scan only one value with each scanf( ) call, unless you really know what you are doing.
  - Use %f for float, %lf for double and %Lf for long double.
  - Don't forget the &, except with strings. {Someday you'll know why that is, and it will make sense.}
  - For %c every character you type is a candidate, even <return>. Placing a space in front of the %c in the format string will cause scanf( ) to skip whitespace characters.
  - scanf( ) is NOT without it's problems. However, it provides an easy way to get text input into a program and has some very handy conversion capabilities.
CONVERSION SPECIFIERS

printf( )

%d  signed decimal int
%hd  signed short decimal integer
%ld  signed long decimal integer
%lld signed long long decimal integer
%u  unsigned decimal int
%lu  unsigned long decimal int
%llu unsigned long long decimal int
%o  unsigned octal int
%x  unsigned hexadecimal int with lowercase
%X  unsigned hexadecimal int with uppercase
%f  float or double [-]dddd.dddd.
%e  float or double of the form [-]d.dddd  e[+/-]ddd
%g  either e or f form, whichever is shorter
%E  same as e; with E for exponent
%G  same as g; with E for exponent if e format used
%Lf,
%Le,
%Lf  long double
%Le  long double
%c  single character
%s  string
%p  pointer

scanf( )

%d  signed decimal int
%hd  signed short decimal integer
%ld  signed long decimal integer
%lu  unsigned decimal int
%lu  unsigned long decimal int
%o  unsigned octal int
%x  unsigned hexadecimal int
%f  float
%lf double \textbf{NOTE}: double & float are distinct for scanf !!!!
%LF  long double
%c  single character
%s  string
ESCAPE SEQUENCES

- Certain characters are difficult to place in C code so an escape code or escape sequence is used to encode these characters.
- These escape sequences all begin with a backslash ‘\’ and cause the encoded character to be placed into the program.

<table>
<thead>
<tr>
<th>Escape</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>newline</td>
</tr>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\f</td>
<td>formfeed</td>
</tr>
<tr>
<td>\a</td>
<td>alarm</td>
</tr>
<tr>
<td>\b</td>
<td>backspace</td>
</tr>
<tr>
<td>\r</td>
<td>carriage return</td>
</tr>
<tr>
<td>\v</td>
<td>vertical tab</td>
</tr>
</tbody>
</table>

Ex:

```c
#include <stdio.h>

int main( )
{
    int x = 12;
    float y = 3.75;
    printf("%d", x);
    printf("\n\nx = %d\n", x);
    printf("y = %f\n", y);
    return 0;
}
```

/*     FILE: print.c     */
/*
   Illustration of printf( ) and conversion specifiers.
*/
#include <stdio.h>

int main( )
{
    int x = 12;
    float y = 3.75;
    printf("%d", x);
    printf("\n\nx = %d\n", x);
    printf("y = %f\n", y);
    return 0;
}

/*     OUTPUT: print.c     */
12
x = 12
y = 3.750000

*/
Ex:

```c
/*     FILE: scan.c     */
/*
   Illustration of scanf( ).
*/
#include <stdio.h>

int main( )
{
  int x;
  float y;
  printf("x = %d\n", x);
  printf("y = %f\n", y);
  printf("Enter an integer value for x: ");
  scanf("%d", &x);
  printf("Enter a floating-point value for y: ");
  scanf("%f", &y);
  printf("x = %d\n", x);
  printf("y = %f\n", y);
  return 0;
}

/*    OUTPUT: scan.c    */
x = 4206596
y = 0.000000
Enter an integer value for x: 7
Enter a floating-point value for y: 3.3
x = 7
y = 3.300000
*/
```
Ex:

```c
/*     FILE: scan2.c     */
/
/*     Illustration of scanf( ) with characters and characters are integers. */
#include <stdio.h>

int main( )
{
    char c;
    printf("Enter a character: ");
    scanf("%c", &c);
    printf("c = %c\n", c);
    printf("c = %d\n", c);
    return 0;
}

/*     OUTPUT: scan2.c */
Enter a character: A
    c = A
    c = 65
*/
```
Ex:

/* FILE: scan3.c */
/*
   Illustration of interpretation caused by conversion specifiers.
*/
#include <stdio.h>

int main( )
{
    char c;
    int x;

    printf("Enter a character: ");
    scanf("%c", &c);
    printf("c = %c\n", c);
    printf("c = %d\n", c);

    printf("Enter an integer: ");
    scanf("%d", &x);
    printf("x = %d\n", x);
    printf("x = %c\n", x);
    return 0;
}

/* OUTPUT: scan3.c

Enter a character: 6
   c = 6
   c = 54
Enter an integer: 6
   x = 6
   x = _

*/
OPERATORS

Arithmetic operators:

* / %  multiplication/division/modulus
+ −  addition/subtraction
+ −  positive/negative sign (unary)
++ − −  increment/decrement (unary)

Logical operators:

&&  AND
||  OR
!  NOT (unary)

Relational operators:

< <= > >=  less than, less than or equal to, greater than, greater than or equal to
== !=  equal to and not equal to

Bit operators:

<< >>  left and right bit shift
&  bitwise AND
|  bitwise OR
^  bitwise exclusive or XOR
~  bitwise NOT (unary)

Assignment operators:

= += -= *= /= %= &= ^= |= <<= >>=

Address/Pointer operators:

&  address of (unary)
*  dereference (unary)

Structure operators:

.  structure member access
->  member access thru a structure pointer

Other operators:

()  function call
[]  array access
(type)  type cast (unary)
sizeof  data object size in bytes (unary)
?:  conditional operator
,  comma operator
OPERATOR PRECEDENCE

- The C compiler determines the order of operations in a C statement by operator precedence.
- Operator Precedence is the ranking of operators in C. The higher the rank the sooner the operator is evaluated.
- Parentheses can be used to override operator precedence.
- There are many kinds of operators but all operators are ranked via operator precedence.
- In the case of operators with the same rank, associativity is used and the operators are evaluated left-to-right or right-to-left.
- Operator precedence and associativity are detailed in the Operator Precedence Chart in the appendix, on the following page, and on pg. 53 in the K&R book.
## OPERATOR PRECEDENCE CHART

<table>
<thead>
<tr>
<th>Operators</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ) [ ] ! ~ ++ – – + – * &amp; (type) sizeof</td>
<td>left to right</td>
</tr>
<tr>
<td>* / %</td>
<td>All Unary</td>
</tr>
<tr>
<td>+ – &lt;&lt; &gt;&gt;</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>left to right</td>
</tr>
<tr>
<td>== != &amp;</td>
<td>left to right</td>
</tr>
<tr>
<td>^</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>left to right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>?:</td>
<td>right to left</td>
</tr>
<tr>
<td>= += -= *= /= %= &amp;= ^=</td>
<td>= &lt;&lt;= &gt;&gt;=</td>
</tr>
<tr>
<td>,</td>
<td>left to right</td>
</tr>
</tbody>
</table>
ARITHMETIC OPERATORS

• Arithmetic operators are the symbols used by C to indicate when an arithmetic operation is desired.

• Arithmetic operators follow the same precedence rules we learned as kids. Multiplication & division before addition and subtraction. In case of a tie evaluate left-to-right. { Look at a precedence chart and see if this is true. }

• The modulus operator, %, is an additional arithmetic operator. It produces the remainder of integer division and ranks at the same level as division in the precedence chart.

• The increment, ++, and decrement, --, operators are basically a shorthand notation for increasing or decreasing the value of a variable by one.

Ex:
/*   FILE: arith_1.c   */
/*/ Arithmetic operators */
#include <stdio.h>
int main( )
{
    int first, second, sum;
    first = 11;
    second = 12;
    sum = first + second;
    printf("sum = %d\n", sum);
    sum = first - second;
    printf("sum = %d\n", sum);
    sum = first * second;
    printf("sum = %d\n", sum);
    sum = first / second;
    printf("sum = %d\n", sum);
    return 0;
}

/*   OUTPUT: arith_1.c   
    sum = 23
    sum = -1
    sum = 132
    sum = 0
   */
Ex:
/*     FILE: arith_2.c     */
/* Arithmetic operators with nicer output */
#include <stdio.h>

int main( )
{
    int first, second, sum;
    first = 11;
    second = 12;
    sum = first + second;
    printf("%d + %d = %d\n", first, second, sum);
    sum = first - second;
    printf("%d - %d = %d\n", first, second, sum);
    sum = first * second;
    printf("%d * %d = %d\n", first, second, sum);
    sum = first / second;
    printf("%d / %d = %d\n", first, second, sum);
    return 0;
}

/*  OUTPUT: arith_2.c
   11 + 12 = 23
   11 - 12 = -1
   11 * 12 = 132
   11 / 12 = 0
*/
Ex:
/*     FILE: arith_3.c     */
/* More arithmetic operators with nicer output */
#include <stdio.h>

int main( )
{
    int first, second, sum;
    first = 11;
    second = 12;
    sum = first + second;
    printf("%d + %d = %d\n", first, second, sum);
    sum = first - second;
    printf("%d - %d = %d\n", first, second, sum);
    sum = second - first;
    printf("%d - %d = %d\n", second, first, sum);
    sum = first * second;
    printf("%d * %d = %d\n", first, second, sum);
    sum = first / second;
    printf("%d / %d = %d\n", first, second, sum);
    return 0;
}

/*     OUTPUT: arith_3.c     */
11 + 12 = 23
11 - 12 = -1
12 - 11 = 1
11 * 12 = 132
11 / 12 = 0
*/
Ex:

/*     FILE: arith_4.c     */
/* Arithmetic operators with floating-point data */

#include <stdio.h>

int main( )
{
  float first, second, sum;

  first = 11;
  second = 12;

  sum = first + second;
  printf("%f + %f = %f\n", first, second, sum);

  sum = first - second;
  printf("%f - %f = %f\n", first, second, sum);

  sum = second - first;
  printf("%f - %f = %f\n", second, first, sum);

  sum = first * second;
  printf("%f * %f = %f\n", first, second, sum);

  sum = first / second;
  printf("%f / %f = %f\n", first, second, sum);

  return 0;
}

/*    OUTPUT: arith_4.c
   11.000000 + 12.000000 = 23.000000
   11.000000 - 12.000000 = -1.000000
   12.000000 - 11.000000 = 1.000000
   11.000000 * 12.000000 = 132.000000
   11.000000 / 12.000000 = 0.916667
 */
Ex:

/*     FILE: arith_5.c     */
/* More arithmetic operators with floating-point data */
#include <stdio.h>

int main( )
{
    float first, second, sum;
    first = 1.35;
    second = 2.75;
    sum = first + second;
    printf("%f + %f = %f\n", first, second, sum);
    sum = first - second;
    printf("%f - %f = %f\n", first, second, sum);
    sum = second - first;
    printf("%f - %f = %f\n", second, first, sum);
    sum = first * second;
    printf("%f * %f = %f\n", first, second, sum);
    sum = first / second;
    printf("%f / %f = %f\n", first, second, sum);
    return 0;
}

/*    OUTPUT: arith_5.c    */
1.350000 + 2.750000 = 4.100000
1.350000 - 2.750000 = -1.400000
2.750000 - 1.350000 = 1.400000
1.350000 * 2.750000 = 3.712500
1.350000 / 2.750000 = 0.490909
*/
Ex:

/* FILE: arith_6.c */
/* Precedence of operators */
#include <stdio.h>

int main()
{
    int first, second, sum;

    first = 10;
    second = 12;

    sum = first + second / 3;
    printf("%d + %d / 3 = %d\n", first, second, sum);
    return 0;
}

/* OUTPUT: arith_6.c

    10 + 12 / 3 = 14

*/
Ex:
/*     FILE: arith_7.c     */
/* Parentheses override precedence of operators */
#include <stdio.h>
int main( )
{
    int first, second, sum;
    first = 10;
    second = 12;
    sum = (first + second) / 3;
    printf("(%d + %d) / 3 = %d\n", first, second, sum);
    return 0;
}

/*    OUTPUT: arith_7.c    */
(10 + 12) / 3 = 7
*/
Ex:

/*     FILE: computation.c     */

/* Computes the cost per sq inch of pizza
   -- inspired by pizza.c example in C
   Primer Plus by Prata */

#include <stdio.h>

int main( )
{
    int diameter, radius, area, price, pricePerInch;

    printf("What is the price of your pizza: ");
    scanf("%d", &price);

    printf("What is the diameter of your pizza: ");
    scanf("%d", &diameter);

    radius = diameter/2;
    area = 3.14159 * radius * radius;
    pricePerInch = price/area;

    printf("Pizza analysis:\n");
    printf("    diameter = %d\n", diameter);
    printf("      radius = %d\n", radius);
    printf("        area = %d\n", area);
    printf("       price = %d per sq. inch\n", pricePerInch);

    return 0;
}

/*    OUTPUT: computation.c

    What is the price of your pizza: 10.50
    What is the diameter of your pizza:
    Pizza analysis:
        diameter = 4206596
        radius = 2103298
        area = -2147483648
        price = 0 per sq. inch

    What is the price of your pizza: 10
    What is the diameter of your pizza: 14
    Pizza analysis:
        diameter = 14
        radius = 7
        area = 153
        price = 0 per sq. inch */
Ex:

/* FILE: computation2.c */

/* Computes the cost per sq inch of pizza

Uses a float for price, to get dollars
and cents.

*/

#include <stdio.h>

int main( )
{
    int diameter, radius, area, pricePerInch;
    float price;

    printf("What is the price of your pizza: ");
    scanf("%f", &price);

    printf("What is the diameter of your pizza: ");
    scanf("%d", &diameter);

    radius = diameter/2;
    area = 3.14159 * radius * radius;
    pricePerInch = price/area;

    printf("Pizza analysis:\n");
    printf("    diameter = %d\n", diameter);
    printf("    radius = %d\n", radius);
    printf("    area = %d\n", area);
    printf("    price = %d per sq. inch\n", pricePerInch);

    return 0;
}

/* OUTPUT: computation2.c

What is the price of your pizza: 10.50
What is the diameter of your pizza: 14
Pizza analysis:
    diameter = 14
    radius = 7
    area = 153
    price = 0 per sq. inch

*/
Ex:

/*     FILE: computation3.c     */
/* Computes the cost per sq inch of pizza

More floating-point.           */

#include <stdio.h>

int main( )
{
    int diameter;
    float price, radius, area, pricePerInch;

    printf("What is the price of your pizza: ");
    scanf("%f", &price);
    printf("What is the diameter of your pizza: ");
    scanf("%d", &diameter);

    radius = diameter/2;
    area = 3.14159 * radius * radius;
    pricePerInch = price/area;

    printf("Pizza analysis:\n");
    printf("    diameter = %d\n", diameter);
    printf("    radius = %.2f\n", radius);
    printf("    area = %.2f\n", area);
    printf("    price = %.2f per sq. inch\n", pricePerInch);

    return 0;
}

/*    OUTPUT: computation3.c

What is the price of your pizza: 10.50
What is the diameter of your pizza: 14
Pizza analysis:
    diameter = 14
    radius = 7.000000
    area = 153.937912
    price = 0.07 per sq. inch

What is the price of your pizza: 15.50
What is the diameter of your pizza: 18
Pizza analysis:
    diameter = 18
    radius = 9.000000
    area = 254.468796
    price = 0.06 per sq. inch

What is the price of your pizza: 15.50
What is the diameter of your pizza: 19
Pizza analysis:
    diameter = 19
    radius = 9.000000
    area = 254.468796
    price = 0.06 per sq. inch

*/
Ex:

/*     FILE: computation4.c     */
/* Computes the cost per sq inch of pizza
   A type cast. */

#include <stdio.h>
#define PI 3.14159

int main( )
{
    int diameter;
    float price, radius, area, pricePerInch;

    printf("What is the price of your pizza: ");
    scanf("%f", &price);

    printf("What is the diameter of your pizza: ");
    scanf("%d", &diameter);

    radius = (float)diameter/2;
    area = PI * radius * radius;
    pricePerInch = price/area;

    printf("Pizza analysis:\n");
    printf("    diameter = %d\n", diameter);
    printf("      radius = %f\n", radius);
    printf("        area = %f\n", area);
    printf("  price = %.2f per sq. inch\n", pricePerInch);

    return 0;
}

/*    OUTPUT: computation4.c

        What is the price of your pizza: 15.50
        What is the diameter of your pizza: 18
        Pizza analysis:
               diameter = 18
               radius = 9.000000
               area = 254.468796
               price = 0.06 per sq. inch


        What is the price of your pizza: 15.50
        What is the diameter of your pizza: 19
        Pizza analysis:
               diameter = 19
               radius = 9.500000
               area = 283.528503
               price = 0.05 per sq. inch

        */
INCREMENT ++/DECREMENT -- OPERATORS

- C has two specialized operators for incrementing and decrementing the value of a variable.
  
  
  ```
  // will increase a variables value by “one”
  -- - will decrease a variables value by “one”
  ```

- Both operators can be written in both prefix and postfix notation. Each has implications as to when the actual increment or decrement takes place. Fortunately the implications are reasonable. Prefix notation causes the increment/decrement to occur “before” the value of the variable is supplied to an expression. Postfix notation causes the increment/decrement to occur “after” the value of the variable is supplied to an expression. In all cases the variables value is increased/decreased by “one”

Ex:

```c
/*     FILE: incDec.c     */
/* Example of increment & decrement, postfix and prefix. */
#include <stdio.h>
int main( )
{
  i = 7;
  printf("i = %d\n", i++);
  printf("After postfix ++, i = %d\n", i);
  printf("i = %d\n", +i);
  printf("After prefix ++, i = %d\n", i);
  printf("i = %d\n", i--);
  printf("After postfix --, i = %d\n", i);
  printf("i = %d\n", --i);
  printf("After prefix --, i = %d\n", i);
  return 0;
}
/*   OUTPUT: incDec.c   */
i = 7
After postfix ++, i = 8
i = 9
After prefix ++, i = 9
i = 9
After postfix --, i = 8
i = 7
After prefix --, i = 7
*/
```
Strings

• The C definition of a string is: a set of characters terminated by a null character.
• A set of characters written inside of double quotes indicates to the compiler that it is a string.
• Placement of the null character gets handled by C itself, when C can identify that it is working with strings.
• A programmer can create and manipulate a string as a set of char locations. This set of locations can be created as an array. The programmer must then be sure that the set is used properly so that the terminating null gets placed at the end of the characters so that it represents a legitimate string.
Ex:

/*       FILE: string.c       */
/* Basic C string functionality */

#include <stdio.h>

int main( )
{
    char name[81];

    printf("Prompts are strings.\n");
    printf("String - %s", "Please enter a string: ");
    scanf("%s", name);
    printf("\n\nYou entered- %s\n", name);
    return 0;
}

/* OUTPUT: string.c
   Prompts are strings.
   String - Please enter a string: Jim
       You entered- Jim
*/

Ex:

/*       FILE: string2.c       */
/* Basic C string functionality */

#include <stdio.h>

int main( )
{
    char name[81];

    name[0] = 'J';
    name[1] = 'i';
    name[2] = 'm';
    name[3] = '\0';

    printf("\n\nYou created: %s\n", name);
    return 0;
}

/* OUTPUT: string2.c
   You created: Jim
*/
```c
/*     FILE: string3.c     */
/* Standard C string library routines
   Note the inclusion of string.h     */
#include <stdio.h>
#include <string.h>

int main( )
{
    char name[81];
    strcpy(name,"Jim");
    printf("You created: %s\n", name);
    return 0;
}

/*     OUTPUT: string3.c     */
   You created: Jim

/*     */
Ex:

```c
/*     FILE: string4.c     */
/* Standard C string library routines */

#include <stdio.h>
#include <string.h>

int main()
{
    char name[81];
    strcpy(name, "Jim");
    strcat(name, " Polzin");
    printf("You created: %s\n", name);
    if(strcmp(name, "jim polzin") == 0)
        printf("%s matches %s\n", name, "jim polzin");
    else
        printf("%s doesn't match %s\n", name, "jim polzin");
    if(strcmp(name, "Jim Polzin") == 0)
        printf("%s matches %s\n", name, "Jim Polzin");
    else
        printf("%s doesn't match %s\n", name, "Jim Polzin");

    printf("\n\nString length = %d\n", strlen(name));
    printf("\n\nSize of name  = %d\n", sizeof(name));
    return 0;
}

/*    OUTPUT: string4.c    */

You created: Jim Polzin
Jim Polzin doesn't match jim polzin
Jim Polzin matches Jim Polzin

String length = 10

Size of name  = 81

*/
```
Ex:

```c
/* FILE: stringRead.c */
/* Reading strings with scanf( ) */

#include <stdio.h>

int main( )
{
    char name[81];
    printf("Enter your name: ");
    scanf("%s", name);

    printf("\n\n");
    printf("You entered: %s\n", name);
    return 0;
}

/* OUTPUT: stringRead.c */

Enter your name: Jim Polzin

You entered: Jim

Enter your name: One Two Three

You entered: One

*/
```
Ex:

```c
/*     FILE: stringRead2.c     */

/* Reading strings with scanf( )
   - it gets more complicated */

#include <stdio.h>

int main()
{
    char name[81];
    int age;

    printf("Enter your name: ");
    scanf("%s", name);

    printf("Enter your age: ");
    scanf("%d", &age);

    printf("\n\n");
    printf("Hello %s\n", name);
    printf("you are %d years old.\n", age);

    return 0;
}

/*    OUTPUT: stringRead2.c

    Enter your name: Jim Polzin
    Enter your age:

    Hello Jim
    you are 1 years old.

*/
```
Ex:

/* FILE: stringRead3.c */

/* Reading strings with scanf() 
- the rough repair */

#include <stdio.h>

int main() {
    char firstName[81];
    char lastName[81];
    int age;

    /* scanf() treats whitespace as a delimiter. So... 
    ... you CAN read each separate piece. */
    printf("Enter your first name: ");
    scanf("%s", firstName);

    printf("Enter your last name: ");
    scanf("%s", lastName);

    printf("Enter your age: ");
    scanf("%d", &age);

    printf("\n\n");
    printf("Hello %s %s
", firstName, lastName);
    printf("you are %d years old.\n", age);

    return 0;
}

/* OUTPUT: stringRead3.c

Enter your first name: Jim
Enter your last name: Polzin
Enter your age: 44

Hello Jim Polzin
you are 44 years old. */
Ex:

```c
/* FILE: stringRead4.c */

/* Reading strings with scanf() 
   - the real fix */
#include <stdio.h>

int main( )
{
    char name[81];
    int age;

    printf("Enter your name: ");
    gets(name); /* gets() knows all about strings
               ... it reads all the input through
               ... the end-of-line. */

    printf("Enter your age: ");
    scanf("%d", &age);

    printf("\n\n");
    printf("Hello %s\n", name);
    printf("you are %d years old.\n", age);

    return 0;
}

/* OUTPUT: stringRead4.c

Enter your name: Jim Polzin
Enter your age: 44

Hello Jim Polzin
you are 44 years old.
*/```
FUNCTIONS

- C allows a block of code to be separated from the rest of the program and named.
- These named blocks of code, or modules, are called functions.
- Functions can be passed information thru a parameter list and can pass back a result thru a return value.
- Any number of parameters can be passed to a function but at most one return value can be produced.
- All the C data types are candidates for parameter types and return types.
- Ideally a function can be treated as a black-box. If you know what to pass it and what it will return you don’t need to know how it works.
- C has a special keyword, void, that is used to explicitly state that there are no parameters or no return value.
Ex:

/*     FILE: aFunction.c     */
/* Computes the cost per sq inch of pizza 

A function example. No parameters, no 
return value. 

*/

#include <stdio.h>
#define PI 3.14159

void instructions(void);   /* Function prototype */

int main( )
{
    int diameter;
    float price, radius, area, pricePerInch;

    instructions( ); /* Call the instructions( ) 
... function */

    printf("What is the price of your pizza: ");
    scanf("%f", &price);

    printf("What is the diameter of your pizza: ");
    scanf("%d", &diameter);

    radius = (float)diameter/2;
    area = PI * radius * radius;
    pricePerInch = price/area;

    printf("Pizza analysis:\n");
    printf("   diameter = %d\n", diameter);
    printf("   radius = %f\n", radius);
    printf("   area = %f\n", area);
    printf("   price = %.2f per sq. inch\n", pricePerInch);

    return 0;
}

void instructions(void)   /* Function definition */
{
    printf("This program will compute the price per \n");
    printf("square inch of a circular pizza. \n\n");

    printf("It will prompt you for the price and the \n");
    printf("diameter of the pizza. Then it will display \n");
    printf("the results of its computations.\n\n");

    printf("Then compare several different price/size \n");
    printf("combinations to determine your best pizza \n");
    printf("value .\n\n");
}

cont...
/* OUTPUT: aFunction.c

This program will compute the price per square inch of a circular pizza.

It will prompt you for the price and the diameter of the pizza. Then it will display the results of its computations.

Then compare several different price/size combinations to determine your best pizza value.

What is the price of your pizza: 10.50
What is the diameter of your pizza: 14
Pizza analysis:
    diameter = 14
    radius = 7.000000
    area = 153.937912
    price = 0.07 per sq. inch

This program will compute the price per square inch of a circular pizza.

It will prompt you for the price and the diameter of the pizza. Then it will display the results of its computations.

Then compare several different price/size combinations to determine your best pizza value.

What is the price of your pizza: 15.50
What is the diameter of your pizza: 18
Pizza analysis:
    diameter = 18
    radius = 9.000000
    area = 254.468796
    price = 0.06 per sq. inch
*/
Ex:

/*     FILE: aFunction2.c     */

/* Computes the cost per sq inch of pizza

Functions with parameter(s) and return
value. */

#include <stdio.h>
#define PI 3.14159

void instructions(void);
float circleArea(float radius);

int main( )
{
    int diameter;
    float price, radius, area, pricePerInch;

    instructions( ); /* Call the instructions( )... function */

    printf("What is the price of your pizza: ");
    scanf("%f", &price);

    printf("What is the diameter of your pizza: ");
    scanf("%d", &diameter);

    radius = (float)diameter/2;
    area = circleArea(radius); /* Call the circleArea( )... function */

    pricePerInch = price/area;

    printf("Pizza analysis:\n");
    printf("    diameter = %d\n", diameter);
    printf("    radius = %f\n", radius);
    printf("    area = %f\n", area);
    printf("    price = %.2f per sq. inch\n", pricePerInch);

    return 0;
}

void instructions(void)
{
    printf("This program will compute the price per \n");
    printf("square inch of a circular pizza. \n\n");

    printf("It will prompt you for the price and the \n");
    printf("diameter of the pizza. Then it will display \n");
    printf("the results of its computations.\n\n");

    printf("Then compare several different price/size \n");
    printf("combinations to determine your best pizza \n");
    printf("value .\n\n");
}

float circleArea(float radius)
{
    float area;

    area = PI * radius * radius;

    return area;
}

cont...
/* OUTPUT: aFunction2.c

This program will compute the price per square inch of a circular pizza.

It will prompt you for the price and the diameter of the pizza. Then it will display the results of its computations.

Then compare several different price/size combinations to determine your best pizza value.

What is the price of your pizza: 10.50
What is the diameter of your pizza: 14
Pizza analysis:
  diameter = 14
  radius = 7.000000
  area = 153.937912
  price = 0.07 per sq. inch

This program will compute the price per square inch of a circular pizza.

It will prompt you for the price and the diameter of the pizza. Then it will display the results of its computations.

Then compare several different price/size combinations to determine your best pizza value.

What is the price of your pizza: 15.50
What is the diameter of your pizza: 18
Pizza analysis:
  diameter = 18
  radius = 9.000000
  area = 254.468796
  price = 0.06 per sq. inch
*/
Ex:

```c
/* FILE: aFunction3.c */

/* Computes the cost per sq inch of pizza 
Functions with parameter(s) and return 
value. */

#include <stdio.h>
define PI 3.14159

void instructions(void);
float circleArea(float radius);
float computePPI(float price, float area);

int main()
{
    float price, radius, area, pricePerInch;
    instructions();
    printf("What is the price of your pizza: ");
    scanf("%f", &price);
    printf("What is the diameter of your pizza: ");
    scanf("%d", &diameter);

    radius = (float)diameter/2;
    area = circleArea(radius);
    pricePerInch = computePPI(price, area);
    printf("Pizza analysis: \\
    diameter = %d\n", diameter);
    printf(" radius = %f\n", radius);
    printf(" area = %f\n", area);
    printf(" price = %.2f per sq. inch\n", pricePerInch);
    return 0;
}

void instructions(void)
{
    printf("This program will compute the price per \n");
    printf("square inch of a circular pizza. \n\n");

    printf("It will prompt you for the price and the \n");
    printf("diameter of the pizza. Then it will display \n");
    printf("the results of its computations.\n\n");

    printf("Then compare several different price/size \n");
    printf("combinations to determine your best pizza \n");
    printf("value .\n\n");
}

float circleArea(float radius)
{
    return PI * radius * radius;
}

float computePPI(float price, float area)
{
    return price/area;
}

cont...
/* OUTPUT: aFunction3.c

This program will compute the price per square inch of a circular pizza.

It will prompt you for the price and the diameter of the pizza. Then it will display the results of its computations.

Then compare several different price/size combinations to determine your best pizza value.

What is the price of your pizza: 10.50
What is the diameter of your pizza: 14
Pizza analysis:
  diameter = 14
  radius = 7.000000
  area = 153.937912
  price = 0.07 per sq. inch

This program will compute the price per square inch of a circular pizza.

It will prompt you for the price and the diameter of the pizza. Then it will display the results of its computations.

Then compare several different price/size combinations to determine your best pizza value.

What is the price of your pizza: 15.50
What is the diameter of your pizza: 18
Pizza analysis:
  diameter = 18
  radius = 9.000000
  area = 254.468796
  price = 0.06 per sq. inch
*/
Ex:

```c
/*     FILE: aFunction4.c     */
/* Computes the cost per sq inch of pizza
Embedded function calls. (This is NOT necessarily the right way to do this.)
main( ) has fewer variables, no need to store what you don't need.
Functions have fewer variables.             */

#include <stdio.h>
define PI 3.14159
void instructions(void);
float circleArea(float radius);
float computePPI(float price, float area);
int main()
{
  int diameter;
  float price;

  instructions( );
  printf("What is the price of your pizza: ");
  scanf("%f", &price);

  printf("What is the diameter of your pizza: ");
  scanf("%d", &diameter);

  printf("Pizza analysis:
";
  printf("price = %.2f per sq. inch
", computePPI(price, circleArea((float)diameter/2)));
  return 0;
}

void instructions(void)
{
  printf("This program will compute the price per \n")
  printf("square inch of a circular pizza. \n\n")
  printf("It will prompt you for the price and the \n")
  printf("diameter of the pizza. Then it will display \n")
  printf("the results of its computations.\n\n")
  printf("Then compare several different price/size \n")
  printf("combinations to determine your best pizza \n")
  printf("value .\n\n");
}

float circleArea(float radius)
{
  return PI * radius * radius;
}

float computePPI(float price, float area)
{
  return price/area;
}

cont...
/* OUTPUT: aFunction4.c

This program will compute the price per square inch of a circular pizza.

It will prompt you for the price and the diameter of the pizza. Then it will display the results of its computations.

Then compare several different price/size combinations to determine your best pizza value.

What is the price of your pizza: 10.50
What is the diameter of your pizza: 14
Pizza analysis:
  price = 0.07 per sq. inch

This program will compute the price per square inch of a circular pizza.

It will prompt you for the price and the diameter of the pizza. Then it will display the results of its computations.

Then compare several different price/size combinations to determine your best pizza value.

What is the price of your pizza: 15.50
What is the diameter of your pizza: 18
Pizza analysis:
  price = 0.06 per sq. inch
*/
LOGICAL, TRUE/FALSE VALUES

• The C definition of true and false is that 0 is false and any non-zero value is true.
• This definition allows some unusual expressions to be used as test conditions.

RELATIONAL OPERATORS

• Relational operators are used quite often to produce the logical value for a conditional statement.

<table>
<thead>
<tr>
<th>operator</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equality</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
</tr>
</tbody>
</table>

LOGICAL OPERATORS

• Logical operators work on logical values, i.e. true and false.

<table>
<thead>
<tr>
<th>operator</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>NOT</td>
</tr>
</tbody>
</table>
LOOPING

- C has three looping constructs, for, while, and do while.
- The while loop is a fundamental pre-test condition loop that repeats as long as the test condition is true.
- The for loop is just a specialized while loop that allows initialization and post-iteration processing to be specified adjacent to the test condition. It is the most commonly used loop in C.
- The do while is just a while loop with the test condition moved to the bottom of the loop. It is a post-test condition loop so the test is executed after each iteration of the loop. (The positioning of the test makes the timing clear.) The main feature of the do while is that it will always execute the body of the loop at least once.

Ex:
/* FILE: for_1.c */
/* for loop example. */
#include <stdio.h>
int main( )
{
    int i;
    for(i = 0; i < 10; i++)
    {
        printf("i = %d\n", i);
    }
    return 0;
}

/* OUTPUT: for_1.c
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
*/
Ex:

```c
/* FILE: for_2.c */
/* for loop example with adjustment for counting from 0. */

#include <stdio.h>

int main( )
{
    int i;
    for(i = 0; i < 10; i++)
    {
        printf("i = %d\n", i + 1);
    }
    return 0;
}

/* OUTPUT: for_2.c
   i = 1
   i = 2
   i = 3
   i = 4
   i = 5
   i = 6
   i = 7
   i = 8
   i = 9
   i = 10
*/
```

Ex:

```c
/* FILE: while_1.c */
/* while loop example. */

#include <stdio.h>

int main( )
{
    int i;
    i = 0;
    while (i < 10)
    {
        printf("i = %d\n", i + 1);
        i++;
    }
    return 0;
}

/* OUTPUT: while_1.c
   i = 1
   i = 2
   i = 3
   i = 4
   i = 5
   i = 6
   i = 7
   i = 8
   i = 9
   i = 10
*/
```
Ex:

```c
/*     FILE: loopChar.c     */
/* Reading characters in a loop.

    Note the space in front of the %c. It causes scanf() to skip leading
    whitespace characters.

    Ctrl/z produces an EOF from the keyboard on a PC.
*/
#include <stdio.h>
int main( )
{
    int ch;
    while(scanf(" %c", &ch) != EOF)
    {
        printf("character = %c\n", ch);
    }
    return 0;
}

/*    OUTPUT: loopChar.c    */
character = a
character = b
character = c
character = d
character = F

INPUT:
    a
    b
c  d
    F
*/```
Ex:
/* FILE: loopChar2.c */
/* Reading characters in a loop with getchar(). */
#include <stdio.h>
int main( )
{
    int ch;
    while((ch = getchar( )) != EOF)
    {
        printf("character = %c\n", ch);
    }
    return 0;
}

/* OUTPUT: loopChar2.c
    character = a
    character =
    character = b
    character =
    character =
    character = c
    character =
    character =
    character =
    character =
    character =
    character =
    character =
    character = F
    character =
    character =
    character =
        INPUT:
        a
        b
cd
    F
*/
Ex:

```c
/*     FILE: loopChar3.c     */
/* Reading characters in a loop with
   getchar( ).
*/
#include <stdio.h>

int main( )
{
    int ch;
    while((ch = getchar( )) != EOF)
    {
        if (ch != '
' && ch != '	' && ch != ' ')
            printf("character = %c\n", ch);
    }
    return 0;
}

/*     OUTPUT: loopChar3.c     */
character = a
character = b
character = c
character = d
character = F

INPUT:

   a
   b
   c d
   F
*/
Ex:

```c
/*     FILE: loopChar4.c     */
/*
   Reading characters in a loop with
   getchar().

   Using the isspace() function to skip
   whitespace.
*/
#include <stdio.h>
#include <ctype.h>

int main( )
{
    int ch;
    while((ch = getchar( )) != EOF)
    {
        if (!isspace(ch))
            printf("character = %c\n", ch);
    }
    return 0;
}

/* OUTPUT: loopChar4.c

c  h  a  r  a  t  e  r  =  a
   c  h  a  r  a  t  e  r  =  b
   c  h  a  r  a  t  e  r  =  c
   c  h  a  r  a  t  e  r  =  d
   c  h  a  r  a  t  e  r  =  F

   INPUT:
   a
   b
   c  d

   F
*/```
• C has a library of pre-defined mathematical functions.

Ex:

```c
/*     FILE: math1.c     */
/* Program to compute the sine function for various values. */
#include <stdio.h>
#include <math.h>

int main( )
{
    double start, end, current, step, value;

    /* Set initial values */
    start = 0.0;
    end = 2 * M_PI;
    step = 0.01;

    /* Loop to compute and display values */
    for(current = start; current <= end; current += step){
        value = sin(current);
        printf("%f\n", value);
    }

    return 0;
}

/*     OUTPUT: math1.c     */
0.000000
0.010000
0.019999
0.029996
0.039989
0.049979
0.059964
0.069943
0.079915
0.089879
0.099833
.
.
.
0.021591
0.011592
0.001593
-0.001593
-0.028407
-0.013806
-0.008407
-0.003806
-0.048388
.
.
.
-0.023183
-0.013185
-0.003185
*/
Ex:

/*     FILE: math2.c     */

/* Program to compute the sine function for
 various values.

 Reads inputs.      */

#include <stdio.h>
#include <math.h>

int main( )
{
    double start, end, current, step, value;

    /* Read initial values */
    scanf("%lf", &start);
    scanf("%lf", &end);
    scanf("%lf", &step);

    /* Loop to compute and display values */
    for(current = start; current <= end; current += step){
        value = sin(current);
        printf("%f\n", value);
    }

    return 0;
}

/*    OUTPUT: math2.c

  0.000000
  0.010000
  0.019999

  0.021591
  0.011592
  0.001593
 -0.008407
 -0.018406
 -0.028404

  0.021591
  0.011592
  0.001593
 -0.008407
 -0.018406
 -0.028404

  0.023183
 -0.013185
 -0.003185
  0.006815
  0.016814
  0.026811

  0.024775
  0.014777
  0.004778

 INPUT:

  0.0
 9.424777
 0.01

*/
Ex:

```c
/*     FILE: math3.c     */
/* Program to compute various values using
the power function.   pow()             */
#include <stdio.h>
#include <math.h>

int main( )
{
    double start, end, current, step, value;
    /* Read initial values */
    scanf("%lf", &start);
    scanf("%lf", &end);
    scanf("%lf", &step);
    /* Loop to compute and display values */
    for(current = start; current <= end; current += step){
        value = pow(current,2.0);
        printf("%f
", value);
    }
    return 0;
}

/*   OUTPUT: math3.c    */
  0.000000
  0.000100
  0.000400
  .
  .
  88.172100
  88.360000
  88.548100
  88.736400

INPUT:
  0.0
  9.4247779
  0.01
/*
```
CONDITIONAL STATEMENTS

- C has two conditional statements and a conditional operator.
- The basic conditional statement in C is the if. An if is associated with a true/false condition. Code is conditionally executed depending on whether the associated test evaluates to true or false.
- The switch statement allows a labeled set of alternatives or cases to be selected from based on an integer value.
- The conditional operator ?: allows a conditional expression to be embedded in a larger statement.

Ex:

```c
/*     FILE: if.c     */
/* if examples. */
#include <stdio.h>

int main( )
{
    int i;
    i = 5;
    if(i > 0)
        printf("%d > 0\n", i);
    i = -2;
    if(i > 0)
        printf("%d > 0\n", i);
    else
        printf("%d <= 0\n", i);
    i = -2;
    if(i > 0)
        printf("%d > 0\n", i);
    else
        if(i == 0) /* Test for equality is == */
            printf("%d == 0\n", i);
        else
            printf("%d < 0\n", i);
    return 0;
}

/*     OUTPUT: if.c     */
5 > 0
-2 <= 0
-2 < 0
*/
```
Ex:

```c
/*     FILE: switch.c     */
/* switch example. */
#include <stdio.h>

int main( )
{
    int ch;
    /* Display menu of choices */
    printf("\tA- append data\n");
    printf("\tD- delete data\n");
    printf("\tR- replace data\n");
    printf("\n\tQ- to quit\n");
    printf("\n\n\tChoice: ");
    ch =getchar( );
    /* Loop to quit on upper or lower case Q */
    while(ch != 'q' && ch != 'Q'){
        switch(ch){
        case 'a':
            printf("Case 'Append' selected.\n", ch);
            break;
        case 'd':
            printf("Case 'Delete' selected.\n", ch);
            break;
        case 'r':
            printf("Case 'Replace' selected.\n", ch);
            break;
        default:
            printf("Invalid choice- '%c'.\n", ch);
            break;
        }
        getchar( );         /* strip trailing newline */
        /* Display menu of choices */
        printf("\n\n\tA- append data\n");
        printf("\tD- delete data\n");
        printf("\tR- replace data\n");
        printf("\n\tQ- to quit\n");
        printf("\n\n\tChoice: ");
    }
    return 0;
}
cont…
/*  OUTPUT: switch.c

A- append data
D- delete data
R- replace data
Q- to quit

Choice: r
Case 'Replace' selected.

A- append data
D- delete data
R- replace data
Q- to quit

Choice: R
Case 'Replace' selected.

A- append data
D- delete data
R- replace data
Q- to quit

Choice: d
Case 'Delete' selected.

A- append data
D- delete data
R- replace data
Q- to quit

Choice: t
Invalid choice- 't'.

A- append data
D- delete data
R- replace data
Q- to quit

Choice: w
Invalid choice- 'w'.

A- append data
D- delete data
R- replace data
Q- to quit

Choice: q
*/
Ex:

```c
/*     FILE: switch2.c     */
/* A function that displays info. */
#include <stdio.h>
void print_menu(void);

int main( )
{
  int ch;

  /* Display menu of choices */
  print_menu( );
  ch = getchar( );

  /* Loop to quit on upper or lower case Q */
  while(ch != 'q' && ch != 'Q'){
    switch(ch){
    case 'a':
    case 'A':
      printf("Case 'Append' selected.\n", ch);
      break;
    case 'd':
    case 'D':
      printf("Case 'Delete' selected.\n", ch);
      break;
    case 'r':
    case 'R':
      printf("Case 'Replace' selected.\n", ch);
      break;
    default:
      printf("Invalid choice- '%c'.\n", ch);
      break;
    }
    getchar( );       /* strip trailing newline */
  }

  /* Display menu of choices */
  printf("\n\n");
  print_menu( );
  ch = getchar( );
}

return 0;
}

void print_menu(void)
{
  printf("\tA- append data\n");
  printf("\tD- delete data\n");
  printf("\tR- replace data\n");
  printf("\n\tQ- to quit\n");
  printf("\n\n\n\tChoice: ");
  return;
}

cont…
```
/* OUTPUT: switch2.c

A- append data
D- delete data
R- replace data
Q- to quit

Choice: r
Case 'Replace' selected.

A- append data
D- delete data
R- replace data
Q- to quit

Choice: D
Case 'Delete' selected.

A- append data
D- delete data
R- replace data
Q- to quit

Choice: q
*/
Ex:

#include <stdio.h>

void printMenu(void);
void printStatus(int, int);

int main( )
{
    int x=0, y=0;
    int ch;

    printStatus(x,y);  /* Print current x,y */
    /* Display menu of choices */
    printMenu( );
    ch =getchar( );

    /* Loop to quit on upper or lower case Q */
    while(ch != 'q' && ch != 'Q'){
        switch(ch){
        case 'u':
        case 'U':
            printf("Case 'Up' selected.\n", ch);
            y++;
            break;
        case 'd':
        case 'D':
            printf("Case 'Down' selected.\n", ch);
            y--;
            break;
        case 'l':
        case 'L':
            printf("Case 'Left' selected.\n", ch);
            x--;
            break;
        case 'r':
        case 'R':
            printf("Case 'Right' selected.\n", ch);
            x++;
            break;
        default:
            printf("Invalid choice- '%c'.\n", ch);
            break;
        }
        getchar( ); /* strip trailing newline */
        printStatus(x,y);  /* Print current x,y */
        /* Display menu of choices */
        printf("\n\n");
        printMenu( );
        ch =getchar( );
    }

    return 0;
}

cont...
void printMenu(void)
{
    printf("\tU- Increase y\n");
    printf("\tD- Decrease y\n");
    printf("\tL- Decrease x\n");
    printf("\tR- Increase x\n");

    printf("\n\tQ- to quit\n");
    printf("\n\tChoice: ");

    return;
}

void printStatus(int x, int y)
{
    printf("Current location: x = %d, y = %d \n", x, y);
    return;
}

/* OUTPUT: tracker.c

Current location: x = 0, y = 0
U- Increase y
D- Decrease y
L- Decrease x
R- Increase x
Q- to quit

Choice: u
Case 'Up' selected.
Current location: x = 0, y = 1

U- Increase y
D- Decrease y
L- Decrease x
R- Increase x
Q- to quit

Choice: U
Case 'Up' selected.
Current location: x = 0, y = 2

U- Increase y
D- Decrease y
L- Decrease x
R- Increase x
Q- to quit

Choice: r
Case 'Right' selected.
Current location: x = 1, y = 2

cont...
U- Increase y
D- Decrease y
L- Decrease x
R- Increase x
Q- to quit

Choice: r
Case 'Right' selected.
Current location: x = 2, y = 2

U- Increase y
D- Decrease y
L- Decrease x
R- Increase x
Q- to quit

Choice: l
Case 'Left' selected.
Current location: x = 1, y = 2

U- Increase y
D- Decrease y
L- Decrease x
R- Increase x
Q- to quit

Choice: l
Case 'Left' selected.
Current location: x = 0, y = 2

U- Increase y
D- Decrease y
L- Decrease x
R- Increase x
Q- to quit

Choice: l
Case 'Left' selected.
Current location: x = -1, y = 2

U- Increase y
D- Decrease y
L- Decrease x
R- Increase x
Q- to quit

Choice: q

*/
Ex:

/*     FILE: cond_op.c     */
/* conditional operator example. */

#include <stdio.h>

int main( )
{
    int i;

    /* Loop to read integers and quit on non-integer */
    printf("Enter an integer (q to quit): ");
    while(scanf("%d", &i) == 1){ /* scanf returns # of items read. */
        printf("Value entered = %d, absolute value = %d\n", i, i<0?-i:i);
        printf("Enter an integer (q to quit): ");
    }

    return 0;
}

/*    OUTPUT: cond_op.c     */

Enter an integer (q to quit): 7
Value entered = 7, absolute value = 7
Enter an integer (q to quit): -7
Value entered = -7, absolute value = 7
Enter an integer (q to quit): 13
Value entered = 13, absolute value = 13
Enter an integer (q to quit): -27
Value entered = -27, absolute value = 27
Enter an integer (q to quit): q

*/
FUNCTIONS – THE DETAILS

- C allows a block of code to be separated from the rest of the program and named.
- These blocks of code or modules are called functions.
- Functions can be passed information thru a parameter list. Any number of parameters can be passed to a function.
- Functions can pass back a result thru a return value. At most one return value can be produced.
- All the C data types are candidates for parameter types and return types.
- Ideally a function can be treated as a black-box. If you know what to pass it and what it will return; you don’t need to, or sometimes want to, know how it works.
- C has a special keyword, `void`, that is used to explicitly state that there are no parameters or no return type.
- Using a function takes place in three steps:
  - Defining the function
    The definition is the C code that completely describes the function, what it does, what formal parameters it expects, and what it’s return value and type will be.
  - Calling the function
    When the function is needed to do its work, it is “called” by its name and supplied actual parameters for the formal parameters it requires. Its return value is used if provided and needed.
  - Prototyping the function
    A prototype provides the communication information for the function, the parameter types and return value, to the compiler. This allows the compiler to more closely scrutinize your code. (This is a very, very good thing.) A prototype looks like the first line of the function definition, it identifies the parameter types and the return type of the function. A prototype should be placed within the source code at a point before the call is made. Often prototypes are placed near
the top of the source code file. More often, the prototypes are placed into a .h file and \#include is used to include them in the source code file.
Ex:

```c
#include <stdio.h>

void print_menu(void);  /* Prototype: - no parameters  
                        - no return value */

int main( )
{
    int ch;

    /* Display menu of choices */
    print_menu( );
    ch =getchar( );

    /* Loop to quit on upper or lower case Q */
    while(ch != 'q' && ch != 'Q'){
        switch(ch){
            case 'a':
                printf("Case 'Append' selected.\n", ch);
            break;
            case 'd':
                printf("Case 'Delete' selected.\n", ch);
            break;
            case 'r':
                printf("Case 'Replace' selected.\n", ch);
            break;
            case 'A':
                printf("Case 'Append' selected.\n", ch);
            break;
            case 'D':
                printf("Case 'Delete' selected.\n", ch);
            break;
            case 'R':
                printf("Case 'Replace' selected.\n", ch);
            break;
            default:
                printf("Invalid choice- '%c'.\n", ch);
            break;
        }
        getchar( );         /* strip trailing newline */
        /* Display menu of choices */
        printf("\n\n");
    print_menu( );
    ch =getchar( );
    }

    return 0;
}

void print_menu(void)
{
    printf("\tA- append data\n");
    printf("\tD- delete data\n");
    printf("\tR- replace data\n");
    printf("\tQ- to quit\n");
    printf("\n\n\tChoice: ");
    return;
}

cont…
```
/*    OUTPUT: switch3.c

    A- append data
    D- delete data
    R- replace data
    Q- to quit

    Choice: r
    Case 'Replace' selected.

    A- append data
    D- delete data
    R- replace data
    Q- to quit

    Choice: D
    Case 'Delete' selected.

    A- append data
    D- delete data
    R- replace data
    Q- to quit

    Choice: q
*/
Ex:
/* FILE: binary.c */
/* A couple functions that get passed a value,
  display some output and return nothing. */

#include <stdio.h>

void print_binary_int(unsigned int);
void print_binary_char(unsigned char); /* Prototypes:
  - no return values
  - one parameter each */

int main( )
{
  int first;
  char second;

  printf("Enter an integer: ");
  scanf("%d", &first);

  printf("Enter a character: ");
  scanf("%c", &second);

  printf("Integer %d = ", first);
  print_binary_int(first);
  printf("\n\n");
  printf("Character %c = %d = ", second, second);
  print_binary_char(second);
  printf("\n\n");
  return 0;
}

void print_binary_int(unsigned int x)
{
  unsigned int divisor = 2147483648U;

  while(divisor > 0){
    if(divisor <= x){
      printf("1");
      x = x - divisor;
    } else
      printf("0");
    divisor = divisor/2;
  }

  return;
}

cont…
void print_binary_char(unsigned char c)
{
    unsigned char divisor = 128;

    while(divisor > 0){
        if(divisor <= c){
            printf("1");
            c = c - divisor;
        }
        else
            printf("0");

        divisor = divisor/2;
    }

    return;
}

/*
 * OUTPUT: binary.c
 *
 Enter an integer: 127
 Enter a character: A
 Integer 127 = 00000000000000000000000001111111
 Character A = 65 = 01000001
 */
Ex:

/*     FILE: average.c     */
/*@     A function that is passed two values and returns
     one too. */

#include <stdio.h>

double average(int, int); /* Parameters: 2 ints
     Return value: a double */

int main( )
{
    int first, second;
    double avg;

    printf("Enter first integer: ");
    scanf("%d", &first);

    printf("Enter second integer: ");
    scanf("%d", &second);

    avg = average(first, second);

    printf("Average = %f\n", avg);
    return 0;
}

double average(int x, int y)
{
    double temp;
    temp = (x + y)/2.0;
    return temp;
}

/*     OUTPUT: average.c     */
    Enter first integer: 1
    Enter second integer: 2
    Average = 1.500000
*/
Ex:

/*   FILE: average2.c    */

/* A function that is passed two values and returns
 one too. Function average( ) with less overhead. */

#include <stdio.h>

double average(int, int);    /* Parameters: 2 ints
Return value: a double */

int main( )
{
    int first, second;
    double avg;

    printf("Enter first integer: ");
    scanf("%d", &first);

    printf("Enter second integer: ");
    scanf("%d", &second);

    avg = average(first, second);

    printf("Average = %.1f\n", avg);
    return 0;
}

double average(int x, int y)
{
    return (x + y)/2.0;
}

/* OUTPUT: average2.c

Enter first integer: 1
Enter second integer: 2
Average = 1.500000

*/
Ex:

```c
/*     FILE: recursion.c     */
/* Recursive function to display octal. */

#include <stdio.h>
void printOctal(int x);

int main( )
{
    int value = 71;
    /* Display value in decimal */
    printf("Value = %d decimal\n", value);
    /* Display value in octal */
    printf("Value = ");
    printOctal(value);
    printf(" octal\n");
    return 0;
}

void printOctal(int x)  /* Recursive - printOctal( ) calls itself. */
{
    if(x < 0){
        printf("-"),
        printOctal(-x);
    } else {
        if (x > 7)
            printOctal(x/8);
        printf("%d", x%8);
    }
    return;

    /*    OUTPUT: recursion.c
     Value = 71 decimal
     Value = 107 octal
     */
```

/*    */
Ex:

```c
/*     FILE: recursion2.c     */

/* Recursive functions to display octal and hexadecimal. */

#include <stdio.h>
void printOctal(int x);
void printHex(int x);

int main( )
{
    int value = 175;
    /* Display value in decimal */
    printf("Value = %d decimal\n", value);
    /* Display value in octal */
    printf("Value = ");
    printOctal(value);
    printf(" octal\n");
    /* Display value in hexadecimal */
    printf("Value = ");
    printHex(value);
    printf(" hexadecimal\n");
    return 0;
}

void printOctal(int x) /* Recursive - printOctal( ) calls */
{
    /* ... itself. */
    if(x < 0){
        printf("-");
        printOctal(-x);
    } else {
        if (x > 7)
            printOctal(x/8);
        printf("%d", x%8);
    }
    return;
}

void printHex(int x) /* Recursive - printHex( ) calls */
{
    /* ... itself. */
    if(x < 0){
        printf("-");
        printHex(-x);
    } else {
        if (x > 15)
            printHex(x/16);
        if((x%16) < 10)
            printf("%d", x%16);
        else
            printf("%c", 'A' + x%16 - 10);
    }
    return;
}

/*    OUTPUT: recursion2.c    
   Value = 175 decimal
   Value = 257 octal
   Value = AF hexadecimal
*/
```
POINTERS

- A pointer in C is a data type that can store the address of some other storage location.
- Pointers are used when a variable’s location is of interest and not just its value.
- A pointer is declared by using a data type followed by an asterisk, *.
- To produce the address of a variable, apply the address-of operator, & to a variable.
- Since the contents of a pointer variable are an address you need to dereference the pointer to access the value it references. That will be the value at the address the pointer contains, or the value the pointer references.

Ex:

```c
/*     FILE: pointer.c     */
/* A pointer variable. */
#include <stdio.h>

int main( )
{
    int* ptr;
    int i;
    i = 7;
    ptr = &i;    /* ptr now knows where i is. */
    printf("i = %d and is at address %p\n", i, &i);
    printf("i = %d and is at address %p\n", *ptr, ptr);
    return 0;
}

/*    OUTPUT: pointer.c    */
    i = 7 and is at address 0022FF68
    i = 7 and is at address 0022FF68
*/
```
Ex:

```c
/*     FILE: funcPt1.c     */
/* swap( ) function that fails due to pass by value. */

#include <stdio.h>
void swap(int x, int y);

int main()
{
    int x, y;
    x = 3;
    y = 5;

    printf("Before swap, x = %d, y = %d\n", x, y);
    swap(x, y);
    printf("After swap, x = %d, y = %d\n", x, y);
    return 0;
}

void swap(int x, int y)
{
    int temp;

    printf("In swap before: %d %d\n", x, y);
    temp = x;
    x = y;
    y = temp;

    printf("In swap after: %d %d\n", x, y);
    return;
}

/*    OUTPUT: funcPt1.c    */

Before swap, x = 3, y = 5
In swap before: 3 5
In swap after: 5 3
After swap, x = 3, y = 5
*/
```
Ex:

/*       FILE: funcPt2.c       */
/* swap( ) function that works due to pointers */

#include <stdio.h>
void swap(int* x, int* y);

int main( )
{
    int x, y;
    x = 3;
    y = 5;
    printf("Before swap, x = %d, y = %d\n", x, y);
    swap(&x,&y);
    printf("After swap, x = %d, y = %d\n", x, y);
    return 0;
}

void swap(int* x, int* y)
{
    int temp;
    printf("In swap before: %d %d\n", *x, *y);
    temp = *x;
    *x = *y;
    *y = temp;
    printf("In swap after: %d %d\n", *x, *y);
    return;
}

/*       OUTPUT: funcPt2.c       */

Before swap, x = 3, y = 5
In swap before: 3 5
In swap after: 5 3
After swap, x = 5, y = 3

*/
ARRAYS

- C allows easy creation and access to sets of storage locations with arrays.
- An array is set of storage locations all referred to by the same name. Each individual location is uniquely identified by the array name and an index value, or offset, into the array.
- C arrays are indexed beginning with the value 0 for the index of the first location and ending with the size-1 for the index of the last location.
- Since the only difference between successive locations in an array is the index value, the computer can be used to generate the index values. This allows an entire array to be processed with very little programming effort.
- An array is homogeneous, that is all elements are of the same data type.
Ex:

```c
/*     FILE: array1.c     */

/* A simple array example. 
   Stores values and displays them. */

#include <stdio.h>

main( )
{
    int ar[10];
    int i;

    for(i=0; i<10; i++)
        ar[i] = 23 - i;

    for(i=0; i<10; i++)
        printf("%d\n", ar[i]);

    return 0;
}

/*    OUTPUT: array1.c
   23
   22
   21
   20
   19
   18
   17
   16
   15
   14

   */
```
Ex:

/* FILE: array2.c */

/* A simple array example.
Stores values and displays them.

The output is a little fancier. */

#include <stdio.h>

main( )
{
    int ar[10];
    int i;
    for(i=0; i<10; i++)
        ar[i] = 23 - i;

    for(i=0; i<10; i++)
        printf("ar[%d] = %d\n", i, ar[i]);

    return 0;
}

/* OUTPUT: array2.c

ar[0] = 23
ar[1] = 22
ar[2] = 21
ar[3] = 20
ar[4] = 19
ar[5] = 18
ar[6] = 17
ar[7] = 16
ar[8] = 15
ar[9] = 14
*/
Ex:

```c
/*     FILE: array3.c     */
/* Reads values and displays them. */
#include <stdio.h>

main( )
{
    int ar[10];
    int i;

    for(i=0; i<10; i++)
    {
        printf("Enter value %d of %d: ", i+1, 10);
        scanf("%d", &ar[i]);
    }

    for(i=0; i<10; i++)
        printf("%d
", ar[i]);

    return 0;
}

/*    OUTPUT: array3.c

Enter value 1 of 10: 1
Enter value 2 of 10: 2
Enter value 3 of 10: 3
Enter value 4 of 10: 4
Enter value 5 of 10: 5
Enter value 6 of 10: 6
Enter value 7 of 10: 7
Enter value 8 of 10: 8
Enter value 9 of 10: 9
Enter value 10 of 10: 10

1
2
3
4
5
6
7
8
9
10
*/
```
Ex:

/* FILE: array4.c */
/* Reads in values, computes their average, and displays them. */

#include <stdio.h>

main( )
{
    int ar[10];
    int i, sum;
    double avg;

    for(i=0; i<10; i++)
    {
        printf("Enter value %d of %d: ", i+1, 10);
        scanf("%d", &ar[i]);
    }

    sum = 0;
    for(i=0; i<10; i++)
        sum = sum + ar[i];

    avg = (double)sum / 10;
    printf("avg = %f\n", avg);

    return 0;
}

/* OUTPUT: array4.c */
Enter value 1 of 10: 4
Enter value 2 of 10: 4
Enter value 3 of 10: 4
Enter value 4 of 10: 4
Enter value 5 of 10: 4
Enter value 6 of 10: 5
Enter value 7 of 10: 5
Enter value 8 of 10: 5
Enter value 9 of 10: 5
Enter value 10 of 10: 5
avg = 4.500000
/*
Ex:
/*     FILE: array5.c     */
/* Reads in values, computes their average, and displays them. 
Uses a defined constant to simplify future changes and 
increase readability. */
#include <stdio.h>
define SIZE 10
main( )
{
    int ar[SIZE];
    int i, sum;
    double avg;

    for(i=0; i<SIZE; i++)
    {
        printf("Enter value %d of %d: ", i+1, SIZE);
        scanf("%d", &ar[i]);
    }

    sum = 0;
    for(i=0; i<SIZE; i++)
    {
        sum = sum + ar[i];
    }

    avg = (double)sum / SIZE;
    printf("avg = %f\n", avg);

    return 0;
}

/*    OUTPUT: array5.c
Enter value 1 of 10: 4
Enter value 2 of 10: 4
Enter value 3 of 10: 4
Enter value 4 of 10: 4
Enter value 5 of 10: 4
Enter value 6 of 10: 5
Enter value 7 of 10: 5
Enter value 8 of 10: 5
Enter value 9 of 10: 5
Enter value 10 of 10: 5
avg = 4.500000
*/
Ex:

```c
/*     FILE: max_cnt.c     */

/*
Loads an array with up to SIZE values.
Finds the max and the count of values
greater than 90.
*/
#include <stdio.h>
#define SIZE 50

int main() {
    int scores[SIZE];
    int i, n, max, a_count;

    /* Get number of values to read */
    printf("Please enter number of scores (%d or less): ", SIZE);
    scanf("%d", &n);

    /* Validate number entered by user. */
    if (n<=SIZE && n>0) {
        /* Read score values into array */
        for(i=0; i<n; i++)
            {printf("Enter value %d of %d: ", i+1, n);
             scanf("%d", &scores[i]);
            }

        /* Find maximum of values read. */
        max = scores[0];
        for(i=1; i<n; i++)
            {if (scores[i] > max)
             max = scores[i];
            }

        printf("Max score = %d\n", max);

        /* Count number of A's, scores greater than 90 */
        a_count = 0;
        for(i=0; i<n; i++)
            {if (scores[i] > 90)
             a_count++;
            }

        printf("A's = %d\n", a_count);
    }

    return 0;
}

/*    OUTPUT: max_cnt.c

Please enter number of scores (50 or less): 4
Enter value 1 of 4: 75
Enter value 2 of 4: 85
Enter value 3 of 4: 95
Enter value 4 of 4: 92
Max score = 95
A's = 2
*/
```
Ex:

```c
/*     FILE: sort1.c     */

/* An example of sorting with selection sort. */

#include <stdio.h>
#define SIZE 10
main( )
{
    int ar[SIZE], pass, item, position, temp;
    for(item=0; item<SIZE; item++)  /* load array with values */
        ar[item] = item*10;
    printf("\nOriginal array:\n");
    for(item=0; item<SIZE; item++)  /* display values in array */
        printf("ar[%d] = %d\n", item, ar[item]);
    /* Selection-sort the values read in. */
    for(pass=0; pass<SIZE-1; pass++){
        position = pass;
        for(item=pass+1; item<SIZE; item++)
            if (ar[position] < ar[item])
                position = item;
        if(pass != position){
            temp = ar[pass];
            ar[pass] = ar[position];
            ar[position] = temp;
        }
    }
    printf("\nSorted array:\n");
    for(item=0; item<SIZE; item++)  /* display values in array */
        printf("ar[%d] = %d\n", item, ar[item]);
    return 0;
}

/*    OUTPUT: sort1.c

Original array:
    ar[0] = 0
    ar[1] = 10
    ar[2] = 20
    ar[3] = 30
    ar[4] = 40
    ar[5] = 50
    ar[6] = 60
    ar[7] = 70
    ar[8] = 80
    ar[9] = 90

Sorted array:
    ar[0] = 90
    ar[1] = 80
    ar[2] = 70
    ar[3] = 60
    ar[4] = 50
    ar[5] = 40
    ar[6] = 30
    ar[7] = 20
    ar[8] = 10
    ar[9] = 0
*/
```
Ex:

```c
/*     FILE: select.c     */
/* Loads an array with up to 50 values.  
   Sorts the values into descending order. */

#include <stdio.h>
define SIZE 50

int main( )
{
    int scores[SIZE];
    int i, n, pass, item, position, temp;

    /* Get number of values to read */
    printf("Please enter number of scores (%d or less): ", SIZE);
    scanf("%d", &n);

    /* Validate number entered by user. */
    if (n<=SIZE & n>0){
        /* Read score values into array */
        for(i=0; i<n; i++)
            {          
            printf("Enter value %d of %d: ", i+1, n);
            scanf("%d", &scores[i]);
        }

        /* Selection-sort the values read in. */
        for(pass=0; pass<n-1; pass++)
            
            position = pass;
        for(item=pass+1; item<n; item++)
            if (scores[position] < scores[item])
                position = item;
        if(pass != position){
            temp = scores[pass];
            scores[pass] = scores[position];
            scores[position] = temp;
        }

        /* Display scores in sorted order */
        printf("\n\nThe scores in order.\n");
        for(i=0; i<n; i++)
            printf("%d- %d\n", i+1, scores[i]);
    }

    return 0;
}

/*    OUTPUT: select.c

Please enter number of scores (50 or less): 6
Enter value 1 of 6: 75
Enter value 2 of 6: 42
Enter value 3 of 6: 88
Enter value 4 of 6: 37
Enter value 5 of 6: 99
Enter value 6 of 6: 92

The scores in order.
 1- 99
 2- 92
 3- 88
 4- 75
 5- 42
 6- 37
*/
```
Ex:

/*     FILE: arrayString.c     */
/* Strings are arrays */

#include <stdio.h>
#include <string.h>

int main( )
{
    char name[81];
    strcpy(name,"Jim");
    strcat(name," Polzin");
    printf("You created: %s\n", name);
    name[6] = 'L';
    printf("It was changed to: %s\n", name);
    return 0;
}

/*    OUTPUT: arrayString.c
    You created: Jim Polzin
    It was changed to: Jim PoLzin
    */
Ex:

/*     FILE: arrayString2.c     */
/* Strings are arrays */

#include <stdio.h>
#include <string.h>

int main( )
{
    int i;
    char name[81];

    strcpy(name,"Jim");
    strcat(name," Polzin");

    printf("You created: ");
    for(i=0; name[i] != '\0'; i++)
        putchar(name[i]);
    putchar('\\n');
    return 0;
}

/*    OUTPUT: arrayString2.c

    You created: Jim Polzin

*/
Ex:

```c
/* FILE: arrayString3.c */
/* Strings as parameters */
#include <stdio.h>
#include <string.h>

void myPuts(char[]);
void myStrcpy(char[], char[]);

int main()
{
    int i;
    char name[81];

    myStrcpy(name,"Jim");
    strcat(name," Polzin");
    printf("You created: ");
    myPuts(name);
    return 0;
}

void myPuts(char s[])
{
    int i;
    for(i=0; s[i] != '\0'; i++)
        putchar(s[i]);
    putchar('\n');
    return;
}

void myStrcpy(char dest[], char src[])
{
    int i;
    for(i=0; src[i] != '\0'; i++)
        dest[i] = src[i];
    dest[i] = '\0';
    return;
}

/* OUTPUT: arrayString3.c
   You created: Jim Polzin */
```
Ex:

```c
/*     FILE: arrayString4.c     */
/* Strings as parameters
   myStrcpy - altered       */
#include <stdio.h>
#include <string.h>
void myPuts(char [ ]);
void myStrcpy(char [ ], char[ ]);

int main( )
{
    int i;
    char name[81];
    myStrcpy(name,"Jim");
    strcat(name," Polzin");
    printf("You created: ");
    myPuts(name);
    return 0;
}

void myPuts(char s[ ])
{
    int i;
    for(i=0; s[i] != '\0'; i++)
        putchar(s[i]);
    putchar('\n');
    return;
}

void myStrcpy(char dest[ ], char src[ ])  /* C style, streamlined! */
{
    int i;
    for(i=0; (dest[i]=src[i]) != '\0'; i++)
    {
    }
    return;
}

/*    OUTPUT: arrayString4.c
   You created: Jim Polzin
 */
```
ARRAYS AND POINTERS

• With a 1-D array the array name is the address of the first thing in the array.

\[
\text{int } x[3];
\]

\[
\begin{array}{c|c|c}
0 & x[0] & x + 0 \\
1 & x[1] & x + 1 \\
2 & x[2] & x + 2 \\
\end{array}
\]

- \(x\) - address of the first thing in the integer array
- \(x + 1\) - address of the second thing in the integer array
- \(x + 2\) - address of the third thing in the integer array

• With a 1-D array dereferencing once, or indexing into the array once using the array access operator, gives a value in the array.

\[
\begin{array}{c|c|c|c}
0 & x[0] & \ast(x + 0) & \ast \ast \ast \ast \ast \ast \\
1 & x[1] & \ast(x + 1) & \ast \ast \ast \ast \ast \ast \\
2 & x[2] & \ast(x + 2) & \ast \ast \ast \ast \ast \ast \\
\end{array}
\]

- \(\ast x \equiv x[0]\) - value of the first element in the array
- \(\ast(x + 1) \equiv x[1]\) - value of the second element in the array
- \(\ast(x + 2) \equiv x[2]\) - value of the third element in the array
Ex:

/*     FILE: pointer2.c     */

/* Array names are addresses. */
#include <stdio.h>

int main( )
{
    int* ptr;
    int i;
    int ar[5];
    for (i=0; i<5; i++)
        ar[i] = i+1;
    ptr = ar;  /* ptr now knows where ar is. */
    printf("ar[0] = %d and is at address %p\n", ar[0], ar);
    printf("*ptr = %d and is at address %p\n", *ptr, ptr);
    for (i=0; i<5; i++)
        printf("ar[%d] = %d and is at address %p\n", i, ar[i], ar+i);
    printf("\n");
    for (i=0; i<5; i++)
        printf("*(ptr+i) = %d and is at address %p\n", *(ptr+i), ptr+i);
    return 0;
}

/*    OUTPUT: pointer2.c

   ar[0] = 1 and is at address 0022FF38
   *ptr = 1 and is at address 0022FF38
   ar[0] = 1 and is at address 0022FF38
   ar[1] = 2 and is at address 0022FF3C
   ar[2] = 3 and is at address 0022FF40
   ar[3] = 4 and is at address 0022FF44
   ar[4] = 5 and is at address 0022FF48
   *(ptr+i) = 1 and is at address 0022FF38
   *(ptr+i) = 2 and is at address 0022FF3C
   *(ptr+i) = 3 and is at address 0022FF40
   *(ptr+i) = 4 and is at address 0022FF44
   *(ptr+i) = 5 and is at address 0022FF48

*/
Ex:

```c
/*     FILE: array6.c     */
/* Passing an array to a function.
   Array name/pointer equivalence.*/

#include <stdio.h>
#define SIZE 5
void print_array(int a[], int length);
void print_array2(int* a, int length);

main( )
{
    int ar[SIZE];
    int i;
    for(i=0; i<SIZE; i++)
    {
        printf("Enter value %d of %d: ", i+1, SIZE);
        scanf("%d", ar + i);
    }
    printf("\n");
    print_array(ar, SIZE);
    printf("\n");
    print_array2(ar, SIZE);
    return 0;
}
void print_array(int a[], int length)
{
    int i;
    for(i=0; i<length; i++)
    {
        printf("a[%d] = %d\n", i, a[i]);
    }
    return;
}
void print_array2(int* a, int length)
{
    int i;
    for(i=0; i<length; i++)
    {
        printf("a[%d] = %d\n", i, a[i]);
    }
    return;
}
/*    OUTPUT: array6.c
 Enter value 1 of 5: 11
 Enter value 2 of 5: 22
 Enter value 3 of 5: 33
 Enter value 4 of 5: 44
 Enter value 5 of 5: 55
 a[0] = 11
 a[1] = 22
 a[2] = 33
 a[3] = 44
 a[4] = 55
 a[0] = 11
 a[1] = 22
 a[2] = 33
 a[3] = 44
 a[4] = 55
 */
```
Ex:

```c
/*     FILE: string5.c     */
/* Passing a string to a function - pointer */
#include <stdio.h>
#include <string.h>

void myPuts(char *str);

int main( )
{
    char name[81];

    strcpy(name, "Jim");
    strcat(name, " Polzin");

    printf("You created: %s\n", name);
    /* Another way */
    myPuts("You created: ");
    myPuts(name);
    myPuts("\n");

    return 0;
}

void myPuts(char *str)
{
    while(*str != '\0')
    {
        putchar(*str);
        str++;  
    }
    return;
}

/*     OUTPUT: string5.c     */
You created: Jim Polzin
You created: Jim Polzin
```

```
Ex:

```
/*     FILE: string6.c     */
/* Passing a string to a function - pointers */
#include <stdio.h>
#include <string.h>
void myPuts(char *str);
void myStrcpy(char *dest, char *src);
char * myStrcpy2(char *dest, char *src);

int main( )
{
    char name[81];
    myStrcpy(name,"Jim Polzin");
    myPuts("You created: ");
    myPuts(name);
    myPuts("\n");
    myPuts("You created: ");
    myPuts(myStrcpy2(name,"C Programming Language."));
    myPuts("\n");
    return 0;
}
void myPuts(char *str)
{
    while(*str)
        putchar(*str++);
    return;
}
void myStrcpy(char *dest, char *src)
{
    while(*src != '\0'){
        *dest = *src;
        dest++; src++;
    }
    *dest = *src;
    return;
}
char * myStrcpy2(char *dest, char *src)
{
    char * ret = src;
    while(*dest++ = *src++);
    return ret;
}

/*     OUTPUT: string6.c     */
You created: Jim Polzin
You created: C Programming Language.
*/
```
BASIC MULTI-DIMENSIONAL ARRAYS

• Basically, a 2-dimensional array can be thought of as a 2-D table of storage locations. The first index determines a row in the table and the second the column in that row.

\[
\text{int } x[2][3]; \quad /* \text{a 2-D set of ints }*/
\]

• To access a particular location 2 index values must be provided.

\[
\begin{array}{cccc}
0 & [ ] & x[0][0] & x[0][1] & x[0][2] \\
1 & [ ] & x[1][0] & x[1][1] & x[1][2] \\
\end{array}
\]

• A 3-dimensional array can be thought of as a 3-D set of storage locations. The first index determines a layer in the set, the second, a row in that layer, and the third, a particular element in that layer and row.

\[
\text{int } x[2][3][4]; \quad /* \text{a 3-D set of ints }*/
\]

cont…
• Equivalently a 2-D array can be thought of as a set of 1-D arrays. Each row being a 1-D array of values.

    int x[2][3];        /* a set of 2, 1-D arrays containing 3 ints */

    x[ ][ ][0] [ ][1] [ ][2]
        [0][ ]             [1][ ]

• A 3-D array can be thought of as a set of 2-D arrays. Each layer being a 2-D array of values.

    int x[2][3][4];        /* a set of 2, 3x4 2-D arrays */
    x[1][ ][ ]

    x[0][ ][ ]

• This conceptualization of arrays, as sets of sets, allows us to easily comprehend arrays of greater-than 3 dimensions. It also is a better model for understanding the relationship of pointers with n-dimensional arrays.

    int x[2][2][3][4];        /* a set of 2, 2x3x4 3-D arrays */

    x[1][ ][ ][ ]
        x[1][1][ ][ ]
        x[1][0][ ][ ]

    x[0][ ][ ][ ]
        x[0][1][ ][ ]
        x[0][0][ ][ ]
MULTIDIMENSIONAL ARRAYS AND POINTERS

- With a 2-D array the array name is the address of the first thing in the array. In this case the first thing in the array is a 1-D array:

  ```c
  int x[2][3]; /* a set of 2 1-D arrays containing 3 ints */
  ```

- So the array name of this [2][3] array is the address of a set of 3 integers.

  - `x` - address of the first array in the set of 2, 3 integer arrays.
  - `x + 1` - address of the second array in the set of 2, 3 integer arrays.

  ```c
  x == x[0]
  // address of the first element in the first array
  // (same as a 1-D array now that x has been dereferenced once.)
  ```

  ```c
  *(x + 1) == x[1]
  // address of the first element in the second array
  // (same as a 1-D array now that x has been dereferenced once.)
  ```

  ```c
  **(*(x + 0) + 1) == x[0][1]
  // - x + 0 is the address of the first array
  // - *(*(x + 0) + 1) is the address of the second element in the first array
  ```

  ```c
  **(*(x + 1) + 2) == x[1][2]
  // - x + 1 is the address of the second array
  // - *(x + 1) is the address of the first element in the second array
  ```

  With a 2-D array, dereferencing once, or indexing into the array once using the array access operator, gives an address. Dereferencing twice, or indexing into the array twice using the array access operator, gives a value from the array.

  Unless you dereference or index as many times as you have dimensions in an array, you still have an address, just a different kind of address.
Ex:

/* FILE: MultiArray.c */

/* Program with multiple strings containing the
   names of the months. */

#include <stdlib.h>

int main( )
{
    char jan[8] = "January";
    char feb[9] = "February";
    char mar[6] = "March";
    char apr[6] = "April";
    char may[4] = "May";
    char jun[5] = "June";
    char jul[5] = "July";
    char aug[7] = "August";
    char sep[10] = "September";
    char oct[8] = "October";
    char nov[9] = "November";
    char dec[9] = "December";

    printf("The months of the year: \n");
    printf("%s \n", jan);
    printf("%s \n", feb);
    printf("%s \n", mar);
    printf("%s \n", apr);
    printf("%s \n", may);
    printf("%s \n", jun);
    printf("%s \n", jul);
    printf("%s \n", aug);
    printf("%s \n", sep);
    printf("%s \n", oct);
    printf("%s \n", nov);
    printf("%s \n", dec);

    return 0;
}

/* OUTPUT: MultiArray.c

   The months of the year:
   January
   February
   March
   April
   May
   June
   July
   August
   September
   October
   November
   December

*/
Ex:

/*     FILE: MultiArray2.c     */

/* Program with multiple strings containing the names of the months. */
Simplified initialization. */

#include <stdlib.h>

int main( ) {
    char jan[ ] = "January"; /* Compiler computes necessary */
    char feb[ ] = "February"; /* ... size from initializers */
    char mar[ ] = "March";
    char apr[ ] = "April";
    char may[ ] = "May";
    char jun[ ] = "June";
    char jul[ ] = "July";
    char aug[ ] = "August";
    char sep[ ] = "September";
    char oct[ ] = "October";
    char nov[ ] = "November";
    char dec[ ] = "December";

    printf("The months of the year: \n");
    printf("%s \n", jan);
    printf("%s \n", feb);
    printf("%s \n", mar);
    printf("%s \n", apr);
    printf("%s \n", may);
    printf("%s \n", jun);
    printf("%s \n", jul);
    printf("%s \n", aug);
    printf("%s \n", sep);
    printf("%s \n", oct);
    printf("%s \n", nov);
    printf("%s \n", dec);

    return 0;
}

/* OUTPUT: MultiArray2.c */
The months of the year:
January
February
March
April
May
June
July
August
September
October
November
December
*/
Ex:

/*     FILE: MultiArray3.c     */

/* Program with multiple strings containing the
names of the months.

Since a multi-dimensional array is an "array
of arrays", we can use here for our sets of
sets of characters. */

#include <stdlib.h>
#include <string.h>

int main( )
{
    char months[12][10];
    int i;

    strcpy(months[0], "January");
    strcpy(months[1], "February");
    strcpy(months[2], "March");
    strcpy(months[3], "April");
    strcpy(months[4], "May");
    strcpy(months[5], "June");
    strcpy(months[6], "July");
    strcpy(months[7], "August");
    strcpy(months[8], "September");
    strcpy(months[9], "October");
    strcpy(months[10], "November");
    strcpy(months[11], "December");

    printf("The months of the year: \n");

    for (i=0; i<12; i++)
        printf("%s \n", months[i]);

    return 0;
}

/*     OUTPUT: MultiArray3.c     */

The months of the year:
January
February
March
April
May
June
July
August
September
October
November
December
Ex:

/* FILE: MultiArray4.c */

/* Program with multiple strings containing the names of the months. Since a multi-dimensional array is an "array of arrays", we can use here for our sets of sets of characters. */

#include <stdlib.h>

int main( )
{
    char months[12][10] = {"January", /* Using an initialization list. */ "February",
                           "March",
                           "April",
                           "May",
                           "June",
                           "July",
                           "August",
                           "September",
                           "October",
                           "November",
                           "December"};

    int i;
    printf("The months of the year: \n");

    for (i=0; i<12; i++)
        printf("%s \n", months[i]);

    return 0;
}

/* OUTPUT: MultiArray4.c
The months of the year:
January
February
March
April
May
June
July
August
September
October
November
December
*/
Ex:

/*     FILE: MultiArray5.c     */

/* Program with multiple strings containing the
 names of the months. 

 Since month lengths vary, each array of chars can
 be of a different length. However, each array of
 characters can be tracked by the address of the first
 character in the array. An array of pointers can be
 used to track each of those addresses. */

#include <stdlib.h>

int main( )
{
    char* months[12] = {"January", /* Using an initialization list. */
                       "February", /* Compiler is doing a lot here. */
                       "March",   /* Each "string" is a char address */
                       "April",
                       "May",
                       "June",
                       "July",
                       "August",
                       "September",
                       "October",
                       "November",
                       "December"};

    int i;

    printf("The months of the year: \n");

    for (i=0; i<12; i++)
        printf("%s \n", months[i]);

    return 0;
}

/*    OUTPUT: MultiArray5.c

    The months of the year:
    January
    February
    March
    April
    May
    June
    July
    August
    September
    October
    November
    December

*/
Ex:

```c
/*     FILE: direction.c     */
/* Program with multiple strings containing the 
   names of four directions in Karel's world.   */
#include <stdlib.h>

int main( )
{
    char* direction[5] = {"Invalid", /* Direction zero isn't valid. */
                          "North", /* Using an initilaiztion list. */
                          "South",
                          "East",
                          "West"};

    int i;

    printf("The four directions are: \n");

    for (i=1; i<5; i++)
        printf("%d == %s \n", i, direction[i]);

    return 0;
}

/*    OUTPUT: direction.c

The four directions are:
1 == North
2 == South
3 == East
4 == West

*/
```

Ex:

```c
/*     FILE: direction2.c     */

/* Program with multiple strings containing the
names of four directions in Karel's world.
Will process a path file and display Karel's
directions as Karel travels through a world. */

#include <stdlib.h>

int main( )
{
    char* direction[5] = {"Invalid", /* Direction zero isn't valid. */
        "North",    /* Using an initialzation list. */
        "South",
        "East",
        "West"};

    int street, avenue, dirActivity, beepers; /* Current input line. */
    int oldStreet, oldAvenue, oldDirActivity, oldBeepers; /* Previous line/corner. */
    int dir, oldDir;

    /* Read initial line */
    scanf("%d %d %d %d", &street, &avenue, &dirActivity, &beepers);
    printf("Initially: 
");
    printf("\t St: %d Ave: %d Dir: %s Beepers: %d\n",
        street, avenue, direction[dirActivity], beepers);

    /* Shift input to compare to new input. */
    oldStreet = street;
    oldAvenue = avenue;
    oldDirActivity = dirActivity;
    oldDir = dirActivity;
    oldBeepers = beepers;

    /* Process remaining lines in the file. */
    while(scanf("%d %d %d %d", &street, &avenue, &dirActivity, &beepers) == 4){
        if(street < oldStreet) /* Determine direction of travel. */
            dir = 2;
        else if(street > oldStreet)
            dir = 1;
        else if(avenue < oldAvenue)
            dir = 4;
        else if(avenue > oldAvenue)
            dir = 3;
        else
            dir = oldDir;
        printf("\t St: %d Ave: %d Dir: %s\n",
            street, avenue, direction[dir]);

        /* Shift input to compare to next input. */
        oldStreet = street;
        oldAvenue = avenue;
        oldDirActivity = dirActivity;
        oldDir = dir;
        oldBeepers = beepers;
    }

    /* Print final position and status. */
    printf("Finally: \n");
    printf("\t St: %d Ave: %d Dir: %s Status: %d\n",
        oldStreet, oldAvenue, direction[oldDirActivity], oldBeepers);
    return 0;
}
```

cont…
/* OUTPUT: direction2.c

Initially:
St: 3 Ave: 4 Dir: West Beepers: 0
St: 3 Ave: 4 Dir: West
St: 5 Ave: 4 Dir: North
St: 5 Ave: 2 Dir: West
St: 3 Ave: 2 Dir: South
St: 3 Ave: 3 Dir: East
St: 3 Ave: 3 Dir: East
St: 3 Ave: 2 Dir: West
St: 5 Ave: 2 Dir: North
St: 5 Ave: 4 Dir: East
St: 3 Ave: 4 Dir: South

Finally:
St: 3 Ave: 4 Dir: West Status: 0

COMMAND LINE: direction2 < sample4.pth

*/
Ex:

```c
/*     FILE: direction3.c     */
/* Program with multiple strings containing the
   names of four directions in Karel's world.
   Will process a path file and display Karel's
directions as Karel travels through  a world.
   Isolates final line from other input lines. */
#include <stdlib.h>

int main() {
  char* direction[5] = {"Invalid", /* Direction zero isn't valid. */
                        "North",    /* Using an initialization list. */
                        "South",
                        "East",
                        "West");

  int street, avenue, dirActivity, beepers;             /* Current input line. */
  int oldStreet, oldAvenue, oldDirActivity, oldBeepers; /* Previous line/corner. */
  int nextStreet, nextAvenue, nextDirActivity, nextBeepers; /* Read ahead */
  int dir, oldDir;

  /* Read initial line */
  scanf("%d %d %d %d", &street, &avenue, &dirActivity, &beepers);
  printf("Initially: 
        St: %d Ave: %d Dir: %s Beepers: %d
", street, avenue, direction[dirActivity], beepers);

  /* Shift input to compare to new input. */
  oldStreet = street;
  oldAvenue = avenue;
  oldDirActivity = dirActivity;
  oldDir = dirActivity;
  oldBeepers = beepers;

  /* Scan ahead to determine when last line has been read. */
  scanf("%d %d %d %d", &street, &avenue, &dirActivity, &beepers);

  /* While not on last line of file. */
  while(scanf("%d %d %d %d", &nextStreet, &nextAvenue, &nextDirActivity, &nextBeepers) ==

        4){
    if(street < oldStreet) /* Determine direction of travel. */
      dir = 2;
    else if(street > oldStreet)
      dir = 1;
    else if(avenue < oldAvenue)
      dir = 4;
    else if(avenue > oldAvenue)
      dir = 3;
    else
      dir = oldDir;
    printf("\t St: %d Ave: %d Dir: %s\n", street, avenue, direction[dir]);
    if(dirActivity != 0)
      printf("\t beeper activity\n");
  }
  cont...
```
/* Shift input to compare to next input. */
oldStreet = street;
oldAvenue = avenue;
oldDirActivity = dirActivity;
oldDir = dir;
oldBeepers = beepers;
/* Shift back the read-ahead line. */
street = nextStreet;
avenue = nextAvenue;
dirActivity = nextDirActivity;
beepers = nextBeepers;
}
/* Print final position and status. */
printf("Finally: \
");
printf("\t St: %d Ave: %d Dir: %s Status: %d
",
street, avenue, direction[dirActivity], beepers);

return 0;
}

/* OUTPUT: direction3.c

Initially:
    St: 3 Ave: 4 Dir: West Beepers: 0
    St: 3 Ave: 4 Dir: West
    St: 5 Ave: 4 Dir: North
    St: 5 Ave: 2 Dir: West
    St: 3 Ave: 2 Dir: South
    St: 3 Ave: 3 Dir: East
    beeper activity
    St: 3 Ave: 3 Dir: East
    St: 3 Ave: 2 Dir: West
    St: 5 Ave: 2 Dir: North
    St: 5 Ave: 4 Dir: East

Finally:
    St: 3 Ave: 4 Dir: West Status: 0

COMMAND LINE: direction3 < sample4.pth

Initially:
    St: 3 Ave: 7 Dir: West Beepers: 0
    St: 3 Ave: 5 Dir: West
    St: 3 Ave: 3 Dir: West

Finally:
    St: 3 Ave: 1 Dir: West Status: 0

COMMAND LINE: direction3 < motion6rotate.pth

*/
COMMAND-LINE ARGUMENTS

- Information can be passed to a C program from the operating system’s command line.
- The command-line arguments are packaged up into an array of strings, and the count of the number of strings and the array of strings are passed to `main()`. 
- The first line of the definition of `main()` will now look like:
  
  ```
  int main(int argc, char *argv[]) 
  ```

- `argc` is the argument count, `argv` is the array strings.
- Each command-line argument can now be accessed within the program.
Ex:

```c
/*     FILE: cmdLine1.c     */
/* Program echoes all the command line arguments given. */
#include <stdio.h>
int main(int argc, char *argv[]) {
    int i;
    for(i=0; i<argc; i++)
        printf("argv[%d] = %s\n", i, argv[i]);
    return 0;
}

/* OUTPUT: cmdLine1.c */
argv[0] = cmdLine1
argv[1] = one
argv[2] = two
argv[3] = three
    COMMAND LINE: cmdLine1 one two three
argv[0] = cmdLine1
argv[1] = one
argv[2] = 1
argv[3] = two
argv[4] = 2
argv[5] = three
argv[6] = 3
    COMMAND LINE: cmdLine1 one 1 two 2 three 3
argv[0] = cmdLine1
argv[1] = Jim Polzin
    COMMAND LINE: cmdLine1 "Jim Polzin" "Teaches C, sometimes."
*/
OPENGL/GLUT

- OpenGL is an open source Graphics Library. There are versions available for most major operating system. There’s versions for the WinTel platform.
- Using the Graphics Library Utility Toolkit (GLUT) you can write C/C++ code with very little effort that produces windowing and graphics capability.
- See the Appendix for this document for additional notes and resources regarding OpenGL and GLUT.

Ex:

```c
/*     FILE: hatch.c     */

/* Program marks the left and bottom boundaries of
 a window with hatch-marks, much like the first
 quadrant in the cartesian coordinate system. */
#include <stdlib.h>
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit(); /* <- Functions written by the programmer to */
void display(); /* ... produce the graphics they desire. */
void reshape(int, int); /* ... myinit() is called once. Display() */
void hatch(); /* ... and reshape() are called as needed. */
/* ... hatch() is called by display. */

/* Function to initialize OpenGL parameters and
 prepare for drawing as the programmer sees fit. */
void myinit()
{
  glClearColor(1.,1.,1.,1.); /* white background */
  glColor3f(0.,0.,0.); /* black foreground */
  glShadeModel(GL_FLAT);
  /* set up viewing scrWidth x scrHeight window with origin lower left */
  glViewport(0,0,scrWidth,scrHeight);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.,(GLdouble)scrWidth,0.,(GLdouble)scrHeight);
  glMatrixMode(GL_MODELVIEW);
}

/* Function registered with OpenGL for producing the graphics */
void display()
{
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_LINES);
  hatch();
  glEnd();
  glFlush();
}

cont...
```
/* Hatch the boundaries
... Called by display( ) to allow display( ) to be
... more succinct. */
void hatch( )
{
    int i;
    for(i=5; i < scrHeight; i+=5){
        glVertex2i(0,i);
        glVertex2i(4,i);
        if(i%25 == 0){
            glVertex2i(0,i);
            glVertex2i(10,i);
        }
    }
    for(i=5; i < scrWidth; i+=5){
        glVertex2i(i,0);
        glVertex2i(i,4);
        if(i%25 == 0){
            glVertex2i(i,0);
            glVertex2i(i,10);
        }
    }
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    /* set up viewing scrWidth x scrHeight window with origin lower left */
    glViewport( 0, 0, scrWidth, scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(0.,(GLdouble)scrWidth,0.,(GLdouble)scrHeight);
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);                          /* These 1st four function   */
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);  /*  ... calls are OpenGL/GLUT */
    glutInitWindowSize(scrWidth,scrHeight);        /*  ... preparatory calls     */
    glutCreateWindow("Hatch");
    glutDisplayFunc(display);    /* <- Registers the programmers drawing code  */
    /* ... so that the graphics can be displayed */
    /* ... in the window. */
    myinit( );             /* This is a local function to establish the */
    /* ... current state desired by the programmer. */
    glutReshapeFunc(reshape);     /* <- Registers the programmers drawing code  */
    /* ... with OpenGL/GLUT so that if window is */
    /* ... reshaped, the graphics can be redrawn. */
    glutMainLoop( );       /* Starts the event-loop for the graphics environment. */
    return 0;
}

cont...
Ex:

```c
/* FILE: hatch2.c */

#include <stdlib.h>
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit( ); /* <- Functions written by the programmer to */
void display( ); /* ... produce the graphics they desire. */
void reshape(int, int); /* ... myinit() is called once. Display() */
void hatch( ); /* ... and reshape() are called as needed. */
/* ... hatch() is called by display. */

/* Function to initialize OpenGL parameters and */
/* prepare for drawing as the programmer sees fit. */
void myinit( )
{
    glClearColor(1.,1.,1.,1.); /* white background */
    glColor3f(0.,0.,0.); /* black foreground */
    glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin centered */
    /* in the window. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(-(GLdouble)scrWidth/2,(GLdouble)scrWidth/2,
                -(GLdouble)scrHeight/2,(GLdouble)scrHeight/2);
    glMatrixMode(GL_MODELVIEW);
}

/* Function registered with OpenGL for producing the graphics */
void display( )
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    hatch( );
    glEnd( );
    glFlush( );
}

/* Hatch the boundaries */
/* Called by display() to allow display() to be */
/* ... more succinct. */
void hatch( )
{
    int i;
    int halfHeight = scrHeight/2;
    int halfWidth = scrWidth/2;
    for(i=5; i < halfHeight; i+=5){
        glVertex2i(-2,i);
        glVertex2i(2,i);
        glVertex2i(-2,-i);
        glVertex2i(2,-i);
        if(i%25 == 0){
            glVertex2i(-5,i);
            glVertex2i(5,i);
            glVertex2i(-5,-i);
            glVertex2i(5,-i);
        }
    }
    cont...
```
for(i=5; i < halfWidth; i+=5){
    glVertex2i(i,-2);
    glVertex2i(i,2);
    glVertex2i(-i,-2);
    glVertex2i(-i,2);
    if(i%25 == 0){
        glVertex2i(i,-5);
        glVertex2i(i,5);
        glVertex2i(-i,-5);
        glVertex2i(-i,5);
    }
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    /* set up viewing scrWidth x scrHeight window with origin centered
     * in the window. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(-(GLdouble)scrWidth/2,(GLdouble)scrWidth/2,-
               (GLdouble)scrHeight/2,(GLdouble)scrHeight/2);
    glMatrixMode(GL_MODELVIEW);
    display( );
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);                          /* These 1st four function   */
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);  /*  ... calls are OpenGL/GLUT */
    glutInitWindowSize(scrWidth,scrHeight);        /*  ... preparatory calls     */
    glutCreateWindow("Hatch 2");
    glutDisplayFunc(display);    /* <- Registers the programmers drawing code  */
    myinit( );             /* This is a local function to establish the    */
                           /* ... current state desired by the programmer. */
    glutReshapeFunc(reshape);     /* <- Registers the programmers drawing code  */
                           /* ... with OpenGL/GLUT so that if window is * /
                           /* ... reshaped, the graphics can be redrawn. */
    glutMainLoop( );       /* Starts the event-loop for the graphics environment. */
    return 0;
}

cont…
Ex:

/* FILE: Grid.c */
/* Program that draws a "Karel-like" grid */

#include <stdlib.h>
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit(); /* <- Functions written by the programmer to */
void display(); /* ... produce the graphics they desire. */
void reshape(int, int); /* ... myinit() is called once. Display() */
/* ... and reshape() are called as needed. */

void border(); /* ... border() and drawGrid() are called by */
void drawGrid(); /* ... display. */

/* Function to initialize OpenGL parameters and */
/* prepare for drawing as the programmer sees fit. */
void myinit()
{
    glClearColor(1.,1.,1.,1.); /* white background */
    glColor3f(0.,0.,0.); /* black foreground */
    glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin shifted */
    /* up and to the right 25 pixels. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-25.,(GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
}

/* Function registered with OpenGL for producing the graphics */
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(3.0); /* Width line-width for boundary */
    glBegin(GL_LINES);
    border();
    glEnd();

    glLineWidth(1.0); /* Small line-width for St. & Ave. */
    glBegin(GL_LINES);
    drawGrid();
    glEnd();
    glFlush();
}

cont…
/* Draw the world boundaries */
/* ... Called by display() to allow display() to be 
   ... more succinct. */
void border()
{
    int i;
    int halfHeight = scrHeight/2;
    int halfWidth = scrWidth/2;

    glVertex2i(0,0);
    glVertex2i(0,halfHeight);
    glVertex2i(0,0);
    glVertex2i(halfWidth,0);
}

/* Draw the streets and avenues */
/* ... Called by display() to allow display() to be 
   ... more succinct. */
void drawGrid()
{
    int i;
    int halfHeight = scrHeight/2;
    int halfWidth = scrWidth/2;

    for(i=25; i < halfHeight; i+=25){
        glVertex2i(0,i);
        glVertex2i(halfWidth,i);
    }
    for(i=25; i < halfWidth; i+=25){
        glVertex2i(i,0);
        glVertex2i(i,halfHeight);
    }
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0,0,scnscrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-25.,(GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
    display();
}

cont...
int main(int argc, char** argv)
{
    glutInit(&argc,argv); /* These 1st four function */
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); /* ... calls are OpenGL/GLUT */
    glutInitWindowSize(scrWidth,scrHeight); /* ... preparatory calls */
    glutCreateWindow("Grid");

    glutDisplayFunc(display); /* <- Registers the programmers drawing code */
    /* ... so that the graphics can be displayed */
    /* ... in the window. */

    myinit(); /* This is a local function to establish the */
    /* ... current state desired by the programmer. */

    glutReshapeFunc(reshape); /* <- Registers the programmers drawing code */
    /* ... with OpenGL/GLUT so that if window is */
    /* ... reshaped, the graphics can be redrawn. */

    glutMainLoop( ); /* Starts the event-loop for the graphics environment. */

    return 0;
}
Ex:

/*     FILE: Grid2.c     */

/* Program that draws a "Karel-like" grid
With labels. */

#include <stdlib.h>
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit( ); /* <- Functions written by the programmer to */
void display( ); /* ... produce the graphics they desire. */
void reshape(int, int); /* ... myinit( ) is called once. Display( ) */
/* ... and reshape( ) are called as needed. */

void border( ); /* ... border( ) and drawGrid() are called by */
void drawGrid( ); /* ... display. */

/* Writes each character in a string */
void printString(void *font,char *str)
{
    int i, len;

    len=strlen(str);

    for(i=0;i<len;i++)
        glutBitmapCharacter(font,*str++);
}

/* Function to initialize OpenGL parameters and
prepare for drawing as the programmer sees fit. */
void myinit( )
{
    glClearColor(1.,1.,1.,1.); /* white background */
    glColor3f(0.,0.,0.); /* black foreground */
    glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin shifted
    up and to the right 25 pixels. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(-25.,(GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
}

cont…
/* Function registered with OpenGL for producing the graphics */
void display()
{
    char buf[128];
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(3.0); /* Width line-width for boundary */
    glBegin(GL_LINES);
    border();
    glEnd();
    glLineWidth(1.0); /* Small line-width for St. & Ave. */
    glBegin(GL_LINES);
    drawGrid();
    glEnd();
    /* Label the Graphic in the window */
    strcpy(buf,"Karel's World");
    glRasterPos2i(2,scrHeight-50);
    printString(GLUT_BITMAP_HELVETICA_12,buf);
    /* Label the Avenues below the grid */
    strcpy(buf,"Avenues");
    glRasterPos2i(scrWidth/4,-25);
    printString(GLUT_BITMAP_HELVETICA_12,buf);
    /* Label the Streets along the side */
    strcpy(buf,"Streets");
    glRasterPos2i(2,scrHeight/2+25);
    printString(GLUT_BITMAP_HELVETICA_12,buf);
    glFlush();
}
/* Draw the world boundaries */
/* ... Called by display() to allow display() to be ...
   more succinct. */
void border() {
    int i;
    int halfHeight = scrHeight/2;
    int halfWidth = scrWidth/2;
    glVertex2i(0,0);
    glVertex2i(0,halfHeight);
    glVertex2i(0,0);
    glVertex2i(halfWidth,0);
    glVertex2i(halfWidth,0);
    glVertex2i(0,0);
}
cont...
/* Draw the streets and avenues */
/* ... Called by display( ) to allow display( ) to be
... more succinct. */
void drawGrid( )
{
    int i;
    int halfHeight = scrHeight/2;
    int halfWidth = scrWidth/2;

    for(i=25; i < halfHeight; i+=25){
        glVertex2i(0,i);
        glVertex2i(halfWidth,i);
    }
    for(i=25; i < halfWidth; i+=25){
        glVertex2i(i,0);
        glVertex2i(i,halfHeight);
    }
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(-25.,(GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
    display( );
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);                          /* These 1st four function   */
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);  /*  ... calls are OpenGL/GLUT */
    glutInitWindowSize(scrWidth,scrHeight);        /*  ... preparatory calls     */
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display);    /* <- Registers the programmers drawing code  */
    myinit();             /* This is a local function to establish the    */
    glutReshapeFunc(reshape);     /* <- Registers the programmers drawing code  */
    glutMainLoop( );       /* Starts the event-loop for the graphics environment. */
    return 0;
}

cont...
Karel's World

Streets

Avenues
Ex:

```c
/*     FILE: Grid3.c     */
/* Program that draws a "Karel-like" grid                      */
    With labels.                                           */

#include <stdlib.h>
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit( ); /* <- Functions written by the programmer to */
void display( ); /* ... produce the graphics they desire.     */
void reshape(int, int); /* ... myinit( ) is called once. Display( ) */
    /* ... and reshape( ) are called as needed. */
void border( ); /* ... border( ) and drawGrid( ) are called by */
void drawGrid( ); /* ... display.                             */

/* Writes each character in a string */
void printString(void *font,char *str)
{
    int i, len;
    len=strlen(str);
    for(i=0;i<len;i++)
        glutBitmapCharacter(font,*str++);
}
/* Function to initialize OpenGL parameters and */
/* prepare for drawing as the programmer sees fit. */
void myinit( )
{
    glClearColor(1.,1.,1.,1.); /* white background */
    glColor3f(0.,0.,0.); /* black foreground */
    glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin shifted */
    /* up and to the right 25 pixels. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(-25.,(GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
}

cont…
```
/ Function registered with OpenGL for producing the graphics */
void display( )
{
  char buf[128];
  glClear(GL_COLOR_BUFFER_BIT);
  glLineWidth(3.0);   /* Width line-width for boundary */
  glBegin(GL_LINES);
  border( );
  glEnd( );
  glLineWidth(1.0);   /* Small line-width for St. & Ave. */
  glBegin(GL_LINES);
  drawGrid( );
  glEnd( );

  /* Label the Graphic in the window */
  strcpy(buf,"Karel's World");
  glRasterPos2i(2,scrHeight-40);
  printString(GLUT_BITMAP_HELVETICA_12,buf);
  /* Label the Avenues below the grid */
  strcpy(buf,"Avenues");
  glRasterPos2i(scrWidth/4,-25);
  printString(GLUT_BITMAP_HELVETICA_12,buf);
  /* Label the Streets along the side */
  strcpy(buf,"Streets");
  glRasterPos2i(2,scrHeight/2+25);
  printString(GLUT_BITMAP_HELVETICA_12,buf);
  glFlush( );
}

/* Draw the world boundaries */
/* ... Called by display( ) to allow display( ) to be ... more succinct. */
void border( )
{
  int i;
  glVertex2i(0,0);
  glVertex2i(0,scrHeight - 50);
  glVertex2i(0,0);
  glVertex2i(scrWidth - 50,0);
  glVertex2i(0,0);
  glVertex2i(scrWidth - 50,0);
}

/* Draw the streets and avenues */
/* ... Called by display( ) to allow display( ) to be ... more succinct. */
void drawGrid( )
{
  int i;
  for(i=25; i < scrHeight - 50; i+=25)
  {
    glVertex2i(0,i);
    glVertex2i(scrWidth - 50,i);
  }
  for(i=25; i < scrWidth - 50; i+=25)
  {
    glVertex2i(i,0);
    glVertex2i(i,scrHeight - 50);
  }
}

cont...
/ Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0, 0, scrWidth, scrHeight);
    glMatrixMode(GL_PROJECTION);
   glLoadIdentity( );
    gluOrtho2D(-25., (GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
    display( );
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv); /* These 1st four function calls are OpenGL/GLUT */
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); /* ... calls are OpenGL/GLUT */
    glutInitWindowSize(scrWidth, scrHeight); /* ... preparatory calls */
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display); /* <- Registers the programmers drawing code */
    /* ... so that the graphics can be displayed */
    /* ... in the window. */
    myinit(); /* This is a local function to establish the */
    /* ... current state desired by the programmer. */
    glutReshapeFunc(reshape); /* <- Registers the programmers drawing code */
    /* ... with OpenGL/GLUT so that if window is */
    /* ... reshaped, the graphics can be redrawn. */
    glutMainLoop(); /* Starts the event-loop for the graphics environment. */
    return 0;
}

cont...
Ex:

/* FILE: Grid4.c */

/* Program that draws a "Karel-like" grid */

With streets and avenues labeled and numbered. */

#include <stdlib.h>
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit( );        /* <- Functions written by the programmer to */
void display( );       /* ... produce the graphics they desire. */
void reshape(int, int); /* ... myinit( ) is called once. Display( ) */
/* ... and reshape( ) are called as needed. */

void border( );        /* ... border( ) and drawGrid( ) are called by */
void drawGrid( );      /* ... display. */

/* Labels each of the streets and avenues */
void numberStreetsAvenues(void *font, int pts, int st, int ave, int shift)
{
    int i, len;
    char buf[128];
    
    for(i=0; i<ave; i++){
        sprintf(buf, "%d", i+1);  
        glRasterPos2i((i+1)*shift,-(pts+3));
        for(len=0; len<strlen(buf); len++)
            glutBitmapCharacter(font,buf[len]);
    }
    
    for(i=0; i<st; i++){
        sprintf(buf, "%d", i+1);  
        glRasterPos2i(-(pts+3),(i+1)*shift);
        for(len=0; len<strlen(buf); len++)
            glutBitmapCharacter(font,buf[len]);
    }

    /* Writes a string vertically from (x,y) down */
    void verticalPrintString(void *font, char *str, int pts, int x, int y)
    {
        int i, len;
        len=strlen(str);
        
        for(i=0; i<len; i++){
            glRasterPos2i(x, y-i*(pts+3));
            glutBitmapCharacter(font,*str++);
        }
    }

    /* Writes each character in a string */
    void printString(void *font,char *str)
    {
        int i, len;
        len=strlen(str);
        
        for(i=0; i<len; i++)
            glutBitmapCharacter(font,*str++);
    }

    cont...
/* Function to initialize OpenGL parameters and
prepare for drawing. */
void myinit()
{
    glClearColor(1.,1.,1.,1.); /* white background */
    glColor3f(0.,0.,0.);     /* black foreground */
    glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin shifted
    up and to the right 25 pixels. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-25.,(GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
}

/* Function registered with OpenGL for producing display */
void display()
{
    char buf[128];

    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(3.0);   /* Width line-width for boundary */
    glBegin(GL_LINES);
        border();
    glEnd();
    glLineWidth(1.0);   /* Small line-width for St. & Ave. */
    glBegin(GL_LINES);
        drawGrid();
    glEnd();

    strcpy(buf,"Karel's World");
    glRasterPos2i(2,scrHeight-40);
    printString(GLUT_BITMAP_HELVETICA_12,buf);

    strcpy(buf,"Avenues");
    glRasterPos2i(scrWidth/4,-25);
    printString(GLUT_BITMAP_HELVETICA_12,buf);

    strcpy(buf,"Streets");
    verticalPrintString(GLUT_BITMAP_HELVETICA_12,buf,12,-25,scrHeight/2);

    numberStreetsAvenues(GLUT_BITMAP_HELVETICA_12,12,
                             (scrHeight-50)/25, (scrWidth-50)/25, 25);
    glFlush();
}

cont...
/* Draw the world boundaries */
/* ... Called by display() to allow display() to be 
... more succinct. */
void border()
{
    int i;
    glVertex2i(0,0);
    glVertex2i(0,scrHeight - 50);
    glVertex2i(0,0);
    glVertex2i(scrWidth - 50,0);
}

/* Draw the streets and avenues */
/* ... Called by display() to allow display() to be 
... more succinct. */
void drawGrid()
{
    int i;
    for(i=25; i < scrHeight - 50; i+=25)
    {
        glVertex2i(0,i);
        glVertex2i(scrWidth - 50,i);
    }
    for(i=25; i < scrWidth - 50; i+=25)
    {
        glVertex2i(i,0);
        glVertex2i(i,scrHeight - 50);
    }
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-25.,(GLdouble)scrWidth-25.,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
    display();
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);                          /* These 1st four function   */
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);  /*  ... calls are OpenGL/GLUT */
    glutInitWindowSize(scrWidth,scrHeight);        /*  ... preparatory calls     */
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display);    /* <- Registers the programmers drawing code  */
    /* ... so that the graphics can be displayed  */
    /* ... in the window. */
    myinit();             /* This is a local function to establish the    */
    /* ... current state desired by the programmer. */
    glutReshapeFunc(reshape);     /* <- Registers the programmers drawing code  */
    /* ... with OpenGL/GLUT so that if window is  */
    /* ... reshaped, the graphics can be redrawn. */
    glutMainLoop( );       /* Starts the event-loop for the graphics environment. */
    return 0;
}

cont...
Ex:

/*     FILE: Grid5.c     */
/* Program that draws a "Karel-like" grid 
          With streets and avenues labeled and numbered. 
          And has a global constant for the distance 
          between streets and avenues. */ 
#include <stdlib.h> 
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
const int dist = 50; /* Distance between streets ... and avenues. */

void myinit( ); /* <-> - Functions written by the programmer to */
void display( ); /* ... produce the graphics they desire. */
void reshape(int, int); /* ... myinit() is called once. Display() */
/* ... and reshape() are called as needed. */

void border( ); /* ... border() and drawGrid() are called by */
void drawGrid( ); /* ... display. */

/* Writes each character in a string */
void printString(void *font,char *str)
{
    int i, len;

    len=strlen(str);

    for(i=0;i<len;i++)
        glutBitmapCharacter(font,str[i]);
}

/* Function to initialize OpenGL parameters and 
   prepare for drawing. */
void myinit( )
{
    glClearColor(1.,1.,1.,1.); /* white background */
    glColor3f(0.,0.,0.); /* black foreground */
    glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin shifted 
    up and to the right 25 pixels. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D((GLdouble)-dist,(GLdouble)scrWidth-dist,(GLdouble)-dist,(GLdouble)scrHeight-
    dist);
    glMatrixMode(GL_MODELVIEW);
}

cont...
/* Function registered with OpenGL for producing the graphics */
void display( )
{
    char buf[128];
    glClear(GL_COLOR_BUFFER_BIT);

    glLineWidth(3.0);  /* Width line-width for boundary */
    glBegin(GL_LINES);
        border( );
    glEnd( );

    glLineWidth(1.0);  /* Small line-width for St. & Ave. */
    glBegin(GL_LINES);
        drawGrid( );
    glEnd( );

    /* Label the Graphic in the window */
    strcpy(buf,"Karel's World");
    glRasterPos2i(2,scrHeight-50);
    printString(GLUT_BITMAP_HELVETICA_12,buf);
    /* Label the Avenues below the grid */
    strcpy(buf,"Avenues");
    glRasterPos2i(scrWidth/4,-dist);
    printString(GLUT_BITMAP_HELVETICA_12,buf);
    /* Label the Streets along the side */
    strcpy(buf,"Streets");
    glRasterPos2i(2,scrHeight/2+dist);
    printString(GLUT_BITMAP_HELVETICA_12,buf);

    glFlush( );
}

/* Draw the world boundaries */
/* ... Called by display( ) to allow display( ) to be ... more succinct. */
void border( )
{
    int i;
    int halfHeight = scrHeight/2;
    int halfWidth = scrWidth/2;

    glVertex2i(0,0);
    glVertex2i(0,halfHeight);

    glVertex2i(0,0);
    glVertex2i(halfWidth,0);
}

cont...
/* Draw the streets and avenues */
/* ... Called by display() to allow display() to be
... more succinct. */
void drawGrid( )
{
    int i;
    int halfHeight = scrHeight/2;
    int halfWidth = scrWidth/2;

    for(i=dist; i < halfHeight; i+=dist){
        glVertex2i(0,i);
        glVertex2i(halfWidth,i);
    }
    for(i=dist; i < halfWidth; i+=dist){
        glVertex2i(i,0);
        glVertex2i(i,halfHeight);
    }
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D((GLdouble)-dist,(GLdouble)scrWidth-dist,(GLdouble)-dist,(GLdouble)scrHeight-
    dist);
    glMatrixMode(GL_MODELVIEW);
    display( );
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);                          /* These 1st four function   */
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);  /* ... calls are OpenGL/GLUT */
    glutInitWindowSize(scrWidth,scrHeight);        /* ... preparatory calls     */
    glutCreateWindow(argv[0]);

    glutDisplayFunc(display);    /* <- Registers the programmers drawing code  */
    /* ... so that the graphics can be displayed */
    /* ... in the window. */
    myinit( );             /* This is a local function to establish the    */
    /* ... current state desired by the programmer. */
    glutReshapeFunc(reshape);     /* <- Registers the programmers drawing code */
    /* ... with OpenGL/GLUT so that if window is */
    /* ... reshaped, the graphics can be redrawn. */
    glutMainLoop( );       /* Starts the event-loop for the graphics environment. */
    return 0;
}

cont...
Ex:

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "glut.h"
#define SIZE 100

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit( );
void display( );
void reshape(int, int);
void drawLine(int x1, int y1, int x2, int y2);
void drawArrow(int x1, int y1, int x2, int y2);
int readPairs(void);

int count; /* Number of points */
int pairs[SIZE][2];

/* Function to initialize OpenGL parameters and prepare for drawing. */
void myinit( )
{
    glClearColor(1.,1.,1.,1.); /* white background */
    glColor3f(0.,0.,0.); /* black foreground */
    glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin at lower-left. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(0.0,(GLdouble)scrWidth,0.0,(GLdouble)scrHeight);
    glMatrixMode(GL_MODELVIEW);
}

/* Function registered with OpenGL for producing display */
void display( )
{
    int i;
    char buf[128];
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    for(i=0; i<count-1; i++) /* draws between all the pairs */
        drawArrow(pairs[i][0],pairs[i][1],pairs[i+1][0],pairs[i+1][1]);
    glEnd( );
    glFlush( );
}
```

cont...
/* Draw a line between two (x,y) pairs */
void drawLine(int x1, int y1, int x2, int y2)
{
    glVertex2i(x1,y1);
    glVertex2i(x2,y2);
}

/* Draw a vertical or horizontal arrow between two (x,y) pairs */
void drawArrow(int x1, int y1, int x2, int y2)
{
    int dx1,dy1,dx2,dy2;
    drawLine(x1,y1,x2,y2); /* Arrow shaft */
    if(x2<x1){            /* Arrow tip shift */
        dx2 = dx1 = +8;
        dy1 = -6;
        dy2 = +6;
    } else if(x1<x2){
        dx2 = dx1 = -8;
        dy1 = -6;
        dy2 = +6;
    } else if(y1<y2){
        dx2 = +6;
        dx1 = -6;
        dy1 = -8;
        dy2 = -8;
    } else if(y2<y1){
        dx2 = +6;
        dx1 = -6;
        dy1 = +8;
        dy2 = +8;
    }
    drawLine(x2,y2,x2+dx1,y2+dy1); /* Arrow tip 1/2 */
    drawLine(x2,y2,x2+dx2,y2+dy2); /* Arrow tip 1/2 */
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(0.0,(GLdouble)scrWidth,0.0,(GLdouble)scrHeight);
    glMatrixMode(GL_MODELVIEW);
    display( );
}

/* Reads pairs of ints from stdin and counts them */
int readPairs(void)
{
    int i = 0;
    while(i < SIZE && scanf("%d%d", &pairs[i][0], &pairs[i][1]) == 2){
        i++;
    }
    return i;
}

cont...
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(scrWidth, scrHeight);
  glutCreateWindow(argv[0]);
  glutDisplayFunc(display);
  myinit();
  glutReshapeFunc(reshape);

  count = readPairs(); /* read points to draw */

  glutMainLoop();
  return 0;
}

/* OUTPUT: drawPath.c

INPUT:
    10 10
    100 10
    100 100
    50 100
    50 300
*/
MAKE UTILITY

• The *make* utility allows easy construction/compilation of applications involving many source files.

• *make* reads a set of rules from a "make" file that describes what pieces need to be combined to produce the final product, and any dependencies between those pieces.

• *make* is a very powerful utility and can do much more than will be described here.

• Lines in a basic make file consist of commands that describe how to build the parts that make something, and dependencies that describe which parts to make first.

• *make* by default uses a file named *makefile* in the current working directory to determine how to build something. You can tell *make* to use a file of another name.

Ex:

```
# ************************************************* **********
# *** A make file to build the opengl, GLUT files        ****
# ************************************************* **********
glutsourcex.exe: glutsource.o
    gcc -o glutsource glutsource.o libglut32.a -lglu32 -lopengl32

```  

Result when only the source code exists:

```
H:\c\examples\OpenGL>make -f glutsource.mak
    gcc -c glutsource.c
    gcc -o glutsource glutsource.o libglut32.a -lglu32 -lopengl32
```

Result when previously compiled and source code is unchanged:

```
H:\c\examples\OpenGL>make -f glutsource.mak
    make: `glutsourcex.exe' is up to date.
```
Ex:

```
# ************************************************* **********
# *** A make file to build s set of interdependent files.****
# ************************************************* **********
extern3.exe: extern3.o extern2.o
  gcc -o extern3 extern3.o extern2.o
extern3.o: extern3.c
  gcc -c extern3.c
extern2.o: extern2.c
  gcc -c extern2.c
```

- **Result when only the source code exists:**
  ```
  H:\c\examples>make -f extern3.mak
gcc -c extern3.c
gcc -c extern2.c
gcc -o extern3 extern3.o extern2.o
  ```

- **Result when previously compiled and source code is unchanged:**
  ```
  H:\c\examples>make -f extern3.mak
make: `extern3.exe' is up to date.
  ```

- **Result when extern2.o is not present:**
  ```
  H:\c\examples>del extern2.o

  H:\c\examples>make -f extern3.mak
gcc -c extern2.c
gcc -o extern3 extern3.o extern2.o
  ```
C STORAGE CLASSES

Automatic
- variables defined in a function or block of code are automatic by default
- can be explicitly declared using the \textit{auto} keyword
- known only in the function or block of code they are defined in
- exist only while the function or block is executing
- not initialized by default

External
- variables defined outside any function
- known to all functions defined in the source file after the variable definition
- \textit{extern} keyword declares an external and makes it known to a function or file regardless of where the external variable is actually defined
- exist for the entire duration of the program
- initialized to zero by default

Static automatic
- known only to the function in which they are defined
- \textit{static} keyword defines a static automatic variable
- exist for the entire duration of the program
- initialized once, to zero by default
- retain their value between function calls

Static External
- external variable restricted to the file it is defined in
- \textit{static} keyword declares an external variable to be static external

Dynamic memory
- allocated using \textit{malloc}()
- exists until released by \textit{free}()
- accessed by address

Function scope
External
- function that can be accessed by other files
- functions are external by default

Static
- function accessible only in the defining file
- \textit{static} keyword declares a function to be static
Ex:

```c
/*     FILE: extern.c     */
/* Uses an external set of variables and a static variable. */
#include <stdio.h>
void change_char(void);
char str[81] = "Hello world!";
int position;

int main( )
{
    printf("str = %s\n", str);
    position = 11;
    change_char( );
    printf("str = %s\n", str);
    for(position=10; position>5; position --)
        change_char( );
    printf("str = %s\n", str);
    return 0;
}

void change_char(void)
{
    static int calls;
    if(calls != 0)
        printf("I've been called %d time%c before.\n", calls, calls==1?' ':'s');
    else
        printf("I've never been called before.\n");
    str[position] = 'X';
    calls++;
    return;
}

/*    OUTPUT: extern.c    */
str = Hello world!
I've never been called before.
str = Hello worldX
I've been called 1 time before.
I've been called 2 times before.
I've been called 3 times before.
I've been called 4 times before.
I've been called 5 times before.
str = Hello XXXXX
*/
```
Ex:

```c
/*     FILE: extern2.c     */

/* Functions that use an external set of variables */
extern char str[81];
extern int position;

void change_char(void)
{
    static int calls;
    if(calls != 0)
        printf("I've been called %d time%s before.
", calls, calls==1?' ':'s');
    else
        printf("I've never been called before.
");
    str[position] = 'X';
    calls++;
    return;
}

/*     FILE: extern3.c     */

/* Uses an external set of variables and a static variable. */
/* Calls function found in another file. */
#include <stdio.h>

void change_char(void);
char str[81] = "Hello world!";
int position;

int main( )
{
    printf("str = %s\n", str);
    position = 11;
    change_char( );
    printf("str = %s\n", str);
    for(position=10; position>5; position --)
        change_char( );
    printf("str = %s\n", str);
    return 0;
}

/*     OUTPUT: extern3.c     */

str = Hello world!
I've never been called before.
str = Hello worldX
I've been called 1 time before.
I've been called 2 times before.
I've been called 3 times before.
I've been called 4 times before.
I've been called 5 times before.
str = Hello XXXXXX

*/
Dynamic Memory Allocation

Up until now, the storage requirements of a C program needed to be established at compile time. Variables needed to be defined so that their storage requirements would be known. Arrays needed to be sized so that their storage requirements would be known.

Dynamic memory allocation allows storage requirements to be determined at run-time.

When the requirements are known, a request is made for the required amount of storage and the program can then proceed. This allows the program to tailor its storage use to exactly fit its needs during each run, or at any given point in time during a run.

There is a cost. There is overhead incurred while the request for storage is being met.

The key to dynamic memory allocation is pointers and the appropriate allocation function.

`malloc()` is the basic memory allocation function. `Malloc` is told how many bytes of storage are needed and if the allocation can be satisfied `malloc` returns the address of the storage. If the allocation fails, `NULL` is returned.

Dynamically allocated memory can and should be deallocated using `free()`.
Ex:

```c
/* FILE: dynamic1.c */

/* Dynamic memory allocation using malloc( ).
   Note: #include stdlib.h for memory allocation functions. */

#include <stdio.h>
#include <stdlib.h>

int main( )
{
    int* ptr;
    int size, i;

    printf("Please enter number of integers to be read: ");
    scanf("%d", &size);

    ptr = malloc(size * sizeof(int)); /* allocation is requested in bytes */
    /* Once the storage is allocated, ptr can be treated like an array. */

    for(i=0; i<size; i++)
        printf("Enter integer %d of %d: ", i+1, size);
    scanf("%d", &ptr[i]);

    for(i=0; i<size; i++)
        printf("ptr[%d] = %d\n", i, ptr[i]);
    free(ptr);
    return 0;
}

/* OUTPUT: dynamic1.c
   Please enter number of integers to be read: 5
   Enter integer 1 of 5: 11
   Enter integer 2 of 5: 22
   Enter integer 3 of 5: 33
   Enter integer 4 of 5: 44
   Enter integer 5 of 5: 55
   ptr[0] = 11
   ptr[1] = 22
   ptr[2] = 33
   ptr[3] = 44
   ptr[4] = 55

   */
```
Ex:

```c
#include <stdio.h>
#include <stdlib.h>

int main( )
{
    int* ptr;
    int size, i;

    printf("Please enter number of integers to be read: ");
    scanf("%d", &size);

    ptr = malloc(size * sizeof(int));   /* allocation is requested in bytes */

    if(ptr != NULL){
        /* Once the storage is allocated, ptr can be treated like an array. */
        for(i=0; i<size; i++){
            printf("Enter integer %d of %d: ", i+1, size);
            scanf("%d", &ptr[i]);
        }
        for(i=0; i<size; i++)
            printf("ptr[%d] = %d\n", i, ptr[i]);
        free(ptr);
    }
    else
        printf("FAILURE: Unable to allocate storage.\n");
    return 0;
}
```

/* OUTPUT: dynamic2.c

Please enter number of integers to be read: 5
Enter integer 1 of 5: 511
Enter integer 2 of 5: 522
Enter integer 3 of 5: 533
Enter integer 4 of 5: 544
Enter integer 5 of 5: 555
ptr[0] = 511
ptr[1] = 522
ptr[2] = 533
ptr[3] = 544
ptr[4] = 555
*/
Ex:

```c
/*     FILE: max_cnt.c     */

/*
Loads an array with up to SIZE values.
Finds the max and the count of values
greater than 90.
*/

#include <stdio.h>
#define SIZE 50

int main( )
{
    int scores[SIZE];
    int i, n, max, a_count;

    /* Get number of values to read */
    printf("Please enter number of scores (%d or less): ", SIZE);
    scanf("%d", &n);

    /* Validate number entered by user. */
    if (n<=SIZE && n>0){
        /* Read score values into array */
        for(i=0; i<n; i++)
        {
            printf("Enter value %d of %d: ", i+1, n);
            scanf("%d", &scores[i]);
        }

        /* Find maximum of values read. */
        max = scores[0];
        for(i=1; i<n; i++)
        {
            if (scores[i] > max)
                max = scores[i];
        }

        printf("Max score = %d\n", max);

        /* Count number of A's, scores greater than 90 */
        a_count = 0;
        for(i=0; i<n; i++)
        {
            if (scores[i] > 90)
                a_count++;
        }

        printf("A's = %d\n", a_count);
    }

    return 0;
}

/*    OUTPUT: max_cnt.c

Please enter number of scores (50 or less): 4
Enter value 1 of 4: 75
Enter value 2 of 4: 85
Enter value 3 of 4: 95
Enter value 4 of 4: 92
Max score = 95
A's = 2
*/```
Ex:
/*     FILE: dynamic3.c     */

/* Prompts the user for the number of scores
   and uses malloc( ) to allocate the appropriate
   amount of storage.

   Finds the max and the count of values
   greater than 90. */

#include <stdio.h>
#include <limits.h>
#include <stdlib.h>

int main( )
{
    int* scores; /* tracks malloc-ed score storage */
    int i, n, max, a_count;

    /* Get number of values to malloc( ) and read */
    printf("Please enter number of scores: ");
    scanf("%d", &n);

    /* Validate number entered by user. */
    if (n<=INT_MAX && n>0){
        /* Allocate the appropriate number of bytes */
        scores = (int *)malloc(sizeof(int)*n);
        if (scores != NULL){
            /* Read score values into array */
            for(i=0; i<n; i++)
            {
                printf("Enter value %d of %d: ", i+1, n);
                scanf("%d", &scores[i]);
            }

            /* Find maximum of values read. */
            max = scores[0];
            for(i=1; i<n; i++)
            {
                if (scores[i] > max)
                    max = scores[i];
            }

            printf("Max score = %d\n", max);

            /* Count number of A's, scores greater than 90 */
            a_count = 0;
            for(i=0; i<n; i++)
            {
                if (scores[i] > 90)
                    a_count++;
            }

            printf("A's = %d\n", a_count);

            free(scores);
        }
        else
            printf("Malloc( ) request failed!\n");
    }
    else
        printf("Invalid allocation request: %d\n", n);

    return 0;
}

cont...
/* OUTPUT: dynamic3.c

Please enter number of scores: 4
Enter value 1 of 4: 75
Enter value 2 of 4: 85
Enter value 3 of 4: 95
Enter value 4 of 4: 92
Max score = 95
A's = 2

Please enter number of scores: 7
Enter value 1 of 7: 75
Enter value 2 of 7: 76
Enter value 3 of 7: 85
Enter value 4 of 7: 86
Enter value 5 of 7: 92
Enter value 6 of 7: 97
Enter value 7 of 7: 99
Max score = 99
A's = 3

*/
Ex:

```c
/*     FILE: dynamic4.c     */

/*
Prompts the user for the number of values
and uses malloc( ) to allocate the appropriate
amount of storage.

Passes the array/allocation to functions
for reading, displaying and for sorting.        */
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

int * selectionSort(int [], int);
int * printArray(int [], int);
int * readArray(int [], int);

int main(int argc, char *argv[])
{
    int *ar;
    int n;
    /* Get number of values to malloc( ) and read */
    printf("Please enter number of values: ");
    scanf("%d", &n);
    /* Validate number entered by user. */
    if (n<=INT_MAX && n>0){
        /* Allocate the appropriate number of bytes */
        ar = (int *)malloc(sizeof(int)*n);
        if (ar != NULL){
            readArray(ar, n);
            printf("\nOriginal array:\n");
            printArray(ar, n);
            selectionSort(ar, n);
            printf("\nSorted array:\n");
            printArray(ar, n);
            free(ar);
        } else
            printf("Malloc( ) request failed!\n");
    } else
        printf("Invalid allocation request: \d\n", n);
    return 0;
}
```

cont…
int * selectionSort(int array[], int size)
{
    int pass, item, position, temp;

    /* Selection-sort the values read in. */
    for(pass=0; pass<size-1; pass++)
    {
        position = pass;
        for(item=pass+1; item<size; item++)
        {
            if (array[position] < array[item])
                position = item;
        }
        if(pass != position){
            temp = array[pass];
            array[pass] = array[position];
            array[position] = temp;
        }
    }

    return array;
}

int * printArray(int a[], int s)
{
    int i;

    for(i=0; i<s; i++) /* display values in array */
        printf("%d\n", a[i]);

    return a;
}

int * readArray(int a[], int s)
{
    int i; /* Read score values into array */
    for(i=0; i<s; i++)
    {
        printf("Enter value %d of %d: ", i+1, s);
        scanf("%d", &a[i]);
    }

    return a;
}

/* OUTPUT: dynamic4.c
    Please enter number of values: 4
    Enter value 1 of 4: 75
    Enter value 2 of 4: 85
    Enter value 3 of 4: 95
    Enter value 4 of 4: 92

    Original array:
    75
    85
    95
    92

    Sorted array:
    95
    92
    85
    75
    cont...
Please enter number of values: 7
Enter value 1 of 7: 75
Enter value 2 of 7: 76
Enter value 3 of 7: 85
Enter value 4 of 7: 86
Enter value 5 of 7: 92
Enter value 6 of 7: 97
Enter value 7 of 7: 99

Original array:
75
76
85
86
92
97
99

Sorted array:
99
97
92
86
85
76
75

*/
DYNAMIC MULTIDIMENSIONAL ARRAYS

- Multidimensional arrays can be allocated dynamically.
- Pointers of the correct type must be defined in order to utilize the multidimensional array using standard array notation.
- Only the first dimension of a dynamic multidimensional array can be variable.
- Truly dynamic multidimensional arrays can be created; but then the compiler cannot be as helpful with the offsets computed when indexing. The programmer is therefore completely responsible for computing positions from indices. (We will not look at these types of arrays here.)
Ex:

```c
/* FILE: dynamic5.c */
/*
   Dynamically allocating a 2-D array.

   Notice the pointer definitions and the cast from malloc.
*/
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

void printArray(int ar[][2], int s);

int main(int argc, char *argv[])
{
    int (*ar)[2];
    int rows, r, c;

    /* Get number of rows for the n x 2 2-D array */
    printf("Please enter number of rows: ");
    scanf("%d", &rows);

    /* Validate number entered by user. */
    if (rows<=INT_MAX && rows>0){
        /* Allocate the appropriate number of bytes */
        ar = (int (*[2])malloc(sizeof(int)*rows * 2);

        if (ar != NULL){
            for(r=0; r<rows; r++)
                for(c=0; c<2; c++)
                    ar[r][c] = (2*r) + c;    /* Use the 2-D array */

            printArray(ar, rows);
            free(ar);
        } else
            printf("Malloc( ) request failed!\n");
    else
        printf("Invalid allocation request: %d\n", rows);
    return 0;
}

void printArray(int ar[][2], int rows)
{
    int r, c;

    printf(" ");
    for(c=0; c<2; c++)
        printf(" [ ]\[%d\]", c);
    printf("\n");

    for(r=0; r<rows; r++)
    {
        printf("ar[%d][ ] =", r);
        for(c=0; c<2; c++)
            printf(" %3d ", ar[r][c]);
        printf("\n");
    }
    return;
}
```

```
/* OUTPUT: dynamic5.c

Please enter number of rows: 3
[ ][0] [ ][1]
ar[0][ ] = 0 1
ar[1][ ] = 2 3
ar[2][ ] = 4 5

Please enter number of rows: 7
[ ][0] [ ][1]
ar[0][ ] = 0 1
ar[1][ ] = 2 3
ar[2][ ] = 4 5
ar[3][ ] = 6 7
ar[4][ ] = 8 9
ar[5][ ] = 10 11
ar[6][ ] = 12 13

*/
Ex:

```c
/*     FILE: dynamic6.c     */

/*
Dynamically allocating a 2-D array.
Notice the pointer definitions and the
cast from malloc.
*/
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define COLUMNS 4

void printArray(int ar[][COLUMNS], int s);

int main(int argc, char *argv[])
{
    int (*ar)[COLUMNS];
    int rows, r, c;

    /* Get number of rows for the n x COLUMNS 2-D array */
    printf("Please enter number of rows: ");
    scanf("%d", &rows);

    /* Validate number entered by user. */
    if (rows<=INT_MAX && rows>0)
    {
        /* Allocate the appropriate number of bytes */
        ar = (int (*)[COLUMNS])malloc(sizeof(int)*rows * COLUMNS);

        if (ar != NULL)
        {
            /* Use the 2-D array */
            for(r=0; r<rows; r++)
                for(c=0; c<COLUMNS; c++)
                    ar[r][c] = (COLUMNS*r) + c;

            printArray(ar, rows);
            free(ar);
        }
        else
            printf("Malloc( ) request failed!\n");
    }
    else
        printf("Invalid allocation request: %d\n", rows);
    return 0;
}
cont…
```
void printArray(int ar[ ][COLUMNS], int rows)
{
    int r, c;

    printf("           ");
    for(c=0; c<COLUMNS; c++)
        printf(" [%d]", c);
    printf("\n");

    for(r=0; r<rows; r++)
    {
        printf("ar[%d][ ] =", r);
        for(c=0; c<COLUMNS; c++)
        {
            printf(" %3d ", ar[r][c]);
        }
        printf("\n");
    }

    return;
}

/* OUTPUT: dynamic6.c

Please enter number of rows: 3
[ ][0] [ ][1] [ ][2] [ ][3]
ar[0][ ] =   0   1   2   3
ar[1][ ] =   4   5   6   7
ar[2][ ] =   8   9  10  11

Please enter number of rows: 7
[ ][0] [ ][1] [ ][2] [ ][3]
ar[0][ ] =   0   1   2   3
ar[1][ ] =   4   5   6   7
ar[2][ ] =   8   9  10  11
ar[3][ ] =  12  13  14  15
ar[4][ ] =  16  17  18  19
ar[5][ ] =  20  21  22  23
ar[6][ ] =  24  25  26  27
*/
Ex:

```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define COLUMNS 10

void printArray(int ar[][COLUMNS], int s);

int main(int argc, char *argv[])
{
    int (*ar)[COLUMNS];
    int rows, r, c;

    /* Get number of rows for the n x COLUMNS 2-D array */
    printf("Please enter number of rows: ");
    scanf("%d", &rows);

    /* Validate number entered by user. */
    if (rows<=INT_MAX && rows>0){
        /* Allocate the appropriate number of bytes */
        ar = (int (*)[COLUMNS])malloc(sizeof(int)*rows * COLUMNS);

        if (ar != NULL){
            for(r=0; r<rows; r++)
                for(c=0; c<COLUMNS; c++)
                    ar[r][c] = (COLUMNS*r) + c;    /* Use the 2-D array */

            printArray(ar, rows);
            free(ar);
        }
        else
            printf("Malloc( ) request failed!\n");
    }
    else
        printf("Invalid allocation request: %d\n", rows);
    return 0;
}
```

cont...
void printArray(int ar[ ][COLUMNS], int rows)
{
    int r, c;
    printf(" ");
    for(c=0; c<COLUMNS; c++)
        printf(" [ ][%d]", c);
    printf("\n");
    for(r=0; r<rows; r++)
    {
        printf("ar[%d][ ] =", r);
        for(c=0; c<COLUMNS; c++)
        {
            printf(" %3d ", ar[r][c]);
        }
        printf("\n");
    }
    return;
}

/*    OUTPUT: dynamic7.c

    Please enter number of rows: 3
           [ ][0] [ ][1] [ ][2] [ ][3] [ ][4] [ ][5] [ ][6] [ ][7] [ ][8] [ ][9]
    ar[0][ ] = 0  1  2  3  4  5  6  7  8  9
    ar[1][ ] = 10 11 12 13 14 15 16 17 18 19
    ar[2][ ] = 20 21 22 23 24 25 26 27 28 29

    Please enter number of rows: 7
           [ ][0] [ ][1] [ ][2] [ ][3] [ ][4] [ ][5] [ ][6] [ ][7] [ ][8] [ ][9]
    ar[0][ ] = 0  1  2  3  4  5  6  7  8  9
    ar[1][ ] = 10 11 12 13 14 15 16 17 18 19
    ar[2][ ] = 20 21 22 23 24 25 26 27 28 29
    ar[3][ ] = 30 31 32 33 34 35 36 37 38 39
    ar[4][ ] = 40 41 42 43 44 45 46 47 48 49
    ar[5][ ] = 50 51 52 53 54 55 56 57 58 59
    ar[6][ ] = 60 61 62 63 64 65 66 67 68 69

*/
Ex:

/* FILE: dynamic8.c */

/*
   Dynamically allocating a 3-D array.
   Notice the pointer definitions and the
   cast from malloc.
 */

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define COLUMNS 4
#define ROWS 3

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

#define COLUMNS 4
#define ROWS 3

void printArray(int ar[][ROWS][COLUMNS], int layers);

int main(int argc, char *argv[])
{
    int (*ar)[ROWS][COLUMNS];
    int layers, r, c, l;

    /* Get number of rows for the n x ROWS x COLUMNS 3-D array */
    printf("Please enter number of \%d x \%d layers: ", ROWS, COLUMNS);
    scanf("\%d", &layers);

    /* Validate number entered by user. */
    if (layers<=INT_MAX && layers>0){
        /* Allocate the appropriate number of bytes */
        ar = (int (**)[ROWS][COLUMNS])malloc(sizeof(int)*layers * ROWS * COLUMNS);

        if (ar != NULL){
            /* Use the 3-D array */
            for(l=0; l<layers; l++)
                for(r=0; r<ROWS; r++)
                    for(c=0; c<COLUMNS; c++)
                        ar[l][r][c] = r + c + l*10;

            printArray(ar, layers);
            free(ar);
        }
        else
            printf("Malloc( ) request failed!\n");
    }
    else
        printf("Invalid allocation request: \%d\n", layers);

    return 0;
}

cont…
```c
void printArray(int ar[][ROWS][COLUMNS], int layers)
{
    int r, c, l;

    for(l=0; l<layers; l++)
    {
        printf("\n");
        printf("\n");
        for(c=0; c<COLUMNS; c++)
            printf(" [%d][ ][%d]", l, c);
        printf("\n");

        for(r=0; r<ROWS; r++)
        {
            printf("ar[%d][%d][ ] =", l, r);
            for(c=0; c<COLUMNS; c++)
            {
                printf(" %3.2d ", ar[l][r][c]);
            }
            printf("\n");
        }
    }

    return;
}

/* OUTPUT: dynamic8.c

Please enter number of 3 x 4 layers: 3

[0][ ][0][ ][ ]][1][ ][0][ ][2][ ][0][ ][3]
ar[0][0][ ] = 00 01 02 03
ar[0][1][ ] = 01 02 03 04
ar[0][2][ ] = 02 03 04 05

[1][ ][0][ ][1][ ][1][ ][2][ ][1][ ][3]
ar[1][0][ ] = 10 11 12 13
ar[1][1][ ] = 11 12 13 14
ar[1][2][ ] = 12 13 14 15

[2][ ][0][ ][2][ ][1][ ][2][ ][2][ ][3]
ar[2][0][ ] = 20 21 22 23
ar[2][1][ ] = 21 22 23 24
ar[2][2][ ] = 22 23 24 25
cont...*/
Please enter number of 3 x 4 layers: 7

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
</tr>
<tr>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
</tr>
<tr>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
</tr>
<tr>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
</tr>
<tr>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
</tr>
<tr>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
</tr>
</tbody>
</table>

*
TEXT FILE I/O

- Basic text file I/O is only slightly more difficult than the I/O done to date.
- Every I/O function seen so far has a sister function that will read/write to a file on disk.
- The programmers connection to a file on disk is a file name. The C connection to a file on disk is a file pointer, `FILE *`. The first step in doing file I/O is to translate a filename into a C file pointer using `fopen( )`.
- The file pointer is then passed to the file I/O function we are using so that C can access the appropriate file.
- Finally the connection to the file is severed by calling `fclose( )` with the file pointer as a parameter.

Ex:

```c
/*     FILE: FileIO.c     */
/* Basic output using printf( ) */
#include <stdio.h>

int main( )
{
    int x = 7;
    double y = 7.25;

    printf("This data will be written to the screen.\n");
    printf("x = %d, y = %f\n", x, y);

    return 0;
}

/*     OUTPUT: FileIO.c     */
This data will be written to the screen.
x = 7, y = 7.250000
*/
```
Ex:

/*     FILE: FileIO_2.c     */

/* Basic output to a file using fprintf( ) */
#include <stdio.h>

int main( )
{
    FILE *fptr;
    int x = 7;
    double y = 7.25;
    fptr = fopen("FileIO_2.out","w");
    fprintf(fptr,"This data will be written to a file.\n";
    fprintf(fptr,"x = %d, y = %f\n", x, y);
    fclose(fptr);
    return 0;
}

/*   OUTPUT: FileIO_2.out
    This data will be written to a file.
    x = 7, y = 7.250000
   */

Ex:

/*     FILE: FileIO_3.c     */

/* Text output using fprintf() */
#include <stdio.h>

int main( )
{
    FILE *fptr;
    int x = 7;
    double y = 7.25;
    fptr = fopen("FileIO_3.out","w");
    if(fptr != NULL){
        fprintf(fptr,"This data will be written to a file.\n";
        fprintf(fptr,"x = %d, y = %f\n", x, y);
        fclose(fptr);
    }
    else
        printf("Unable to open file.\n");
    return 0;
}

/*   OUTPUT: FileIO_3.out
    This data will be written to a file.
    x = 7, y = 7.250000
   */
Ex:

```c
/*     FILE: FileIO_4.c     */

/* Text I/O using fprintf( ) and fscanf( ) */
#include <stdio.h>

int main( )
{
    FILE *fptr;
    int i, x;

    fptr = fopen("FileIO_4.out","w");
    if(fptr != NULL){
        for(i=0; i<5; i++)
            fprintf(fptr,"%d\n", i);
        fclose(fptr);
    } else
        printf("Unable to open file.\n");

    fptr = fopen("FileIO_4.out","r");
    if(fptr != NULL){
        for(i=0; i<5; i++)
            fscanf(fptr,"%d", &x);
        printf("Read: %d\n", x);
    }
    fclose(fptr);
} else
    printf("Unable to open file.\n");

return 0;
}

/*    OUTPUT: FileIO_4.c    */
Read: 0
Read: 1
Read: 2
Read: 3
Read: 4

/* OUTPUT: FileIO_4.out */

0
1
2
3
4
*/
Ex:

```c
/*     FILE: myCopy.c     */
/* A file copy program. Copies a file from the first command-line argument to the second. */
#include <stdio.h>

int main(int argc, char *argv[]) {
    FILE *inptr, *outptr;
    int ch;
    inptr = fopen(argv[1], "r");
    outptr = fopen(argv[2], "w");
    while((ch = getc(inptr)) != EOF) putc(ch, outptr);
    fclose(outptr);
    fclose(inptr);
    return 0;
}

/*    OUTPUT: myCopy.c    */

 COMMAND LINE: myCopy myCopy.c myCopy.txt

 */
```
Ex:

```c
/*     FILE: myCopy2.c     */

/* A file copy program. Does error-checking on the file pointers. */

#include <stdio.h>

int main(int argc, char *argv[])
{
    FILE *inptr, *outptr;
    int ch;

    if(argc == 3){
        inptr = fopen(argv[1],"r");
        if(inptr){
            if(outptr = fopen(argv[2],"w")){
                while((ch = getc(inptr)) != EOF)
                    putc(ch,outptr);
                fclose(outptr);
            } else
                printf("Error - Unable to open output file: %s\n", argv[2]);
            fclose(inptr);
        } else
            printf("Error - Unable to open input file: %s\n", argv[1]);
    } else
        printf("Usage: myCopy2 <input file> <output file> \n");
    return 0;
}

/*     OUTPUT: myCopy2.c     */

COMMAND LINE: myCopy2 myCopy2.c myCopy2.txt

/*     */
BINARY FILE I/O

- Binary file I/O writes data from memory to disk in the same format as it is stored in memory.
- Generally is is not going to be human-readable but it should take up less space and can be done faster since it does not need to be translated into text.
- File pointers are used in the same manner as they are in text I/O.

Ex:

```c
/* FILE: FileIO_5.c */
/* Binary I/O using fwrite( ) and fread( ) */
#include <stdio.h>

int main( )
{
    FILE *fptr;
    int i, x;
    x = 0;
    i = 7;

    printf("i = %d   x = %d\n", i, x);
    fptr = fopen("tmp.dat","w");
    if(fptr != NULL){
        fwrite(&i, 4, 1, fptr);
        fclose(fptr);
    } else
        printf("Unable to open file for write.\n");
    fptr = fopen("tmp.dat","r");
    if(fptr != NULL){
        fread(&x, sizeof(int), 1, fptr);
        fclose(fptr);
    } else
        printf("Unable to open file for read.\n");
    printf("i = %d   x = %d\n", i, x);
    return 0;
}

/* OUTPUT: FileIO_5.c

   i = 7   x = 0
   i = 7   x = 7
*/
```
Ex:

```c
/*     FILE: FileIO_6.c     */

/* Binary I/O using fwrite( ) and fread( ) */
#include <stdio.h>

int main( )
{
    FILE *fptr;
    int i, ar[5], ar2[5];
    for(i=0; i<5; i++)
        ar[i] = i*11;
    fptr = fopen("tmp.dat","w");
    if(fptr != NULL){
        fwrite(&ar[0], sizeof(int), 5, fptr);
        fclose(fptr);
    } else
        printf("Unable to open file for write.\n");
    fptr = fopen("tmp.dat","r");
    if(fptr != NULL){
        fread(ar2, sizeof(int), 5, fptr);
        fclose(fptr);
    } else
        printf("Unable to open file for read.\n");
    for(i=0; i<5; i++)
        printf("ar2[%d] = %d\n", i, ar2[i]);
    return 0;
}

/* OUTPUT: FileIO_6.c */

ar2[0] = 0
ar2[1] = 11
ar2[2] = 22
ar2[3] = 33
ar2[4] = 44
```

/*
STRUCTURES

- An array in C is a set or group of storage locations that are all of the same type.
- A structure in C allows a group of storage locations that are of different types to be created and treated as single unit.
- Structures are termed a data “aggregate”, since pieces of differing types are grouped together in a structure. These pieces are referred to as “members” of the structure.
- Structures are NOT the same as arrays because a structure itself is treated as a single entity and a structure’s name refers to the entire structure. (With an array, the array name is just the address of the first element in the set.)
- A structure definition creates the equivalent of a new data type. Any place you use a basic C data type you can use a structure. They can be passed as parameters, used as return values, you can take the address of one, C can compute the `sizeof` one.
- Since a structure we define is essentially a new data type, no existing C functions or operators were designed with our definitions in mind. So `printf( )` and `scanf( )` have no conversion specifiers for them, and the arithmetic operators won’t operate on them. But we can write our own functions to perform any of these operations.
- Some basic operators do still work with structures, `&` address-of, `sizeof( )`, `*` dereference, `=` assignment, `(type)` type cast.
- There are also two operators just for structure operations. The `. ` member access operator and the `->` member access thru a pointer operator.
Ex:

/* FILE: struct1.c */

/* Defining and using a structure. */

#include <stdio.h>

int main()
{
    struct part
    {
        char name[124];
        long no;
        double price;
    };

    struct part board;

    strcpy(board.name,"I/O card");
    board.no = 127356;
    board.price = 99.50;

    printf("Product: %s\n", board.name);
    printf("Part No.: %ld\n", board.no);
    printf("Unit price: %.2f\n", board.price);

    return 0;
}

/* OUTPUT: struct1.c */

    Product: I/O card
    Part No.: 127356
    Unit price: 99.50

*/
Ex:

```c
/*     FILE: struct2.c     */

/* Defining and using a structure. */

#include <stdio.h>

struct part{
    char name[124];
    long no;
    double price;
};

int main( )
{
    struct part board;
    
    strcpy(board.name,"I/O card");
    board.no = 127356;
    board.price = 99.50;

    printf("Product: %s\n", board.name);
    printf("Part No.: %ld\n", board.no);
    printf("Unit price: %.2f\n", board.price);

    printf("\n\n");
    printf("struct part size: %d\n", sizeof(struct part));
    printf("board size: %d\n", sizeof(board));
    printf("    board at: %p\n", &board);
    printf("    board.name at: %p\n", &board.name);
    printf("    board.no at: %p\n", &board.no);
    printf("      %p = %X + %X \n", 
        &board.no, &board.name, sizeof(board.name));
    printf("    board.price at: %p\n", &board.price);
    printf("      %p = %X + %X \n", 
        &board.price, &board.no, sizeof(board.no));

    return 0;
}

/*    OUTPUT: struct2.c    */

Product: I/O card
Part No. : 127356
Unit price: 99.50

struct part size: 136
board size: 136
    board at: 0022FED8
    board.name at: 0022FED8
        0022FED8 = 22FED8 + 7C
    board.no at: 0022FF54
        0022FF54 = 22FF54 + 4
    board.price at: 0022FF58
        0022FF58 = 22FF58 + 4

*/
Ex:

/*     FILE: struct3.c     */

/* Structures are like basic data types. */

#include <stdio.h>

struct part{
    char name[124];
    long no;
    double price;
};

int main( )
{
    struct part board;
    struct part board2;

    strcpy(board.name,"I/O card");
    board.no = 127356;
    board.price = 99.50;

    board2 = board; /* assign one structure to another. */

    printf("Product: %s\n", board.name);
    printf("Part No.: %ld\n", board.no);
    printf("Unit price: %.2f\n", board.price);

    printf("\n\n");
    printf("Board2 - Product: %s\n", board2.name);
    printf("Board2 - Part No.: %ld\n", board2.no);
    printf("Board2 - Unit price: %.2f\n", board2.price);

    return 0;
}

/* OUTPUT: struct3.c

Product: I/O card
Part No.: 127356
Unit price: 99.50

Board2 - Product: I/O card
Board2 - Part No.: 127356
Board2 - Unit price: 99.50

*/
Ex:

```c
/*     FILE: struct4.c     */

/* Structures are like basic data types. You can pass them to a function and the entire structure is passed. */

#include <stdio.h>

struct part{
    char name[124];
    long no;
    double price;
};

void print_part(struct part p);

int main()
{
    struct part board;
    struct part board2;

    strcpy(board.name,"I/O card");
    board.no = 127356;
    board.price = 99.50;

    board2 = board;   /* assign one structure to another. */
    print_part(board); /* Print part structures with a function. */

    printf("\n\n");
    print_part(board2);

    return 0;
}

void print_part(struct part p)
{
    printf("Product: %s\n", p.name);
    printf("Part No.: %ld\n", p.no);
    printf("Unit price: %.2f\n", p.price);

    return;
}

/*    OUTPUT: struct4.c
     */

Product: I/O card
Part No.: 127356
Unit price: 99.50

Product: I/O card
Part No.: 127356
Unit price: 99.50
*/
Ex:

```c
/* FILE: struct5.c */

/* Arrays can be used with structures just like any other
 C data type. */

#include <stdio.h>

#define SIZE 5

struct part{
    char name[124];
    long no;
    double price;
};

void print_part(struct part p);

int main( )
{
    struct part board; /* One "part" */
    struct part inventory[SIZE]; /* Array to hold SIZE "part"s */

    int i;
    for(i=0; i < SIZE; i++) { /* Load the array of structures. */
        sprintf(board.name, "I/O card #%d", i);
        board.no = 127356 + i;
        board.price = 99.50 + i*3;

        inventory[i] = board;
    }

    for(i=0; i < SIZE; i++) { /* Display the array of structures. */
        print_part(inventory[i]);
        printf("\n");
    }

    return 0;
}

void print_part(struct part p)
{
    printf("Product: %s\n", p.name);
    printf("Part No.: %ld\n", p.no);
    printf("Unit price: %.2f\n", p.price);

    return;
}

cont...
```
OUTPUT: struct5.c

    Product: I/O card #0
    Part No.: 127356
    Unit price: 99.50

    Product: I/O card #1
    Part No.: 127357
    Unit price: 102.50

    Product: I/O card #2
    Part No.: 127358
    Unit price: 105.50

    Product: I/O card #3
    Part No.: 127359
    Unit price: 108.50

    Product: I/O card #4
    Part No.: 127360
    Unit price: 111.50

*/
Ex:

```c
/*     FILE: struct6.c     */

/* The address of a structure can be passed to a function          */
 /* just like any other C data type.                              */

#include <stdio.h>

#define SIZE 5

struct part{
    char name[124];
    long no;
    double price;
};

void print_part(struct part* p);

int main( )
{
    struct part board;              /* One "part" */
    struct part inventory[SIZE];    /* Array to hold SIZE "part"s */
    int i;
    for(i=0; i < SIZE; i++){}        /* Load the array of structures. */
        sprintf(board.name,"I/O card %d", i);
        board.no = 127356 + i;
        board.price = 99.50 + i*3;

        inventory[i] = board;
    }

    print_part(&board);             /* print_part( ) expects an address. */
    printf("\n");

    for(i=0; i < SIZE; i++){}        /* Display the array of structures. */
        print_part(&inventory[i]);
    printf("\n");

    return 0;
}

void print_part(struct part* p)
{
    printf("Product: %s\n", (*p).name);
    printf("Part No.: %ld\n", (*p).no);
    printf("Unit price: %.2f\n", (*p).price);

    return;
}

cont…
/* OUTPUT: struct6.c

Product: I/O card #4
Part No.: 127360
Unit price: 111.50

Product: I/O card #0
Part No.: 127356
Unit price: 99.50

Product: I/O card #1
Part No.: 127357
Unit price: 102.50

Product: I/O card #2
Part No.: 127358
Unit price: 105.50

Product: I/O card #3
Part No.: 127359
Unit price: 108.50

Product: I/O card #4
Part No.: 127360
Unit price: 111.50

*/
Ex:
/*     FILE: struct7.c     */

/* The address of a structure can be passed to a function
   just like any other C data type. */
#include <stdio.h>
#define SIZE 5

struct part{
    char name[124];
    long no;
    double price;
};

void print_part(struct part* p);

int main( ){
    struct part board; /* One "part" */
    struct part inventory[SIZE]; /* Array to hold SIZE "part"s */
    int i;

    for(i=0; i < SIZE; i++) {
        /* Load the array of structures. */
        sprintf(board.name,"I/O card #%d", i);
        board.no = 127356 + i;
        board.price = 99.50 + i*3;
        inventory[i] = board;
    }

    print_part(&board); /* print_part( ) expects an address. */
    printf("\n");

    for(i=0; i < SIZE; i++){
        /* Display the array of structures. */
        print_part(inventory + i); /* Don't need to ask for the address */
        printf("\n"); /* ... since an array name is already */
        /* ... an address. */
    }

    return 0;
}

void print_part(struct part* p) /* -> operator simplifies access thru */
{ /* ... a pointer. */
    printf("Product: %s\n", p->name);
    printf("Part No.: %ld\n", p->no);
    printf("Unit price: %.2f\n", p->price);

    return;
}

cont…
/* OUTPUT: struct7.c

Product: I/O card #4
Part No.: 127360
Unit price: 111.50

Product: I/O card #0
Part No.: 127356
Unit price: 99.50

Product: I/O card #1
Part No.: 127357
Unit price: 102.50

Product: I/O card #2
Part No.: 127358
Unit price: 105.50

Product: I/O card #3
Part No.: 127359
Unit price: 108.50

Product: I/O card #4
Part No.: 127360
Unit price: 111.50

*/
Ex:

```c
/*     FILE:  struct8.c     */
/* Reading data into a structure. */

#include <stdio.h>
#define SIZE 5

struct part{
    char name[124];
    long no;
    double price;
};

void print_part(struct part* p);
struct part read_part(void);

int main( )
{
    struct part inventory[SIZE]; /* Array to hold SIZE "part"s */

    int i;
    for(i=0; i < SIZE; i++) {  /* Load the array of structures. */
        inventory[i] = read_part( );
    }

    for(i=0; i < SIZE; i++) {  /* Display the array of structures. */
        print_part(inventory + i); /* Don't need to ask for the address */
        printf("\n");  /* ... since an array name is already */
    }  /* ... an address. */

    return 0;
}

void print_part(struct part* p) /* -> operator simplifies access thru */
{  /* ... a pointer. */
    printf("Product: %s\n", p->name);
    printf("Part No.: %ld\n", p->no);
    printf("Unit price: %.2f\n", p->price);

    return;
}

struct part read_part(void) /* returns the structure. */
{  /* Strip trailing newline */
    struct part temp;

    gets(temp.name);
    scanf("%ld", &temp.no);
    scanf("%lf", &temp.price);
    getchar( );

    return temp;
}

cont…
/* OUTPUT: struct8.c

Product: I/O card #0
Part No.: 127356
Unit price: 99.50

Product: Network card #1
Part No.: 127357
Unit price: 102.50

Product: USB card #2
Part No.: 127358
Unit price: 105.50

Product: Fax card #3
Part No.: 127359
Unit price: 108.50

Product: Modem card #4
Part No.: 127360
Unit price: 111.50

INPUT:

I/O card #0
127356
99.50
Network card #1
127357
102.50
USB card #2
127358
105.50
Fax card #3
127359
108.50
Modem card #4
127360
111.50

*/
Ex:

/*     FILE:  struct9.c     */
/* Reading data into a structure. 
Passing the address to store into. */

#include <stdio.h>
#define SIZE 5

struct part{
    char name[124];
    long no;
    double price;
};

void print_part(struct part* p);
void read_part(struct part*);

int main( )
{
    struct part inventory[SIZE]; /* Array to hold SIZE "part"s */
    int i;
    for(i=0; i < SIZE; i++){
        read_part(inventory+i);
    }
    for(i=0; i < SIZE; i++){
        print_part(inventory + i);
    }
    return 0;
}

void print_part(struct part* p) /* -> operator simplifies access thru */
    /* ... a pointer. */
{
    printf("Product: %s\n", p->name);
    printf("Part No.: %ld\n", p->no);
    printf("Unit price: %.2f\n", p->price);
    return;
}

void read_part(struct part* temp) /* returns the structure. */
{
    gets(temp->name);
    scanf("%ld", &temp->no);
    scanf("%lf", &temp->price);
    getchar( ); /* Strip trailing newline */
    return;
}

cont…
/* OUTPUT: struct9.c

Product: I/O card #0
Part No.: 127356
Unit price: 99.50

Product: Network card #1
Part No.: 127357
Unit price: 102.50

Product: USB card #2
Part No.: 127358
Unit price: 105.50

Product: Fax card #3
Part No.: 127359
Unit price: 108.50

Product: Modem card #4
Part No.: 127360
Unit price: 111.50

INPUT:

I/O card #0
127356
99.50
Network card #1
127357
102.50
USB card #2
127358
105.50
Fax card #3
127359
108.50
Modem card #4
127360
111.50

*/
Ex:

```c
/*     FILE:  struct10.c     */
/* Reading data into a structure.     */
/* Monitor input for success. */

#include <stdio.h>

#define SIZE 5

struct part{
    char name[124];
    long no;
    double price;
};

void print_part(struct part* p);
int read_part(struct part*);

int main( )
{
    struct part inventory[SIZE];    /* Array to hold SIZE "part"s */
    int i;
    int count = 0;
    for(i=0; i < SIZE && read_part(inventory+i) != EOF; i++)        /* Load the array of 
structures. */
        count++;
    for(i=0; i < count; i++)        /* Display the array of structures. */
        print_part(inventory + i);    /* Don't need to ask for the address */
        printf("
");                 /* ... since an array name is already */
    return 0;
}

void print_part(struct part* p)   /* -> operator simplifies access thru */
    {                                 /* ... a pointer.                      */
        printf("Product: %s
", p->name);
        printf("Part No.: %ld
", p->no);
        printf("Unit price: %.2f
", p->price);
        return;
    }

int read_part(struct part* temp)  /* returns the structure. */
    {                               /* */
        int result = 1;
        if(gets(temp->name) == NULL)
            result = EOF;
        if(result == 1 && scanf("%ld", &temp->no) != 1)
            result = EOF;
        if(result == 1 && scanf("%lf", &temp->price) != 1)
            result = EOF;
        if(result == 1)
            getchar( );                  /* Strip trailing newline */
        return result;
    }

cont…
```
/ * OUTPUT: struct10.c

Product: I/O card #0
Part No.: 127356
Unit price: 99.50

Product: Network card #1
Part No.: 127357
Unit price: 102.50

Product: USB card #2
Part No.: 127358
Unit price: 105.50

Product: Fax card #3
Part No.: 127359
Unit price: 108.50

Product: Modem card #4
Part No.: 127360
Unit price: 111.50

INPUT:

I/O card #0
127356
99.50
Network card #1
127357
102.50
USB card #2
127358
105.50
Fax card #3
127359
108.50
Modem card #4
127360
111.50

Product: Network card #1
Part No.: 127357
Unit price: 102.50

Product: USB card #2
Part No.: 127358
Unit price: 105.50

INPUT:

Network card #1
127357
102.50
USB card #2
127358
105.50

*/
Ex:

```c
/*     FILE:  struct11.c     */
/* Reading data into a structure.
    Monitor input for success.
    Capitalize on for loop features and comma operator*/

#include <stdio.h>
#define SIZE 5
struct part{
    char name[124];
    long no;
    double price;
};
void print_part(struct part* p);
int read_part(struct part*);
int main( )
{
    struct part inventory[SIZE];    /* Array to hold SIZE "part"s */
    int i;
    int count;
    /* Load the array of structures. */
    for(i=0, count=0; i < SIZE && read_part(inventory+i) != EOF; i++, count++)
    {
        for(i=0; i < count; i++)
        {
            printf("\n");
        }
        /* Display the array of structures. */
        print_part(inventory + i); /* Don't need to ask for the address */
        printf("\n"); /* ... since an array name is already */
        /* ... an address. */
    }
    return 0;
}
void print_part(struct part* p) /* -> operator simplifies access thru */
{
    printf("Product: %s\n", p->name);
    printf("Part No.: %ld\n", p->no);
    printf("Unit price: %.2f\n", p->price);
    return;
}
int read_part(struct part* temp) /* returns the structure. */
{
    int result = 1;
    if(gets(temp->name) == NULL)
        result = EOF;
    if(result == 1 && scanf("%ld", &temp->no) != 1)
        result = EOF;
    if(result == 1 && scanf("%lf", &temp->price) != 1)
        result = EOF;
    if(result == 1)
        getchar( ); /* Strip trailing newline */
    return result;
}
cont...
```
/* OUTPUT: struct11.c 
   
   Product: I/O card #0
   Part No.: 127356
   Unit price: 99.50

   Product: Network card #1
   Part No.: 127357
   Unit price: 102.50

   Product: USB card #2
   Part No.: 127358
   Unit price: 105.50

   Product: Fax card #3
   Part No.: 127359
   Unit price: 108.50

   Product: Modem card #4
   Part No.: 127360
   Unit price: 111.50

   INPUT:

   I/O card #0
   127356
   99.50
   Network card #1
   127357
   102.50
   USB card #2
   127358
   105.50
   Fax card #3
   127359
   108.50
   Modem card #4
   127360
   111.50

   Product: Network card #1
   Part No.: 127357
   Unit price: 102.50

   Product: USB card #2
   Part No.: 127358
   Unit price: 105.50

   INPUT:

   Network card #1
   127357
   102.50
   USB card #2
   127358
   105.50
*/
Ex:

```c
/*     FILE: struct12.c     */
/* Binary file I/O - Reading/writing structs */
#include <stdio.h>
#define SIZE 5
struct part{
    char name[124];
    long no;
    double price;
};
void print_part(const struct part * const);

int main( ) {
    struct part board;              /* One "part" */
    struct part inventory[SIZE];    /* Array to hold SIZE "part"s */
    FILE * fp;
    char * filename = "structBin.bin";
    int i;
    for(i=0; i < SIZE; i++) {        /* Load the array of structures. */
        sprintf(board.name,"I/O card #%d", i);
        board.no = 127356 + i;
        board.price = 99.50 + i*3;
        inventory[i] = board;
    }
    fp = fopen(filename,"w");
    if(fp != NULL) {
        for(i=0; i < SIZE; i++) /* Write the structures. */
            fwrite(inventory + i, sizeof(struct part), 1, fp);
        fclose(fp);
        fp = fopen(filename,"r");
        if(fp != NULL) {
            for(i=SIZE-1; i >= 0; i--) /* Read the structures. */
                fread(inventory + i, sizeof(struct part), 1, fp);
            fclose(fp);
            for(i=0; i < SIZE; i++){ /* Display the array of structures. */
                print_part(inventory + i);
                printf("\n");
            }
            else
                fprintf(stderr,"Unable to open file %s for read.\n", filename);
        }
        else
            fprintf(stderr,"Unable to open file %s for write.\n", filename);
        return 0;
    }
}
cont…
```
void print_part(const struct part * const p) /* Display function */
{
    printf("Product: %s\n", p->name);
    printf("Part No.: %ld\n", p->no);
    printf("Unit price: %.2f\n", p->price);

    return;
}

/*    OUTPUT: struct12.c

    Product: I/O card #4
    Part No.: 127360
    Unit price: 111.50

    Product: I/O card #3
    Part No.: 127359
    Unit price: 108.50

    Product: I/O card #2
    Part No.: 127358
    Unit price: 105.50

    Product: I/O card #1
    Part No.: 127357
    Unit price: 102.50

    Product: I/O card #0
    Part No.: 127356
    Unit price: 99.50

    */
Ex:

```c
/*     FILE: struct13.c     */
/* Binary file I/O - Reading/writing structs
   - writing sets/blocks of data */
#include <stdio.h>
#define SIZE 5

struct part{
    char name[124];
    long no;
    double price;
};

void print_part(const struct part * const);

int main( )
{
    struct part board;    /* One "part" */
    struct part inventory[SIZE];    /* Array to hold SIZE "part"s */

    FILE * fp;
    char * filename = "structBin.bin";
    int i;

    fp = fopen(filename,"w");
    if(fp != NULL){
        for(i=0; i < SIZE; i++){
            sprintf(board.name,"I/O card #%d", i);
            board.no = 127356 + i;
            board.price = 99.50 + i*3;
            fwrite(&board, sizeof(struct part), 1, fp);
        }
        fclose(fp);
        fp = fopen(filename,"r");
        if(fp != NULL){
            fread(inventory, sizeof(struct part), SIZE, fp);
            fclose(fp);
            for(i=0; i < SIZE; i++){
                print_part(inventory + i);
                printf("\n");
            }
        } else
            fprintf(stderr,"Unable to open file %s for read.\n", filename);
    } else
        fprintf(stderr,"Unable to open file %s for write.\n", filename);
    return 0;
}
```

cont…
void print_part(const struct part * const p) /* Display function */
{
    printf("Product: %s\n", p->name);
    printf("Part No.: %ld\n", p->no);
    printf("Unit price: %.2f\n", p->price);
    return;
}

/* OUTPUT: struct13.c */

    Product: I/O card #0
    Part No.: 127356
    Unit price: 99.50

    Product: I/O card #1
    Part No.: 127357
    Unit price: 102.50

    Product: I/O card #2
    Part No.: 127358
    Unit price: 105.50

    Product: I/O card #3
    Part No.: 127359
    Unit price: 108.50

    Product: I/O card #4
    Part No.: 127360
    Unit price: 111.50

*/
ENUMERATED TYPES

- enumerated types can be created to give symbolic names to integer values and enlist the compiler for type checking.

Ex:

```c
/*     FILE: enum.c     */
/* Enumerated types give symbolic constants with type checking. */
#include <stdio.h>
enum GPA{F,D,C,B,A};
int main( )
{
    enum GPA grade;
    grade = A;
    printf("Score for an 'A': %d\n", grade);
    grade = B;
    printf("Score for an 'B': %d\n", grade);
    grade = C;
    printf("Score for an 'C': %d\n", grade);
    grade = D;
    printf("Score for an 'D': %d\n", grade);
    grade = F;
    printf("Score for an 'F': %d\n", grade);
    return 0;
}

/*     OUTPUT: enum.c     */
Score for an 'A': 4
Score for an 'B': 3
Score for an 'C': 2
Score for an 'D': 1
Score for an 'F': 0
*/
```
Ex:

```c
/*     FILE: enum2.c     */
/* Enumerated types give symbolic constants with type 
   checking.                                          */
#include <stdio.h>
enum GPA{F,D,C,B,A};
int main( )
{
    enum GPA grade;
    grade = A;
    printf("Score for an 'A': %d\n", grade);
    grade = Q;
    printf("Score for an 'Q': %d\n", grade);
    return 0;
}

/*    OUTPUT: enum2.c  

enum2.c: In function 'main':
enum2.c:15: `Q' undeclared (first use in this function)
enum2.c:15: (Each undeclared identifier is reported only once
enum2.c:15: for each function it appears in.)

*/
```
Ex:

```c
/*     FILE: enum3.c     */
/* Enumerated types give symbolic constants with type 
   checking.                                          */
#include <stdio.h>
enum Direction{North=1, South, East, West};
int main( )
{
    enum Direction dir;
    dir = North;
    printf("Direction 'North': %d\n", dir);
    dir = South;
    printf("Direction 'South': %d\n", dir);
    dir = East;
    printf("Direction 'East': %d\n", dir);
    dir = West;
    printf("Direction 'West': %d\n", dir);
    return 0;
}

/*    OUTPUT: enum3.c 
   Direction 'North': 1
   Direction 'South': 2
   Direction 'East': 3
   Direction 'West': 4
   */
```
Ex:

```c
/*     FILE: enum4.c     */

/* Enumerated types give symbolic constants with type  */
/* checking.                                          */

#include <stdio.h>

enum Direction{North=1, South, East, West};

int main( )
{
    enum Direction dir;
    dir = South;

    switch(dir)
    {
        case North:
            printf("Direction %d is %s\n", dir, "North");
            break;

        case South:
            printf("Direction %d is %s\n", dir, "South");
            break;

        case East:
            printf("Direction %d is %s\n", dir, "East");
            break;

        case West:
            printf("Direction %d is %s\n", dir, "West");
            break;
    }

    return 0;
}

/*    OUTPUT: enum4.c    */

    Direction 2 is South

*/
```
Ex:

```c
/*     FILE: enum5.c     */

/* Enumerated types give symbolic constants with type checking. */
#include <stdio.h>
enum Direction{North=1, South, East, West};
char * directionString(enum Direction);

int main( )
{
    enum Direction dir;
    dir = East;
    printf("Direction %d is %s
",
            dir, directionString(dir));
    return 0;
}

char * directionString(enum Direction d)
{
    static char * dirs[ ] = { "North",
                              "South",
                              "East",
                              "West"  };
    return dirs[d-1];
}

/*    OUTPUT: enum5.c

    Direction 3 is East

*/
```
UNIONS

- A union allows multiple mappings of the same piece of storage.
- Only one is in effect at any given time, but the same piece of memory can be utilized differently using a different mapping defined by the union.

Ex:
```c
/* FILE: union.c */

/* A union that be either an array of
   ints or an array of floats, as needed. */

#include <stdio.h>
union intFloatArray{
    int iarray[5];
    float farray[5];
};

int main( )
{
    union intFloatArray ar;
    int i;
    for(i=0; i<5; i++)
        ar.iarray[i] = i*11;
    for(i=0; i<5; i++)
        printf("int[%d] = %d\n", i, ar.iarray[i]);
    for(i=0; i<5; i++)
        ar.farray[i] = i*1.1;
    for(i=0; i<5; i++)
        printf("float[%d] = %f\n", i, ar.farray[i]);

    printf("size of union int_float_array = %d\n", sizeof(union intFloatArray));
    printf("size of ar = %d", sizeof(ar));
    return 0;
}

/* OUTPUT: union.c
   int[0] = 0
   int[1] = 11
   int[2] = 22
   int[3] = 33
   int[4] = 44
   float[0] = 0.000000
   float[1] = 1.100000
   float[2] = 2.200000
   float[3] = 3.300000
   float[4] = 4.400000
   size of union int_float_array = 20
   size of ar = 20
*/
```
Ex:

```c
/*     FILE: union2.c     */

/* A union that allows byte-wise inspection of     */
/* a storage location.                           */

#include <stdio.h>
union intFloatArrayByte{
    int i;
    float f;
    unsigned char byte[4];
};

int main( )
{
    union intFloatArrayByte map;
    int i;
    map.i = 7;
    printf("int = %d\n", map.i);
    for(i=0; i<4; i++)
        printf("char[%d] = %X\n", i, map.byte[i]);
    return 0;
}

/*    OUTPUT: union2.c    */

    int = 7
    char[0] = 7
    char[1] = 0
    char[2] = 0
    char[3] = 0

/*    */
Ex:

```c
/*     FILE: union3.c     */
/* A union that allows byte-wise inspection of 
a storage location.                             */
#include <stdio.h>
union intFloatArrayByte{
    int i;
    float f;
    unsigned char byte[4];
};

int main( )
{
    union intFloatArrayByte map;
    int i;
    map.f = 7.0;
    printf("float = %f\n", map.f);
    for(i=0; i<4; i++)
        printf("char[%d] = %X\n", i, map.byte[i]);
    return 0;
}

/*    OUTPUT: union3.c

    float = 7.000000
    char[0] = 0
    char[1] = 0
    char[2] = E0
    char[3] = 40

    */
```
**TYPEDEF**

- C allows a type name to be defined using the *typedef* mechanism.
- The type defined used in situations where a standard C type would be used.
- *typedef* is often used to shorten the *struct name* type associated with a structure definition.

Ex:

```c
/*     FILE: struct14.c     */
/* Typedef - simplified naming */
#include <stdio.h>
#define SIZE 5
struct part{
    char name[124];
    long no;
    double price;
};
typedef struct part part;     /* part becomes the type of "struct part" */
void print_part(const struct part * const);
int main( )
{
    part board;              /* One "part" */
    part inventory[SIZE];    /* Array to hold SIZE "part"s */

    FILE * fp;
    char * filename = "structBin.bin";
    int i;

    fp = fopen(filename,"w");
    if(fp != NULL){
        for(i=0; i < SIZE; i++){
            /* Write the structures. */
            sprintf(board.name,"I/O card #%d", i);
            board.no = 127356 + i;
            board.price = 99.50 + i*3;
            fwrite(&board, sizeof(part), 1, fp);
        }
        fclose(fp);
        fp = fopen(filename,"r");
    }
    cont...
```
if(fp != NULL){  /* Read the structures. */
    fread(inventory, sizeof(part), SIZE, fp);
    fclose(fp);
    for(i=0; i < SIZE; i++){  /* Display the array of structures. */
        print_part(inventory + i);
        printf("\n");
    }
} else
    fprintf(stderr,"Unable to open file %s for read.\n", filename);
else
    fprintf(stderr,"Unable to open file %s for write.\n", filename);
return 0;
}

void print_part(const part * const p) /* Display function */
{  
    printf("Product: %s\n", p->name);
    printf("Part No.: %ld\n", p->no);
    printf("Unit price: %.2f\n", p->price);
    return;
}

/* OUTPUT: struct14.c

   Product: I/O card #0
   Part No.: 127356
   Unit price: 99.50

   Product: I/O card #1
   Part No.: 127357
   Unit price: 102.50

   Product: I/O card #2
   Part No.: 127358
   Unit price: 105.50

   Product: I/O card #3
   Part No.: 127359
   Unit price: 108.50

   Product: I/O card #4
   Part No.: 127360
   Unit price: 111.50

*/
FUNCTION POINTERS/QUICK SORT

- The C standard libraries provide many routines that we have not looked at.
- An example of a very useful standard library routine that uses some advanced C features is the quick-sort function, `qsort( )`.
- `qsort( )` will sort any array. This should seem a little surprising since `qsort( )` obviously didn’t know what would be in our array when `qsort( )` was written.
- Everything `qsort( )` needs to know about our array must be passed in as parameters.
- To do this it needs to be passed a very interesting parameter, a function pointer or the address of a function. Fortunately for us it very easy to do, the name of a function is the address of the function. It is this function that will make the comparisons between the items in the array while it is being sorted.
- The other parameters are similar to what we saw with binary I/O. The address of where the data starts, the number of data elements, and the size of each data element.
- With these pieces of information a generic `qsort( )` can sort any array.
Ex:

```
/*     FILE: qsort.c     */
/* qsort - the basics */
#include <stdio.h>
#define SIZE 5
int icomp(int *, int *);

int main( )
{
    int ar[SIZE];
    int i;
    for(i=0; i<SIZE; i++)
        ar[i] = SIZE - i;
    for(i=0; i<SIZE; i++)
        printf("ar[%d] = %d\n", i, ar[i]);
    qsort(ar, SIZE, sizeof(ar[0]), icomp);
    printf("\nSorted:\n");
    for(i=0; i<SIZE; i++)
        printf("ar[%d] = %d\n", i, ar[i]);
    return 0;
}

int icomp(int *i1, int *i2) /* function to compare integers */
    /* ... per qsort( )'s requirements. */
{
    if(*i1 == *i2)
        return 0;
    else if(*i1 < *i2)
        return -1;
    else
        return 1;
}
```

/*    OUTPUT: qsort.c    */

```
ar[0] = 5
ar[1] = 4
ar[2] = 3
ar[3] = 2
ar[4] = 1

Sorted:
ar[0] = 1
ar[1] = 2
ar[2] = 3
ar[3] = 4
ar[4] = 5
*/
```
Ex:

```c
/*     FILE: qsort2.c     */
/* qsort - the basics */
#include <stdio.h>
#define SIZE 5
int icomp(int *, int *);

int main( )
{
    int ar[SIZE];
    int i;
    for(i=0; i<SIZE; i++)
        ar[i] = SIZE - i;
    for(i=0; i<SIZE; i++)
        printf("ar[%d] = %d\n", i, ar[i]);
    qsort(ar,SIZE,sizeof(ar[0]),icomp);
    printf("\nSorted:\n");
    for(i=0; i<SIZE; i++)
        printf("ar[%d] = %d\n", i, ar[i]);
    return 0;
}

int icomp(int *i1, int *i2)  /* function to compare integers */
{                            /* ... per qsort( )'s requirements. */
    return *i1 - *i2;        /* The short version. */
}

/*    OUTPUT: qsort2.c    */
    ar[0] = 5
    ar[1] = 4
    ar[2] = 3
    ar[3] = 2
    ar[4] = 1
    Sorted:
    ar[0] = 1
    ar[1] = 2
    ar[2] = 3
    ar[3] = 4
    ar[4] = 5
*/
Ex:

```c
/* FILE: qsort3.c */
/* qsort - with a prototype from stdlib.h */

#include <stdio.h>
#include <stdlib.h>

#define SIZE 5
int icomp(void const * i1, void const * i2);

int main( )
{
    int ar[SIZE];
    int i;
    for(i=0; i<SIZE; i++)
        ar[i] = SIZE - i;
    for(i=0; i<SIZE; i++)
        printf("ar[%d] = %d\n", i, ar[i]);
    qsort(ar,SIZE,sizeof(ar[0]),icomp);
    printf("\nSorted:\n");
    for(i=0; i<SIZE; i++)
        printf("ar[%d] = %d\n", i, ar[i]);
    return 0;
}

int icomp(void const * i1, void const * i2) /*compare integers per */
{                                           /* ... qsort( )'s prototype */
    return *(int *)i1 - *(int *)i2;          /* cast to int* and dereference */
}
```

/* OUTPUT: qsort3.c

    ar[0] = 5
    ar[1] = 4
    ar[2] = 3
    ar[3] = 2
    ar[4] = 1

    Sorted:
    ar[0] = 1
    ar[1] = 2
    ar[2] = 3
    ar[3] = 4
    ar[4] = 5
*/
Ex:
/*     FILE: qsort4.c     */
/* sort some struct data */
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
struct part{
  char name[128];
  long no;
  double price;
};
int price_compare(void const * p1, void const * p2);
int no_compare(void const * p1, void const * p2);
void print_part(struct part* p);

int main( )
{
  struct part board; /* One "part" */
  struct part inventory[SIZE]; /* Array to hold SIZE "part"s */
  int i;
  for(i=0; i < SIZE; i++) /* Load the array of structures. */
    sprintf(board.name,"I/O card #%d", i);
    board.no = 127999 - rand( )%1000;
    board.price = 199.99 - rand( )%100;
  inventory[i] = board;
  for(i=0; i < SIZE; i++) /* Display the array of structures. */
    print_part(inventory + i);
  printf("
Sorted by price:
------------------
");
  qsort(inventory, SIZE, sizeof(struct part),price_compare);
  for(i=0; i < SIZE; i++) /* Display the array of structures. */
    print_part(inventory + i);
  printf("
Sorted by part no:
------------------
");
  qsort(inventory, SIZE, sizeof(struct part),no_compare);
  for(i=0; i < SIZE; i++) /* Display the array of structures. */
    print_part(inventory + i);
  return 0;
}
```c
int no_compare(void const * p1, void const * p2)
{
    return ((struct part*)p1)->no - ((struct part*)p2)->no;
}

int price_compare(void const * p1, void const * p2)
{
    struct part * part1 = (struct part *)p1;
    struct part * part2 = (struct part *)p2;
    if( part1->price == part2->price)
        return 0;
    if( part1->price < part2->price)
        return -1;
    else
        return 1;
}

void print_part(struct part* p)   /* -> operator simplifies access thru */
{                                 /* ... a pointer.                      */
    printf("Product: %s\n", p->name);
    printf("Part No.: %ld\n", p->no);
    printf("Unit price: %.2f\n", p->price);
    return;
}

/* OUTPUT: qsort4.c

    Product: I/O card #0
    Part No.: 127958
    Unit price: 132.99

    Product: I/O card #1
    Part No.: 127665
    Unit price: 199.99

    Product: I/O card #2
    Part No.: 127830
    Unit price: 175.99

    Product: I/O card #3
    Part No.: 127521
    Unit price: 141.99

    Product: I/O card #4
    Part No.: 127037
    Unit price: 135.99

cont…
```
Sorted by price:
--------------------
Product: I/O card #0
Part No.: 127958
Unit price: 132.99

Product: I/O card #4
Part No.: 127037
Unit price: 135.99

Product: I/O card #3
Part No.: 127521
Unit price: 141.99

Product: I/O card #2
Part No.: 127830
Unit price: 175.99

Product: I/O card #1
Part No.: 127665
Unit price: 199.99

Sorted by part no:
------------------
Product: I/O card #4
Part No.: 127037
Unit price: 135.99

Product: I/O card #3
Part No.: 127521
Unit price: 141.99

Product: I/O card #1
Part No.: 127665
Unit price: 199.99

Product: I/O card #2
Part No.: 127830
Unit price: 175.99

Product: I/O card #0
Part No.: 127958
Unit price: 132.99

*/
BIT OPERATORS

- C has a set of operators that can be used to perform bit-level operations.
- There are a pair of shift operators, and bitwise OR, AND, XOR, and NOT.
- All the operators are applied to each bit in the entire bit pattern. That is, all bits get shifted, all bits get ANDed, etc.

Ex:

```c
/*     FILE: bitop.c     */
/* Exercises several C bit operators */

#include <stdio.h>

void setOneBit(int* ptr, int bit);
void setBit(int* ptr, int bit);
void clearBit(int* ptr, int bit);

int main( )
{
  int x;

  setOneBit(&x, 3);
  printf("x = %8.8X\n", x);

  clearBit(&x, 3);
  printf("x = %8.8X\n", x);

  x = 3;
  printf("\nx = %8.8X\n", x);
  setBit(&x, 3);
  printf("x = %8.8X\n", x);

  clearBit(&x, 3);
  printf("x = %8.8X\n", x);

  return 0;
}

void setOneBit(int* ptr, int bit) /* sets the specified bit on, */
{                              /* ... all others will be off. */
  *ptr = 1 << bit;            /* */
}

void setBit(int* ptr, int bit) /* sets specified bit on and */
{                                /* leaves all others as-is. */
  *ptr = (*ptr) | (1 << bit);
}

void clearBit(int* ptr, int bit) /* turns specified bit off. */
{                                /* */
  *ptr = (*ptr) & (~ (1 << bit));
}

cont…
/* OUTPUT: bitop.c

    x = 00000008
    x = 00000000

    x = 00000003
    x = 0000000B
    x = 00000003

*/
Ex:

```c
/*     FILE: bitop2.c     */
/* Exercises several C bit operators */
/* Don't waste the return value. */

#include <stdio.h>

int setOneBit(int* ptr, int bit);
int setBit(int* ptr, int bit);
int clearBit(int* ptr, int bit);

int main( )
{
    int x;
    printf("x = %8.8X\n", setOneBit(&x, 3));
    printf("x = %8.8X\n", clearBit(&x, 3));
    printf("\nx = %8.8X\n", x = 3);
    printf("x = %8.8X\n", setBit(&x, 3));
    printf("x = %8.8X\n", clearBit(&x, 3));
    return 0;
}

int setOneBit(int* ptr, int bit) /* sets the specified bit on, */
{ /* ... all others will be off. */
    *ptr = 1 << bit;
    return *ptr;
}

int setBit(int* ptr, int bit) /* sets specified bit on and */
{ /* ... leaves all others as-is. */
    *ptr = (*ptr) | (1 << bit);
    return *ptr;
}

int clearBit(int* ptr, int bit) /* turns specified bit off. */
{ /*
    *ptr = (*ptr)&(~(1 << bit));
    return *ptr;
}

/*    OUTPUT: bitop2.c

    x = 00000008
    x = 00000000
    x = 00000003
    x = 0000000B
    x = 00000003

    */
```
PREPROCESSOR – MORE FEATURES

- The preprocessor has many additional uses besides the basic #include and #define mechanism.
- More elaborate macro substitutions can be performed with the #define mechanism.
- Conditional preprocessor statements can allow code to be conditionally compiled/included in the source code.

NOTE: The preprocessor is sophisticated but is not as sophisticated as the C compiler. They perform two different functions. The preprocessor massages your source code prior to compilation. The compiler translates C code into machine instructions. Being aware of this can keep you from misusing/overusing the preprocessor. Recognize that the preprocessor has limitations and is not really C literate.
Ex:

/* FILE: preprocessor.c */

/* Preprocessor macros allow "function-like" substitution.
Inlining allows "function-like" definitions without the function call overhead. */

#include <stdio.h>
#define PLUS5(X) X + 5
inline int plus5(int x){return x + 5;}

int main( )
{
  int x;
  char *s = "Jim Polzin";
  x = 5;
  printf("x = %d x + 5 = %d\n", x, plus5(x));
  printf("x = %d x + 5 = %d\n", x, PLUS5(x));
  // printf("s = %s, s + 5 = %s\n", s, plus5(s));
  printf("s = %s, s + 5 = %s\n", s, PLUS5(s));
  return 0;
}

/* OUTPUT: preprocessor.c
  x = 5 x + 5 = 10
  x = 5 x + 5 = 10
  s = Jim Polzin, s + 5 = olzin
*/
Ex:

```c
/*     FILE: preprocessor2.c     */

/* Preprocessor macros allow "function-like" substitution.
   - No type checking by the preprocessor
   Inlining allows "function-like" definitions without the function call overhead.
   - Compiler handles inline functions and does its usual type-checking */

#include <stdio.h>
#define PLUS5(X)   X + 5
inline int plus5(int x){return x + 5;}

int main( )
{  
  int x;
  char *s = "Jim Polzin";
  x = 5;
  printf("x = %d  x + 5 = %d\n", x, plus5(x));
  printf("x = %d  x + 5 = %d\n", x, PLUS5(x));
  printf("s = %s, s + 5 = %s\n", s, plus5(s));
  printf("s = %s, s + 5 = %s\n", s, PLUS5(s));
  return 0;
}

/*     OUTPUT: preprocessor2.c     */

preprocessor2.c: In function `main':
preprocessor2.c:31: warning: passing arg 1 of `plus5' makes integer from pointer without a cast

*/
```
Ex:
/*     FILE: preprocessor3.c     */
/* Preprocessor allows "conditional compilation" */
#include <stdio.h>
#define TESTFUNCTION
#define PLUS5(X)   X + 5
inline int plus5(int x){return x  + 5;}

int main( )
{
    int x;
    char *s = "Jim Polzin";
    x = 5;
    printf("x = %d  x + 5 = %d\n", x, plus5(x));
    printf("x = %d  x + 5 = %d\n", x, PLUS5(x));
    #ifdef TESTFUNCTION
    printf("s = %s, s + 5 = %s\n", s, plus5(s));
    #endif
    printf("s = %s, s + 5 = %s\n", s, PLUS5(s));
    return 0;
}

/* OUTPUT: preprocessor3.c
  preprocessor3.c: In function `main':
  preprocessor3.c:24: warning: passing arg 1 of `plus5' makes integer from
   pointer without a cast
  */
Ex:

```c
/*     FILE: preprocessor4.c     */
/* Preprocessor allows "conditional compilation" */
#include <stdio.h>
#undef TESTFUNCTION
#define PLUS5(X)   X + 5
inline int plus5(int x){return x + 5;}

int main( )
{
  int x;
  char *s = "Jim Polzin";
  x = 5;
  printf("x = %d  x + 5 = %d\n", x, plus5(x));
  printf("x = %d  x + 5 = %d\n", x, PLUS5(x));

#ifdef TESTFUNCTION
  printf("s = %s, s + 5 = %s\n", s, plus5(s));
#else
  printf("s = %s, s + 5 = %s\n", s, PLUS5(s));
#endif
  return 0;
}

/*     OUTPUT: preprocessor4.c     */
  x = 5  x + 5 = 10
  x = 5  x + 5 = 10
    s = Jim Polzin, s + 5 = olzin

  */
```
Ex:

```c
/*     FILE: preprocessor5.c     */
/* Preprocessor allows "conditional compilation"     
    This can be used to "remove" blocks of code from the compile process. */

#include <stdio.h>
#define PLUS5(X)   X + 5
inline int plus5(int x){return x + 5;}

int main( )
{
    int x;
    char *s = "Jim Polzin";
    x = 5;
    printf("x = %d  x + 5 = %d\n", x, plus5(x));
    printf("x = %d  x + 5 = %d\n", x, PLUS5(x));

#ifdef UNDEF
    /* Existing comments can make it difficult to comment-out a block of code. The C preprocessor isn't sensitive to C comments */
    printf("s = %s, s + 5 = %s\n", s, plus5(s));
#else
    printf("s = %s, s + 5 = %s\n", s, PLUS5(s));
#endif

    return 0;
}

OUTPUT: preprocessor5.c
x = 5  x + 5 = 10
x = 5  x + 5 = 10
s = Jim Polzin, s + 5 = olzin
*/
```
Ex:

```c
/*     FILE: preprocessor6.c     */
/* Preprocessor allows "conditional compilation"
   This can be used to insert debugging code that
   can be turned on/off or included/excluded in the
   program as needed in the development process. */

#include <stdio.h>
#define DEBUG 1
#define PLUS5(X)   X + 5
inline int plus5(int x){return x + 5;}

int main( )
{
    int x;
    char *s = "Jim Polzin";
    x = 5;
    #ifdef DEBUG
    #if DEBUG == 1
        printf("Program has arrived at source line: %d\n", __LINE__);  
    #endif
    #endif
    printf("x = %d  x + 5 = %d\n", x, plus5(x));  
    printf("x = %d  x + 5 = %d\n", x, PLUS5(x));
    #ifdef UNDEF
    printf("s = %s, s + 5 = %s\n", s, plus5(s));
    #endif
    printf("s = %s, s + 5 = %s\n", s, PLUS5(s));
    return 0;
}
```

/*    OUTPUT: preprocessor6.c
   Program has arrived at source line: 26
   x = 5  x + 5 = 10
   x = 5  x + 5 = 10
   s = Jim Polzin, s + 5 = olzin
*/
Ex:

```c
/*     FILE: preprocessor7.c     */

/* Preprocessor allows "conditional compilation"
This can be used to insert debugging code that can be turned on/off or included/excluded in the program as needed in the development process.     */

#include <stdio.h>
#define DEBUG 1
#define PLUS5(X)   X + 5
inline int plus5(int x){return x + 5;}

int main( )
{
    int x;
    char *s = "Jim Polzin";
    x = 5;
    #ifdef DEBUG
        #if DEBUG >= 1
            printf("Program has arrived at source line: %d\n", __LINE__);        
        endif
        #if DEBUG >= 2
            printf("Tracking variable x: x = %d\n", x);
        endif
    endif
    printf("x = %d  x + 5 = %d\n", x, plus5(x));
    #ifdef DEBUG
        #if DEBUG >= 1
            printf("Program has arrived at source line: %d\n", __LINE__);        
        endif
        #if DEBUG >= 2
            printf("Tracking variable x: x = %d\n", x);
        endif
    endif
    printf("x = %d  x + 5 = %d\n", x, PLUS5(x));
    #ifdef UNDEF
        #ifdef DEBUG
            #if DEBUG >= 1
                printf("Program has arrived at source line: %d\n", __LINE__);            
            endif
            #if DEBUG >= 2
                printf("Tracking variable x: x = %d\n", x);
            endif
        endif
        printf("s = %s, s + 5 = %s\n", s, plus5(s));
    endif
    #ifdef DEBUG
        #if DEBUG >= 1
            printf("Program has arrived at source line: %d\n", __LINE__);        
        endif
        #if DEBUG >= 2
            printf("Tracking variable x: x = %d\n", x);
        endif
    endif
    printf("s = %s, s + 5 = %s\n", s, PLUS5(s));
    return 0;
}
```
cont…
/* OUTPUT: preprocessor7.c

Program has arrived at source line: 26
x = 5  x + 5 = 10
Program has arrived at source line: 35
x = 5  x + 5 = 10
Program has arrived at source line: 56
s = Jim Polzin, s + 5 = olzin

*/
Ex:
/*     FILE: preprocessor8.c     */
/* Preprocessor allows "conditional compilation"
This can be used to insert debugging code that can be turned on/off or included/excluded in the program as needed in the development process. */

#include <stdio.h>
#define DEBUG 2
#define PLUS5(X)   X + 5
inline int plus5(int x){return x + 5;}

int main( )
{
  int x;
  char *s = "Jim Polzin";
  x = 5;
  
  #ifdef DEBUG
  #if DEBUG >= 1
    printf("Program has arrived at source line: %d\n", __LINE__);  
  #endif
  #if DEBUG >= 2
    printf("Tracking variable x: x = %d\n", x);  
  #endif
  #endif
  printf("x = %d  x + 5 = %d\n", x, plus5(x));
  #ifdef DEBUG
  #if DEBUG >= 1
    printf("Program has arrived at source line: %d\n", __LINE__);  
  #endif
  #if DEBUG >= 2
    printf("Tracking variable x: x = %d\n", x);  
  #endif
  #endif
  printf("x = %d  x + 5 = %d\n", x, PLUS5(x));
  #ifdef UNDEF
  #ifdef DEBUG
  #if DEBUG >= 1
    printf("Program has arrived at source line: %d\n", __LINE__);  
  #endif
  #if DEBUG >= 2
    printf("Tracking variable x: x = %d\n", x);  
  #endif
  #endif
  printf("s = %s, s + 5 = %s\n", s, plus5(s));
  #endif
  #ifdef DEBUG
  #if DEBUG >= 1
    printf("Program has arrived at source line: %d\n", __LINE__);  
  #endif
  #if DEBUG >= 2
    printf("Tracking variable x: x = %d\n", x);  
  #endif
  #endif
  printf("s = %s, s + 5 = %s\n", s, PLUS5(s));
  
  return 0;
}

cont...
/* OUTPUT: preprocessor8.c

Program has arrived at source line: 26
Tracking variable x: x = 5
x = 5  x + 5 = 10
Program has arrived at source line: 35
Tracking variable x: x = 5
x = 5  x + 5 = 10
Program has arrived at source line: 56
Tracking variable x: x = 5
s = Jim Polzin, s + 5 = olzin

*/
Ex:

```c
/* FILE: preprocessor9.c */
/* Preprocessor allows "conditional compilation"
This can be used to insert debugging code that can be turned on/off or included/excluded in the program as needed in the development process. */

#include <stdio.h>
#define DEBUG 0
#define PLUS5(X) X + 5

inline int plus5(int x) { return x + 5; }

int main() {
    int x;
    char *s = "Jim Polzin";
    x = 5;
    #ifdef DEBUG
        #if DEBUG >= 1
            printf("Program has arrived at source line: %d\n", __LINE__);
        #endif
        #if DEBUG >= 2
            printf("Tracking variable x: x = %d\n", x);
        #endif
    #endif
    printf("x = %d  x + 5 = %d\n", x, plus5(x));
    #ifdef DEBUG
        #if DEBUG >= 1
            printf("Program has arrived at source line: %d\n", __LINE__);
        #endif
        #if DEBUG >= 2
            printf("Tracking variable x: x = %d\n", x);
        #endif
    #endif
    printf("x = %d  x + 5 = %d\n", x, PLUS5(x));
    #ifdef UNDEF
    #ifdef DEBUG
        #if DEBUG >= 1
            printf("Program has arrived at source line: %d\n", __LINE__);
        #endif
        #if DEBUG >= 2
            printf("Tracking variable x: x = %d\n", x);
        #endif
    #endif
    printf("s = %s, s + 5 = %s\n", s, plus5(s));
    #endif
    #ifdef DEBUG
        #if DEBUG >= 1
            printf("Program has arrived at source line: %d\n", __LINE__);
        #endif
        #if DEBUG >= 2
            printf("Tracking variable x: x = %d\n", x);
        #endif
    #endif
    printf("s = %s, s + 5 = %s\n", s, PLUS5(s));
    return 0;
}
```

cont…
/* OUTPUT: preprocessor9.c
   
   x = 5  x + 5 = 10
   x = 5  x + 5 = 10
   s = Jim Polzin, s + 5 = olzin

*/
Ex:

/*     FILE: preprocessor10.c     */
/* Preprocessor allows "conditional compilation"

    The date and time of the production of the executable can be consistently stamped by
    the preprocessor. */

#include <stdio.h>
#define AUTHOR "Jim Polzin"
#define BUILD_DATE __DATE__ " " __TIME__

int main( )
{
    printf("Program by: %s\n", AUTHOR);
    printf("Build date: %s\n", BUILD_DATE);
    return 0;
}

/* OUTPUT: preprocessor10.c

    Program by: Jim Polzin
    Build date: Dec 2 2004 14:38:48

*/
BUILDING LIBRARIES

• Libraries are collections of existing code.

• Since code reuse has significant implications in regards to programmer productivity and code reliability; having, creating and using libraries is an important part of producing reliable code quickly.

• A compiler will allow us to compile files into object modules. These object modules can then be placed into a library file which can then be provided to the linker. The linker will search these library files for any unresolved function calls while it attempts to produce a final executable file.

• Since the compiler doesn’t see any of the code in the library files, included .h files are the mechanism for providing the compiler with the information it needs to accurately compile our code that will eventually utilize the code found in the libraries.
Ex:

/*     FILE: stringLibTest.c     */

/* A program that is dependent on other functions
   in other files.

   This file needs the code from stringLib1.c
   and stringLib2.c

   The intent is for those files to be compiled
   and placed in a library using the ar - archive
   utility that the linker will search.        */

#include "stringLib.h"
#include <string.h>

int main( )
{
    int i;
    char name[81];
    myStrcpy(name,"Jim");
    strcat(name," Polzin");
    printf("You created: ");
    myPuts(name);
    return 0;
}

/*     OUTPUT: stringLibTest.c     */
    You created: Jim Polzin

*/
Ex:

```c
/*     FILE: stringLib.h     */
/* Prototypes for the stringLib functions */

void myPuts(char [ ]);  
void myStrcpy(char [ ], char[ ]); 
```

Ex:

```c
/*     FILE: stringLib1.c     */
/* StringLib version of strcpy( ) */

void myStrcpy(char dest[ ], char src[ ]) 
{ 
    int i;  
    for(i=0; src[i] != '\0'; i++)  
        dest[i] = src[i];  
    dest[i] = '\0'; 
    return; 
}
```

Ex:

```c
/*     FILE: stringLib2.c     */
/* StringLib version of puts( ) */

#include <stdio.h> 

void myPuts(char s[ ]) 
{ 
    int i;  
    for(i=0; s[i] != '\0'; i++)  
        putchar(s[i]);  
    putchar('\n'); 
    return; 
}
```
Building Libraries

Ex:

```
H:\c\examples>dir
06/28/2004 10:56a 107 stringLib.h
06/28/2004 10:56a 197 stringLib1.c
06/28/2004 10:57a 194 stringLib2.c
06/28/2004 10:59a 247 stringLibTest.c
4 File(s)  745 bytes
2 Dir(s)  70,973,628,416 bytes free
```

Compiling only, using the –c option:

```
H:\c\examples>gcc -c stringLibTest.c
H:\c\examples>dir
06/28/2004 11:19a <DIR>   .
06/28/2004 11:19a <DIR>   ..
06/28/2004 10:56a 107 stringLib.h
06/28/2004 10:56a 197 stringLib1.c
06/28/2004 10:57a 194 stringLib2.c
06/28/2004 10:59a 247 stringLibTest.c
06/28/2004 11:24a  676 stringLibTest.o
```

```
H:\c\examples>gcc -c stringLib1.c
H:\c\examples>gcc -c stringLib2.c
```

```
H:\c\examples>dir
06/28/2004 11:19a <DIR>   .
06/28/2004 11:19a <DIR>   ..
06/28/2004 10:56a 107 stringLib.h
06/28/2004 10:56a 197 stringLib1.c
06/28/2004 11:24a  398 stringLib1.o
06/28/2004 10:57a 194 stringLib2.c
06/28/2004 11:24a  442 stringLib2.o
06/28/2004 10:59a 247 stringLibTest.c
06/28/2004 11:24a  676 stringLibTest.o
```

Building the .a library file:

```
H:\c\examples>rm stringLibTest.o
H:\c\examples>ar -r stringLib.a stringLib1.o stringLib2.o
```

```
H:\c\examples>dir
06/28/2004 11:19a <DIR>   .
06/28/2004 11:19a <DIR>   ..
06/28/2004 11:27a  1,058 stringLib.a
06/28/2004 10:56a 107 stringLib.h
06/28/2004 10:56a 197 stringLib1.c
06/28/2004 11:24a  398 stringLib1.o
06/28/2004 10:57a 194 stringLib2.c
06/28/2004 11:24a  442 stringLib2.o
06/28/2004 10:59a 247 stringLibTest.c
7 File(s)  2,643 bytes
2 Dir(s)  70,973,624,320 bytes free
```

cont...
Compiling using only the .a library file:

```
H:\c\examples>gcc stringLibTest.c stringLib.a
H:\c\examples>dir
06/28/2004 11:19a <DIR> .
06/28/2004 11:19a <DIR> ..
06/28/2004 11:29a 24,447 a.exe
06/28/2004 11:27a 1,058 stringLib.a
06/28/2004 10:56a 107 stringLib.h
06/28/2004 10:56a 197 stringLib1.c
06/28/2004 11:24a 398 stringLib1.o
06/28/2004 10:57a 194 stringLib2.c
06/28/2004 11:24a 442 stringLib2.o
06/28/2004 10:59a 247 stringLibTest.c
  8 File(s) 27,090 bytes
  2 Dir(s) 70,973,485,056 bytes free
```

```
H:\c\examples>a
You created: Jim Polzin
```

Compiling using only the .a library file, .o files removed:

```
H:\c\examples>del *.o
H:\c\examples>del stringLib1.c
H:\c\examples>del stringLib2.c
H:\c\examples>gcc stringLibTest.c stringLib.a
H:\c\examples>dir
06/28/2004 11:19a <DIR> .
06/28/2004 11:19a <DIR> ..
06/28/2004 11:30a 24,447 a.exe
06/28/2004 11:27a 1,058 stringLib.a
06/28/2004 10:56a 107 stringLib.h
06/28/2004 10:59a 247 stringLibTest.c
  4 File(s) 25,859 bytes
  2 Dir(s) 70,973,403,136 bytes free
```

```
H:\c\examples>a
You created: Jim Polzin
```
USING SEARCH PATHS

- Collections of library files and include files create logistical complications for organizing and accessing these files.
- Search paths are a mechanism for directing programs and even the operating system where to look for files.
- Most compilers and IDEs allow the programmer to set paths to be searched during the building of an application.
- One tool that addresses these issues is the make utility. Make files can contain search path information, lists of include files to use, lists of libraries to find and also to build.
- The following examples illustrate some basic path communication techniques for the gcc compiler
Ex: (Based on the “Building Libraries” example.)

Files .h and .a moved to another directory:

```
H:\c>dir
06/02/2004  12:32p     <DIR>          .
06/02/2004  12:32p     <DIR>          ..
06/28/2004  11:27a     1,058 stringLib.a
06/28/2004  10:56a     107 stringLib.h
06/28/2004  11:19a     <DIR>          examples
  2 File(s)     1,165 bytes
  3 Dir(s)     70,878,576,640 bytes free
H:\c>rename stringLib.a libstringLib.a
H:\c>dir
06/02/2004  12:32p     <DIR>          .
06/02/2004  12:32p     <DIR>          ..
06/28/2004  11:27a     1,058 libstringLib.a
06/28/2004  10:56a     107 stringLib.h
06/28/2004  11:19a     <DIR>          examples
  2 File(s)     1,165 bytes
  3 Dir(s)     70,878,552,064 bytes free
```

Compile failure:

```
H:\c\examples>gcc -c stringLibTest.c
stringLibTest.c:3:23: stringLib.h: No such file or directory
```

Compile succeeds with –I flag:

```
H:\c\examples>gcc -c -I:H:\c stringLibTest.c
H:\c\examples>gcc -c -I:.\. stringLibTest.c
H:\c\examples>dir
06/28/2004  11:19a     <DIR>          .
06/28/2004  11:19a     <DIR>          ..
06/28/2004  10:59a     247 stringLibTest.c
06/28/2004  02:44p     676 stringLibTest.o
   2 File(s)     923 bytes
   2 Dir(s)     70,878,068,736 bytes free
```

Link fails:

```
H:\c\examples>gcc -I:.\. stringLibTest.c
C:\WINNT\TEMP/ccArcaaa.o(.text+0x48):stringLibTest.c: undefined reference to `myStrcpy'
C:\WINNT\TEMP/ccArcaaa.o(.text+0x7b):stringLibTest.c: undefined reference to `myPuts'
```

Link succeeds with explicit path information:

```
H:\c\examples>gcc -I:.\. stringLibTest.c ..\libstringLib.a
```

Link succeeds with –L option:

```
H:\c\examples>gcc -I:.\. -L:.\. stringLibTest.c -lstringLib
H:\c\examples>dir
06/28/2004  11:19a     <DIR>          .
06/28/2004  11:19a     <DIR>          ..
06/28/2004  02:46p     24,447 a.exe
06/28/2004  10:59a     247 stringLibTest.c
06/28/2004  02:44p     676 stringLibTest.o
   3 File(s)     25,370 bytes
   2 Dir(s)     70,877,958,144 bytes free
```
WRITING TEXT WITH OPENGL/GLUT

- The following examples write some text around a grid that is sized to fit in a subportion of a window.
- The grid resizes with the window and the text follows.

Ex:

```c
/*     FILE: ./OpenGL/Grid2.c     */
/* Program that draws a "Karel-like" grid
   With labels.                            */

#include <stdlib.h>
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit( ); /* <- Functions written by the programmer to */
                 /* ... produce the graphics they desire. */
void display( ); /* ... myinit( ) is called once. Display( ) */
                 /* ... and reshape( ) are called as needed. */
void reshape(int, int); /* ... border( ) and drawGrid( ) are called by */
void border( ); /* ... display. */
void drawGrid( ); /* ... */

/* Writes each character in a string */
void printString(void *font, char *str)
{
    int i, len;
    len=strlen(str);
    for(i=0;i<len;i++)
        glutBitmapCharacter(font,*str++);
}

/* Function to initialize OpenGL parameters and
   prepare for drawing as the programmer sees fit. */
void myinit( )
{
    glClearColor(1.,1.,1.,1.); /* white background */
    glColor3f(0.,0.,0.); /* black foreground */
    glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin shifted */
    /* up and to the right 25 pixels. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-25.,(GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
}

cont...
```
/* Function registered with OpenGL for producing the graphics */
void display()
{
    char buf[128];
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(3.0); /* Width line-width for boundary */
    glBegin(GL_LINES);
    border();
    glEnd();
    glLineWidth(1.0); /* Small line-width for St. & Ave. */
    glBegin(GL_LINES);
    drawGrid();
    glEnd();
    /* Label the Graphic in the window */
    strcpy(buf,"Karel's World");
    glRasterPos2i(2,scrHeight-50);
    printString(GLUT_BITMAP_HELVETICA_12,buf);
    /* Label the Avenues below the grid */
    strcpy(buf,"Avenues");
    glRasterPos2i(scrWidth/4,-25);
    printString(GLUT_BITMAP_HELVETICA_12,buf);
    /* Label the Streets along the side */
    strcpy(buf,"Streets");
    glRasterPos2i(2,scrHeight/2+25);
    printString(GLUT_BITMAP_HELVETICA_12,buf);
    glFlush();
}

/* Draw the world boundaries */
/* ... Called by display( ) to allow display( ) to be ... more succinct. */
void border()
{
    int i;
    int halfHeight = scrHeight/2;
    int halfWidth = scrWidth/2;
    glVertex2i(0,0);
    glVertex2i(0,halfHeight);
    glVertex2i(0,0);
    glVertex2i(halfWidth,0);
}

cont...
/* Draw the streets and avenues */
/* ... Called by display() to allow display() to be more succinct. */
void drawGrid()
{
    int i;
    int halfHeight = scrHeight/2;
    int halfWidth = scrWidth/2;

    for(i=25; i < halfHeight; i+=25){
        glVertex2i(0,i);
        glVertex2i(halfWidth,i);
    }
    for(i=25; i < halfWidth; i+=25){
        glVertex2i(i,0);
        glVertex2i(i,halfHeight);
    }
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-25.,(GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
    display();
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv); /* These 1st four function calls are OpenGL/GLUT */
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); /* ... preparatory calls */
    glutInitWindowSize(scrWidth,scrHeight); /* ... so that the graphics can be displayed */
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display); /* <- Registers the programmers drawing code */
    myinit(); /* This is a local function to establish the current state desired by the programmer. */
    glutReshapeFunc(reshape); /* <- Registers the programmers drawing code */
    glutMainLoop(); /* Starts the event-loop for the graphics environment. */
    return 0;
}
Karel's World

Streets

Avenues
Ex:

```c
/*     FILE: ./OpenGL/Grid3.c     */
/* Program that draws a "Karel-like" grid
With labels.               */

#include <stdlib.h>
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit( ); /* <- Functions written by the programmer to */
void display( ); /* ... produce the graphics they desire. */
void reshape(int, int); /* ... myinit( ) is called once. Display( ) */
/* ... and reshape( ) are called as needed. */

void border( ); /* ... border( ) and drawGrid( ) are called by */
void drawGrid( ); /* ... display. */

/* Writes each character in a string */
void printString(void *font, char *str)
{
    int i, len;
    len=strlen(str);
    for(i=0;i<len;i++)
        glutBitmapCharacter(font,*str++);
}

/* Function to initialize OpenGL parameters and
prepare for drawing as the programmer sees fit. */
void myinit( )
{
    glClearColor(1.,1.,1.,1.); /* white background */
    glColor3f(0.,0.,0.); /* black foreground */
    glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin shifted */
    /* up and to the right 25 pixels. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-25.,(GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
}
```

cont…
/* Function registered with OpenGL for producing the graphics */
void display( )
{
    char buf[128];
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(3.0);   /* Width line-width for boundary */
    glBegin(GL_LINES);
    border( );
    glEnd( );
    glEnd( );
    glLineWidth(1.0);   /* Small line-width for St. & Ave. */
    glBegin(GL_LINES);
    drawGrid( );
    glEnd( );

    /* Label the Graphic in the window */
    strcpy(buf,"Karel's World");
    glRasterPos2i(2,scrHeight-40);
    printString(GLUT_BITMAP_HELVETICA_12,buf);
    /* Label the Avenues below the grid */
    strcpy(buf,"Avenues");
    glRasterPos2i(scrWidth/4,-25);
    printString(GLUT_BITMAP_HELVETICA_12,buf);
    /* Label the Streets along the side */
    strcpy(buf,"Streets");
    glRasterPos2i(2,scrHeight/2+25);
    printString(GLUT_BITMAP_HELVETICA_12,buf);
    glFlush( );
}

/* Draw the world boundaries */
/* ... Called by display( ) to allow display( ) to be 
... more succinct. */
void border( )
{
    int i;
    glVertex2i(0,0);
    glVertex2i(0,scrHeight - 50);
    glVertex2i(0,0);
    glVertex2i(scrWidth - 50,0);
}

/* Draw the streets and avenues */
/* ... Called by display( ) to allow display( ) to be 
... more succinct. */
void drawGrid( )
{
    int i;
    for(i=25; i < scrHeight - 50; i+=25){
        glVertex2i(0,i);
        glVertex2i(scrWidth - 50,i);
    }
    for(i=25; i < scrWidth - 50; i+=25){
        glVertex2i(i,0);
        glVertex2i(i,scrHeight - 50);
    }
    cont…
/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
  scrHeight = nescrHeight;
  scrWidth = nescrWidth;
  glViewport(0, 0, scrWidth, scrHeight);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-25., (GLdouble)scrWidth-25,-25., (GLdouble)scrHeight-25);
  glMatrixMode(GL_MODELVIEW);
  display();
}

int main(int argc, char** argv)
{
  glutInit(&argc, argv);                          /* These 1st four function   */
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);  /*  ... calls are OpenGL/GLUT */
  glutInitWindowSize(scrWidth, scrHeight);        /*  ... preparatory calls     */
  glutCreateWindow(argv[0]);

  glutDisplayFunc(display);    /* <- Registers the programmers drawing code  */
  /* ... so that the graphics can be displayed */
  /* ... in the window. */

  myinit();             /* This is a local function to establish the */
  /* ... current state desired by the programmer. */

  glutReshapeFunc(reshape);     /* <- Registers the programmers drawing code  */
  /* ... with OpenGL/GLUT so that if window is */
  /* ... reshaped, the graphics can be redrawn. */

  glutMainLoop( );       /* Starts the event-loop for the graphics environment. */

  return 0;
}

cont...
Ex:

```c
/*     FILE: ./OpenGL/Grid4.c     */
/* Program that draws a "Karel-like" grid 
With streets and avenues labeled and numbered. */

#include <stdlib.h>
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit( ); /* <- Functions written by the programmer to */
void display( ); /* ... produce the graphics they desire. */
void reshape(int, int); /* ... myinit( ) is called once. Display( ) */
/* ... and reshape( ) are called as needed. */

void border( ); /* ... border( ) and drawGrid( ) are called by */
void drawGrid( ); /* ... display. */

/* Labels each of the streets and avenues */
void numberStreetsAvenues(void *font, int pts, int st, int ave, int shift)
{
    int i, len;
    char buf[128];
    for(i=0;i<ave;i++){
        sprintf(buf,"%d",i+1);
        glRasterPos2i((i+1)*shift,-(pts+3));
        for(len=0;len<strlen(buf);len++)
            glutBitmapCharacter(font,buf[len]);
    }
    for(i=0;i<st;i++){
        sprintf(buf,"%d",i+1);
        glRasterPos2i(-(pts+3),(i+1)*shift);
        for(len=0;len<strlen(buf);len++)
            glutBitmapCharacter(font,buf[len]);
    }
}

/* Writes a string vertically from (x,y) down */
void verticalPrintString(void *font, char *str, int pts, int x, int y)
{
    int i, len;
    len=strlen(str);
    for(i=0;i<len;i++){
        glRasterPos2i(x,y-i*(pts+3));
        glutBitmapCharacter(font,*str++);
    }
}

/* Writes each character in a string */
void printString(void *font,char *str)
{
    int i, len;
    len=strlen(str);
    for(i=0;i<len;i++)
        glutBitmapCharacter(font,*str++);
}
```

cont…

Introduction to C Programming

Page: 246

Copyright © 2015 by James J. Polzin All Rights Reserved
Printed: 6/13/15
/* Function to initialize OpenGL parameters and  
   prepare for drawing. */
void myinit()
{
    glClearColor(1.,1.,1.,1.);    /* white background */
glColor3f(0.,0.,0.);           /* black foreground */
glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin shifted 
   up and to the right 25 pixels. */
glViewport(0,0,scrWidth,scrHeight);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-25.,(GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
glMatrixMode(GL_MODELVIEW);
}

/* Function registered with OpenGL for producing display */
void display()
{
    char buf[128];
    glClear(GL_COLOR_BUFFER_BIT);
glLineWidth(3.0);   /* Width line-width for boundary */
glBegin(GL_LINES);
border();
glEnd();
glLineWidth(1.0);   /* Small line-width for St. & Ave. */
glBegin(GL_LINES);
drawGrid();
glEnd();
strcpy(buf,"Karel's World");
glRasterPos2i(2,scrHeight-40);
printString(GLUT_BITMAP_HELVETICA_12,buf);
strcpy(buf,"Avenues");
glRasterPos2i(scrWidth/4,-25);
printString(GLUT_BITMAP_HELVETICA_12,buf);
strcpy(buf,"Streets");
verticalPrintString(GLUT_BITMAP_HELVETICA_12,buf,12,-25,scrHeight/2);
numberStreetsAvenues(GLUT_BITMAP_HELVETICA_12,12,
    (scrHeight-50)/25, (scrWidth-50)/25, 25);
glFlush();
}

/* Draw the world boundaries */
/* ... Called by display() to allow display() to be 
   ... more succinct. */
void border()
{
    int i;
    glVertex2i(0,0);
    glVertex2i(0,scrHeight - 50);
    glVertex2i(0,0);
    glVertex2i(scrWidth - 50,0);
}

cont…
/* Draw the streets and avenues */
/* ... Called by display() to allow display() to be 
... more succinct. */
void drawGrid()
{
    int i;
    for(i = 25; i < scrHeight - 50; i += 25)
    {
        glVertex2i(0,i);
        glVertex2i(scrWidth - 50,i);
    }
    for(i = 25; i < scrWidth - 50; i += 25)
    {
        glVertex2i(i,0);
        glVertex2i(i,scrHeight - 50);
    }
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-25.,(GLdouble)scrWidth-25,-25.,(GLdouble)scrHeight-25);
    glMatrixMode(GL_MODELVIEW);
    display();
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);                          /* These 1st four function   */
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);  /*  ... calls are OpenGL/GLUT */
    glutInitWindowSize(scrWidth,scrHeight);        /*  ... preparatory calls     */
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display);    /* <- Registers the programmers drawing code  */
    myinit();             /* This is a local function to establish the    */
    glutReshapeFunc(reshape);     /* <- Registers the programmers drawing code  */
    glutMainLoop( );       /* Starts the event-loop for the graphics environment. */
    return 0;
}

cont…
Writing Text with OpenGL/GLUT

Introduction to C Programming

Copyright © 2015 by James J. Polzin All Rights Reserved

Printed: 6/13/15
LINKED LISTS

- A disadvantage of an array is that it cannot expand & contract once it is allocated.
- To allow individual items to be added and deleted from a set, each element must be individually “linked” into the set so that it’s link can be easily removed or added.
- This type of data structure is called a linked list.
- Structures are tailor made for linked lists. A structure can be defined to contain the data and a link to another structure of the same type. These structures are then called “self-referential” structures since the definition of the structure refers to itself in defining the “link pointer” it contains.
- The start of the list is tracked by a separate stand-alone pointer, often referred to as the “head”, for head-of-the-list.
- The end of the list is indicated by the last link in the list pointing to NULL and not another structure.
- New elements can be added to the end of the list or can even be inserted into the list.
- Unwanted elements can be removed and the pointers can be reassigned around the removed link to maintain a legitimate list with no excess storage.
- The links are generally dynamically allocated so that new links are created as needed and unwanted links can be freed along with their resources.
- Some of the disadvantages of linked lists are that there is overhead in the allocation and deallocation, and there is additional house-keeping in maintaining the list.
Ex:

```c
/*     FILE: ./OpenGL/Drawing.c     */
/* Program that draws some shapes

Functions are defined to generate
the appropriate vertices for OpenGL
when passed reasonable parameters
for a particular shape. */

#include <stdlib.h>
#include "glut.h"
int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit( );
void display( );
void reshape(int, int);
void drawLine(int x1, int y1, int x2, int y2);
void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3);

/* Function to initialize OpenGL parameters and
prepare for drawing. */
void myinit( )
{
  glClearColor(1.,1.,1.,1.); /* white background */
  glColor3f(0.,0.,0.); /* black foreground */
  glShadeModel(GL_FLAT);
  /* set up viewing scrWidth x scrHeight window with origin at
       lower-left. */
  glViewport(0,0,scrWidth,scrHeight);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity( );
  gluOrtho2D(0.0,(GLdouble)scrWidth,0.0,(GLdouble)scrHeight);
  glMatrixMode(GL_MODELVIEW);
}

/* Function registered with OpenGL for producing display */
void display( )
{
  char buf[128];
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_LINES);
  drawLine(0,0,scrWidth,scrHeight);
  drawTriangle(25,15,25+scrWidth/2,15+scrHeight/2,25+scrWidth/2,15);
  glEnd( );
  glFlush( );
}

/* Draw a line between two (x,y) pairs */
void drawLine(int x1, int y1, int x2, int y2)
{
  glVertex2i(x1,y1);
  glVertex2i(x2,y2);
}

cont…
```
/* Draw a triangle with 3 (x,y) pairs */
void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3)
{
    glVertex2i(x1,y1);
    glVertex2i(x2,y2);
    glVertex2i(x2,y2);
    glVertex2i(x3,y3);
    glVertex2i(x3,y3);
    glVertex2i(x1,y1);
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,(GLdouble)scrWidth,0.0,(GLdouble)scrHeight);
    glMatrixMode(GL_MODELVIEW);
    display();
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(scrWidth,scrHeight);
    glutCreateWindow(argv[0]);
    glinit();
    myinit();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop( );
    return 0;
}
Ex:

/*     FILE: ./OpenGL/Drawing2.c     */

/* Program that draws some shapes

Functions to draw additional shapes:

- circles
- arrows, only horizontal and vertical

*/

#include <stdlib.h>
#include <math.h>
#include "glut.h"
define M_PI 3.14159265358979323
int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit( );
void display( );
void reshape(int, int);
void drawLine(int x1, int y1, int x2, int y2);
void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3);
void drawCircle(int cx, int cy, int r);
void drawArrow(int x1, int y1, int x2, int y2);

/* Function to initialize OpenGL parameters and
prepare for drawing. */
void myinit( )
{
    glClearColor(1.,1.,1.,1.); /* white background */
    glColor3f(0.,0.,0.); /* black foreground */
    glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin at
lower-left. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(0.0,(GLdouble)scrWidth,0.0,(GLdouble)scrHeight);
    glMatrixMode(GL_MODELVIEW);
}

/* Function registered with OpenGL for producing display */
void display( )
{
    char buf[128];
    glClearColor(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    drawLine(0,0,scrWidth,scrHeight);
    drawTriangle(25,15,25+scrWidth/2,15+scrHeight/2,25+scrWidth/2,15);
    drawCircle(scrWidth/2,scrHeight/2,100);
    drawArrow(0,scrHeight/2,scrWidth/2,scrHeight/2);
    drawArrow(scrWidth/2,scrHeight/2,scrWidth/2,scrHeight);
    glEnd( );
    glFlush( );
}

cont…
/* Draw a line between two (x,y) pairs */
void drawLine(int x1, int y1, int x2, int y2)
{
    glVertex2i(x1,y1);
    glVertex2i(x2,y2);
}

/* Draw a triangle with 3 (x,y) pairs */
void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3)
{
    glVertex2i(x1,y1);  /* Draw lines between each pair of */
    glVertex2i(x2,y2);  /* ... vertices. */
    glVertex2i(x3,y3);
    glVertex2i(x3,y3);
    glVertex2i(x1,y1);
}

/* Draw a circle with center (x,y) and radius r */
/* Draws a series of line segments to produce a circle. */
void drawCircle(int cx, int cy, int r)
{
    double angle;
    const int segments = 30;

    int x,y,oldX,oldY;
    oldX = cx + r;  /* first point */
    oldY = cy;

    for(angle = M_PI/segments; angle < 2*M_PI; angle += M_PI/segments){
        x = cx + cos(angle)*r; /* Compute next point to draw to */
        y = cy + sin(angle)*r;
        glVertex2i(oldX,oldY); /* Draw between previous point and */
        glVertex2i(x,y);       /* ... new point */
        oldX=x;                /* Shift current point back to previous */
        oldY=y;
    }
}

cont…
/* Draw an arrow from 1st point to 2nd */
void drawArrow(int x1, int y1, int x2, int y2)
{
    int dx1,dy1,dx2,dy2;
    drawLine(x1,y1,x2,y2);
    if(x2<x1){ /* figure out how to "tip" it */
        dx2 = dx1 = +8;
        dy1 = -6;
        dy2 = +6;
    }
    else if(x1<x2){
        dx2 = dx1 = -8;
        dy1 = -6;
        dy2 = +6;
    }
    else if(y1<y2){
        dx2 = +6;
        dx1 = -6;
        dy1 = -8;
        dy2 = -8;
    }
    else if(y2<y1){
        dx2 = +6;
        dx1 = -6;
        dy1 = +8;
        dy2 = +8;
    }
    drawLine(x2,y2,x2+dx1,y2+dy1); /* tip it */
    drawLine(x2,y2,x2+dx2,y2+dy2);
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(0.0,(GLdouble)scrWidth,0.0,(GLdouble)scrHeight);
    glMatrixMode(GL_MODELVIEW);
    display( );
}

int main(int argc, char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(scrWidth,scrHeight);
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display);
    myinit( );
    glutReshapeFunc(reshape);
    glutMainLoop( );

    return 0;
}

cont...
Ex:

/*     FILE: ./OpenGL/Drawing3.c     */

/* Program that draws some shapes

Draws a set of connected arrows.

Reads points from stdin */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "glut.h"

#define SIZE 100

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit();
void display();
void reshape(int, int);
void drawLine(int x1, int y1, int x2, int y2);
void drawArrow(int x1, int y1, int x2, int y2);

int count; /* Number of points */

struct pair{ /* (x,y) pair defined */
  int x;
  int y;
};
/* a set of (x,y) pairs */
struct pair pairs[SIZE];

/* Function to initialize OpenGL parameters and prepare for drawing. */
void myinit()
{
  glClearColor(1.,1.,1.,1.); /* white background */
  glColor3f(0.,0.,0.); /* black foreground */
  glShadeModel(GL_FLAT);
  /* set up viewing scrWidth x scrHeight window with origin at lower-left. */
  glViewport(0,0,scrWidth,scrHeight);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0,(GLdouble)scrWidth,0.0,(GLdouble)scrHeight);
  glMatrixMode(GL_MODELVIEW);
}

/* Function registered with OpenGL for producing display */
void display()
{
  int i;
  char buf[128];
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_LINES);
  for(i=0; i<count-1; i++) /* draws between all the pairs */
    drawArrow(pairs[i].x,pairs[i].y,pairs[i+1].x,pairs[i+1].y);
  glEnd();
  glFlush();
}

cont…
/* Draw a line between two (x,y) pairs */
void drawLine(int x1, int y1, int x2, int y2)
{
    glVertex2i(x1,y1);
    glVertex2i(x2,y2);
}

/* Draw a vertical or horizontal arrow between two (x,y) pairs*/
void drawArrow(int x1, int y1, int x2, int y2)
{
    int dx1,dy1,dx2,dy2;
    drawLine(x1,y1,x2,y2); /* Arrow shaft */
    if(x2<x1){ /* Arrow tip shift */
        dx2 = dx1 = +8;
        dy1 = -6;
        dy2 = +6;
    }
    else if(x1<x2){
        dx2 = dx1 = -8;
        dy1 = -6;
        dy2 = +6;
    }
    else if(y1<y2){
        dx2 = +6;
        dx1 = -6;
        dy1 = -8;
        dy2 = -8;
    }
    else if(y2<y1){
        dx2 = +6;
        dx1 = -6;
        dy1 = +8;
        dy2 = +8;
    }
    drawLine(x2,y2,x2+dx1,y2+dy1); /* Arrow tip 1/2 */
    drawLine(x2,y2,x2+dx2,y2+dy2); /* Arrow tip 1/2 */
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(0.0,(GLdouble)scrWidth,0.0,(GLdouble)scrHeight);
    glMatrixMode(GL_MODELVIEW);
    display( );
}

/* Reads pairs of ints from stdin and counts them */
int readPairs( )
{
    int i = 0;
    while(i < SIZE && scanf("%d%d", &pairs[i].x, &pairs[i].y) == 2)
    {
        i++;
    }

    return i;
}

cont…
```c
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(scrWidth, scrHeight);
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display);
    myinit();
    glutReshapeFunc(reshape);
    count = readPairs(); /* read points to draw */
    glutMainLoop(); /* read points to draw */
    return 0;
}
```

cont…
Ex:

/* FILE: ./OpenGL/Drawing4.c */

/* Program that draws a sequence of horizontal and vertical
arrows from point to point.

Exact amount of storage is allocated after making a first
pass through the file. Then file is rewound and records
are read in.

File name is passed as a command-line argument. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit();
void display();
void reshape(int, int);
void drawLine(int x1, int y1, int x2, int y2);
void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3);
void drawCircle(int cx, int cy, int r);
void drawArrow(int x1, int y1, int x2, int y2);

int count; /* Number of points */

struct pair{ /* (x,y) pair defined */
    int x;
    int y;
};

/* pointer for a set of (x,y) pairs */
struct pair* pairs;

/* Function to initialize OpenGL parameters and
prepare for drawing. */
void myinit()
{
    glClearColor(1.,1.,1.,1.); /* white background */
    glColor3f(0.,0.,0.); /* black foreground */
    glShadeModel(GL_FLAT);
    /* set up viewing scrWidth x scrHeight window with origin at
    lower-left. */
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0,(GLdouble)scrWidth,0.0,(GLdouble)scrHeight);
    glMatrixMode(GL_MODELVIEW);
}

/* Function registered with OpenGL for producing display */
void display()
{
    int i;
    char buf[128];
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    for(i=0; i<count-1; i++)
        drawArrow(pairs[i].x,pairs[i].y,pairs[i+1].x,pairs[i+1].y);
    glEnd();
    glFlush();
}

cont…
/* Draw a line between two (x,y) pairs */
void drawLine(int x1, int y1, int x2, int y2)
{
    glVertex2i(x1, y1);
    glVertex2i(x2, y2);
}

/* Draw a vertical or horizontal arrow between two (x,y) pairs */
void drawArrow(int x1, int y1, int x2, int y2)
{
    int dx1, dy1, dx2, dy2;
    drawLine(x1, y1, x2, y2); /* Arrow shaft */
    if(x2<x1){ /* Arrow tip shift */
        dx2 = dx1 = +8;
        dy1 = -6;
        dy2 = +6;
    } else if(x1<x2){
        dx2 = dx1 = -8;
        dy1 = -6;
        dy2 = +6;
    } else if(y1<y2){
        dx2 = +6;
        dx1 = -6;
        dy1 = -8;
        dy2 = -8;
    } else if(y2<y1){
        dx2 = +6;
        dx1 = -6;
        dy1 = +8;
        dy2 = +8;
    }
    drawLine(x2, y2, x2+dx1, y2+dy1); /* Arrow tip 1/2 */
    drawLine(x2, y2, x2+dx2, y2+dy2); /* Arrow tip 1/2 */
}

/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0, 0, scrWidth, scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)scrWidth, 0.0, (GLdouble)scrHeight);
    glMatrixMode(GL_MODELVIEW);
    display();
}

cont…
/* Reads lines from "file" and counts them, *\nallocates storage for them, then reads *\npairs of ints from "file" */
int readPairs(char * file)
{
    int i, size;
    char buffer[256];
    FILE * fp;

    fp = fopen(file, "r");

    i = 0;
    if(fp != NULL) /* makes first pass to count lines. */
    while(fgets(buffer, 256, fp) != NULL){
        i++;
    }

    size = i; /* allocates enough storage */
    pairs = malloc(sizeof(struct pair) * size);

    if(pairs != NULL) /* checks for allocation */
    rewind(fp);
    for(i = 0; i < size; i++) /* Reads points */
    fscanf(fp, "%d%d", &pairs[i].x, &pairs[i].y);
    else{
        fprintf(stderr, "Malloc failed reading file: %s
", file);
        fclose(fp);
        exit(2);
    }
    fclose(fp);
    exit(1);

    return size;
}

int main(int argc, char** argv)
{
    if(argc < 2)
        fprintf(stderr, "Usage: %s <file of points>
", argv[0]);
    else{
        glutInit(&argc, argv);
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(scrWidth, scrHeight);
        glutCreateWindow(argv[0]);
        glutDisplayFunc(display);
        myinit( );
        glutReshapeFunc(reshape);

        count = readPairs(argv[1]); /* read points to draw */
        glutMainLoop( );
    }

    return 0;
}
Ex:
/*     FILE: ./OpenGL/Drawing5.c     */
/* Program that draws a sequence of horizontal and vertical arrows from point to point. */
/* Exact amount of storage is allocated by building a "linked-list" as the elements are read in. */
/* Advantage: One pass through the file. */
/* Note: -The struct definition has changed. */
/* The code accessing the list must change since it is not longer an array in any sense. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "glut.h"

int scrWidth = 500, scrHeight = 500; /* Initial screen size */
void myinit();
void display();
void reshape(int, int);
void drawLine(int x1, int y1, int x2, int y2);
void drawTriangle(int x1, int y1, int x2, int y2, int x3, int y3);
void drawCircle(int cx, int cy, int r);
void drawArrow(int x1, int y1, int x2, int y2);

int count; /* Number of points */
struct pair{ /* (x,y) pair defined w/ a link*/
  int x;
  int y;
  struct pair * next;
};

/* pointer for head of pairs list */
struct pair* head;

/* Function to initialize OpenGL parameters and prepare for drawing. */
void myinit()
{
  glClearColor(1.,1.,1.,1.); /* white background */
  glColor3f(0.,0.,0.);    /* black foreground */
  glShadeModel(GL_FLAT);
  /* set up viewing scrWidth x scrHeight window with origin at lower-left. */
  glViewport(0,0,scrWidth,scrHeight);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0,(GLdouble)scrWidth,0.0,(GLdouble)scrHeight);
  glMatrixMode(GL_MODELVIEW);
}

cont...
/* Function registered with OpenGL for producing display */
void display()
{
    int i;
    struct pair * current;
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINES);
    /* Process list from head->next to end. */
    for(current=head; current->next != NULL; current = current->next)
        drawArrow(current->x,current->y,current->next->x,current->next->y);
    glEnd( );
    glFlush( );
}

/* Draw a line between two (x,y) pairs */
void drawLine(int x1, int y1, int x2, int y2)
{
    glVertex2i(x1,y1);
    glVertex2i(x2,y2);
}

/* Draw a vertical or horizontal arrow between two (x,y) pairs*/
void drawArrow(int x1, int y1, int x2, int y2)
{
    int dx1,dy1,dx2,dy2;
    drawLine(x1,y1,x2,y2);    /* Arrow shaft */
    if(x2<x1){                /* Arrow tip shift */
        dx2 = dx1 = +8;
        dy1 = -6;
        dy2 = +6;
    } else if(x1<x2){
        dx2 = dx1 = -8;
        dy1 = -6;
        dy2 = +6;
    } else if(y1<y2){
        dx2 = +6;
        dx1 = -6;
        dy1 = -8;
        dy2 = -8;
    } else if(y2<y1){
        dx2 = +6;
        dx1 = -6;
        dy1 = +8;
        dy2 = +8;
    }

    drawLine(x2,y2,x2+dx1,y2+dy1);    /* Arrow tip 1/2 */
    drawLine(x2,y2,x2+dx2,y2+dy2);    /* Arrow tip 1/2 */
}
cont…
/* Function called when the window is reshaped. */
void reshape(int nescrWidth, int nescrHeight)
{
    scrHeight = nescrHeight;
    scrWidth = nescrWidth;
    glViewport(0,0,scrWidth,scrHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(0.0,(GLdouble)scrWidth,0.0,(GLdouble)scrHeight);
    glMatrixMode(GL_MODELVIEW);
    display( );
}

/* Reads pairs of ints from "file" and allocates storage and
   links them into a linked list as they are read. */
int readPairs(char * file)
{
    int i;
    int x, y;            /* (x,y) read in */
    struct pair * last;  /* position of last node. */
    FILE * fp;

    fp = fopen(file,"r");

    i = 0;
    if(fp != NULL){    /* Check if open succeeded */
        i = 0;
        head = NULL;
        last = head;
        /* process file */
        while(fscanf(fp,"%d%d", &x, &y) == 2){
            i++;
            if(last == NULL){ /* Allocate nodes */
                head = last = malloc(sizeof(struct pair));
            }
            else{
                last->next = malloc(sizeof(struct pair));
                last = last->next;
            }
            last->next = NULL; /* assign values */
            last->x = x;
            last->y = y;
        }
        fclose(fp);
    }
    else{
        fprintf(stderr,"File open failed: %s\n", file);
        exit(1);
    }

    return i;
}

cont...
int main(int argc, char** argv)
{
    if(argc < 2)
        fprintf(stderr, "Usage: %s <file of points>\n", argv[0]);
    else{
        glutInit(&argc, argv);
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(scrWidth, scrHeight);
        glutCreateWindow(argv[0]);
        glutDisplayFunc(display);
        myinit( );
        glutReshapeFunc(reshape);

        count = readPairs(argv[1]); /* read points to draw */

        glutMainLoop( );
    }

    return 0;
}

cont…
ADT – ABSTRACT DATA TYPE

• An abstract data type describes a data structure and how it functions without specifying the actual data contents.

• An array is an example of an abstract data type provided by C. We know what arrays are and how they work without knowing what is in them.

• Linked lists are an abstract data type we can build in C. We know what linked lists are and how they work without knowing what is in them.

• C allows us to create an abstract data type that is easily altered when we want to change the data it actually contains.
Ex:

/* file: stack.h */

/* Interface and contained type for a stack */
#ifndef STACK_H_
#define STACK_H_
#include <stdbool.h>

/* Item type to be contained in the stack */
typedef struct pair{    /* (x,y) pair defined w/ a link */
  int x;
  int y;
} Item;

#define MAXSTACK 10    /* Upper bound on stack size */

typedef struct node
{
  Item item;
  struct node * next;
} Node;

typedef struct stack
{
  Node * top;    /* Top of the stack */
  int items;     /* Items in the stack */
} Stack;

void InitializeStack(Stack * ps);
int StackItemCount(const Stack * ps);
void EmptyTheStack(Stack * ps);

bool StackIsFull(const Stack * ps);
bool StackIsEmpty(const Stack * ps);

bool push(Item item, Stack * ps);
bool pop(Item *pitem, Stack * ps);

#endif
Ex:

```c
/* file: stack.c */
/* Stack implementation*/
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

static void ItemIntoNode(Item item, Node * pn);
static void ItemFromNode(Node * pn, Item * pi);

void InitializeStack(Stack * ps)
{
   ps->top = NULL;
   ps->items = 0;
}

bool StackIsFull(const Stack * ps)
{
   return ps->items == MAXSTACK;
}

bool StackIsEmpty(const Stack * ps)
{
   return ps->items == 0;
}

int StackItemCount(const Stack * ps)
{
   return ps->items;
}

bool push(Item item, Stack * ps)
{
   Node * pnew;
   if (StackIsFull(ps))
      return false;
   pnew = (Node *) malloc( sizeof(Node));
   if (pnew == NULL)
   {
      fprintf(stderr,"Unable to allocate memory!\n");
      exit(1);
   }
   ItemIntoNode(item, pnew);
   if (StackIsEmpty(ps))
   {
      ps->top = pnew; /* item pushed in front of top */
      pnew->next = NULL;
   }
   else{
      pnew->next = ps->top; /* top reassigned */
      ps->top = pnew;
   }
   ps->items++; /* items count updated */
   return true;
}

cont…
```
bool pop(Item * pitem, Stack * ps)
{
    Node * pt;

    if (StackIsEmpty(ps))
        return false;
    ItemFromNode(ps->top, pitem);
    pt = ps->top;
    ps->top = ps->top->next;
    free(pt);
    ps->items--;

    return true;
}

/* empty the Stack */
void EmptyTheStack(Stack * ps)
{
    Item dummy;
    while (!StackIsEmpty(ps))
        pop(&dummy, ps);
}

/* data transfer functions */
static void ItemIntoNode(Item item, Node * pn)
{
    pn->item = item;
}

static void ItemFromNode(Node * pn, Item * pi)
{
    *pi = pn->item;
}
Ex:

/* file: stackTest.c */

/* Test a stack type implementation*/
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

void stackStatus(Stack *pstack);
void display(Item i);

int main( )
{
    int i;
    Stack st;
    Item p;
    InitializeStack(&st);
    stackStatus(&st);
    for(i=0; i< 10; i++){
        p.x = i;
        p.y = i*10;
        push(p, &st);
    }
    stackStatus(&st);
    printf("Popping stack contents:
");
    while(!StackIsEmpty(&st)){
        pop(&p, &st);
        display(p);
    }
    return 0;
}

void display(Item i)
{
    printf("\t(%d,%d)\n", i.x, i.y);
}

void stackStatus(Stack *pstack)
{
    printf("Current stack status:\n");
    if(StackIsEmpty(pstack))
        printf("\tStack empty.\n");
    else{
        printf("\tStack non-empty.\n");
        printf("\t size: %d\n", StackItemCount(pstack));
    }
    return;
}

cont...
/* OUTPUT: stackTest.c

Current stack status:
Stack empty.
Current stack status:
Stack non-empty.
size: 10
Popping stack contents:
(9, 90)
(8, 80)
(7, 70)
(6, 60)
(5, 50)
(4, 40)
(3, 30)
(2, 20)
(1, 10)
(0, 0)

*/
Ex:

```c
/* file: stack2.h */

/* Interface and contained type for a stack */
#ifndef STACK2_H_
#define STACK2_H_
#include <stdbool.h>

/* Item type to be contained in the stack */
typedef struct student{ /* A student record */
    long id;
    char name[128];
    float gpa;
} Item;
#endif

#define MAXSTACK 10 /* Upper bound on stack size */

typedef struct node
{
    Item item;
    struct node * next;
} Node;

typedef struct stack
{
    Node * top; /* Top of the stack */
    int items; /* Items in the stack */
} Stack;

void InitializeStack(Stack * ps);
int StackItemCount(const Stack * ps);
void EmptyTheStack(Stack * ps);

bool StackIsFull(const Stack * ps);
bool StackIsEmpty(const Stack *ps);

bool push(Item item, Stack * ps);
bool pop(Item *pitem, Stack * ps);
#endif
```
Ex:
/* file: stack2.c */
/* Stack implementation*/
#include <stdio.h>
#include <stdlib.h>
#include "stack2.h"

static void ItemIntoNode(Item item, Node * pn);
static void ItemFromNode(Node * pn, Item * pi);

void InitializeStack(Stack * ps)
{
    ps->top = NULL;
    ps->items = 0;
}

bool StackIsFull(const Stack * ps)
{
    return ps->items == MAXSTACK;
}

bool StackIsEmpty(const Stack * ps)
{
    return ps->items == 0;
}

int StackItemCount(const Stack * ps)
{
    return ps->items;
}

bool push(Item item, Stack * ps)
{
    Node * pnew;
    if (StackIsFull(ps))
        return false;
    pnew = (Node *) malloc( sizeof(Node));
    if (pnew == NULL)
    {
        fprintf(stderr,"Unable to allocate memory!\n");
        exit(1);
    }
    ItemIntoNode(item, pnew);
    if (StackIsEmpty(ps))
        ps->top = pnew; /* item pushed in front of top */
    pnew->next = NULL;
    } else{
        pnew->next = ps->top; /* top reassigned */
        ps->top = pnew;
    }
    ps->items++;
/* items count updated */
    return true;
}

cont...
bool pop(Item * pitem, Stack * ps)
{
    Node * pt;
    if (StackIsEmpty(ps))
        return false;
    ItemFromNode(ps->top, pitem);
    pt = ps->top;
    ps->top = ps->top->next;
    free(pt);
    ps->items--;
    return true;
}

/* empty the Stack */
void EmptyTheStack(Stack * ps)
{
    Item dummy;
    while (!StackIsEmpty(ps))
        pop(&dummy, ps);
}

/* data transfer functions */
static void ItemIntoNode(Item item, Node * pn)
{
    pn->item = item;
}

static void ItemFromNode(Node * pn, Item * pi)
{
    *pi = pn->item;
}
Ex:

```c
/* file: stackTest2.c */
/* Test a stack type implementation */

#include <stdio.h>
#include <stdlib.h>
#include "stack2.h"

#include <stdio.h>  
#include <stdlib.h>  
#include "stack2.h"

void stackStatus(Stack *pstack);
void display(Item i);

int main( )
{
  int i;
  char buff[128];

  Stack st;
  Item p;

  InitializeStack(&st);
  stackStatus(&st);

  for(i=0; i< 10; i++){
    p.id = 1000+i;
    sprintf(buff,"Jim%d, Polzin%.1f", i, i*1.1);
    strcpy(p.name, buff);
    p.gpa = i%5*1.1;
    push(p, &st);
  }

  printf("\n");
  stackStatus(&st);

  printf("\n");
  printf("Popping stack contents:\n");
  while(!StackIsEmpty(&st)){
    pop(&p, &st);
    display(p);
  }

  return 0;
}

void display(Item i)
{
  printf("%s, %ld, GPA=%f\n", i.name, i.id, i.gpa);
}

void stackStatus(Stack *pstack)
{
  printf("Current stack status: ");
  if(StackIsEmpty(pstack))
    printf("\tStack empty.\n");
  else{
    printf("\tStack non-empty.");
    printf("\t size: %d\n", StackItemCount(pstack));
  }
  return;
}
cont...```
/* OUTPUT: stackTest2.c

Current stack status: Stack empty.

Current stack status: Stack non-empty. size: 10

Popping stack contents:
Jim9, Polzin9.9, 1009, GPA=4.400000
Jim8, Polzin8.8, 1008, GPA=3.300000
Jim7, Polzin7.7, 1007, GPA=2.200000
Jim6, Polzin6.6, 1006, GPA=1.100000
Jim5, Polzin5.5, 1005, GPA=0.000000
Jim4, Polzin4.4, 1004, GPA=4.400000
Jim3, Polzin3.3, 1003, GPA=3.300000
Jim2, Polzin2.2, 1002, GPA=2.200000
Jim1, Polzin1.1, 1001, GPA=1.100000
Jim0, Polzin0.0, 1000, GPA=0.000000

*/
# Index

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADT - ABSTRACT DATA TYPE</td>
<td>270</td>
</tr>
<tr>
<td>ARITHMETIC OPERATORS</td>
<td>20</td>
</tr>
<tr>
<td>ARRAYS</td>
<td>81</td>
</tr>
<tr>
<td>ARRAYS AND POINTERS</td>
<td>94</td>
</tr>
<tr>
<td>BASIC C PROGRAM STRUCTURE</td>
<td>9</td>
</tr>
<tr>
<td>BASIC INPUT AND OUTPUT</td>
<td>11</td>
</tr>
<tr>
<td>BASIC MULTI-DIMENSIONAL ARRAYS</td>
<td>99</td>
</tr>
<tr>
<td>BINARY FILE I/O</td>
<td>172</td>
</tr>
<tr>
<td>BIT OPERATORS</td>
<td>214</td>
</tr>
<tr>
<td>BUILDING LIBRARIES</td>
<td>231</td>
</tr>
<tr>
<td>C OVERVIEW</td>
<td>4</td>
</tr>
<tr>
<td>C STORAGE CLASSES</td>
<td>145</td>
</tr>
<tr>
<td>COMMAND-LINE ARGUMENTS</td>
<td>112</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>8</td>
</tr>
<tr>
<td>COMPIILING AND LINKING</td>
<td>6</td>
</tr>
<tr>
<td>CONDITIONAL STATEMENTS</td>
<td>59</td>
</tr>
<tr>
<td>CONVERSION SPECIFIERS</td>
<td>12</td>
</tr>
<tr>
<td>DYNAMIC MEMORY ALLOCATION</td>
<td>148</td>
</tr>
<tr>
<td>DYNAMIC MULTIDIMENSIONAL ARRAYS</td>
<td>157</td>
</tr>
<tr>
<td>ENUMERATED TYPES</td>
<td>197</td>
</tr>
<tr>
<td>ESCAPE SEQUENCES</td>
<td>13</td>
</tr>
<tr>
<td>FUNCTION POINTERS/QUICK SORT</td>
<td>207</td>
</tr>
<tr>
<td>FUNCTIONS</td>
<td>40</td>
</tr>
<tr>
<td>FUNCTIONS - THE DETAILS</td>
<td>68</td>
</tr>
<tr>
<td>FUNDAMENTAL DATA TYPES</td>
<td>7</td>
</tr>
<tr>
<td>IDENTIFIERS</td>
<td>9</td>
</tr>
<tr>
<td>INCREMENT ++/DECREMENT -- OPERATORS</td>
<td>31</td>
</tr>
<tr>
<td>INDEX</td>
<td>281</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>10</td>
</tr>
<tr>
<td>LINKED LISTS</td>
<td>250</td>
</tr>
<tr>
<td>LOGICAL OPERATORS</td>
<td>49</td>
</tr>
<tr>
<td>LOGICAL, TRUE/FALSE VALUES</td>
<td>49</td>
</tr>
<tr>
<td>LOOPING</td>
<td>50</td>
</tr>
<tr>
<td>MAKE UTILITY</td>
<td>143</td>
</tr>
<tr>
<td>MATH LIBRARIES</td>
<td>56</td>
</tr>
<tr>
<td>MULTIDIMENSIONAL ARRAYS AND POINTERS</td>
<td>101</td>
</tr>
<tr>
<td>OPENGL/GLUT</td>
<td>114</td>
</tr>
<tr>
<td>OPERATOR PRECEDENCE</td>
<td>18</td>
</tr>
<tr>
<td>OPERATOR PRECEDENCE CHART</td>
<td>19</td>
</tr>
</tbody>
</table>

**Introduction to C Programming**  
Page: 281
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPERATORS</td>
<td>17</td>
</tr>
<tr>
<td>POINTERS</td>
<td>78</td>
</tr>
<tr>
<td>PREPROCESSOR - MORE FEATURES</td>
<td>217</td>
</tr>
<tr>
<td>RELATIONAL OPERATORS</td>
<td>49</td>
</tr>
<tr>
<td>STRINGS</td>
<td>32</td>
</tr>
<tr>
<td>STRUCTURES</td>
<td>174</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>2</td>
</tr>
<tr>
<td>TEXT FILE I/O</td>
<td>167</td>
</tr>
<tr>
<td>TYPEDEF</td>
<td>205</td>
</tr>
<tr>
<td>UNIONS</td>
<td>202</td>
</tr>
<tr>
<td>USING SEARCH PATHS</td>
<td>236</td>
</tr>
<tr>
<td>WRITING TEXT WITH OPENGL/GLUT</td>
<td>238</td>
</tr>
</tbody>
</table>