

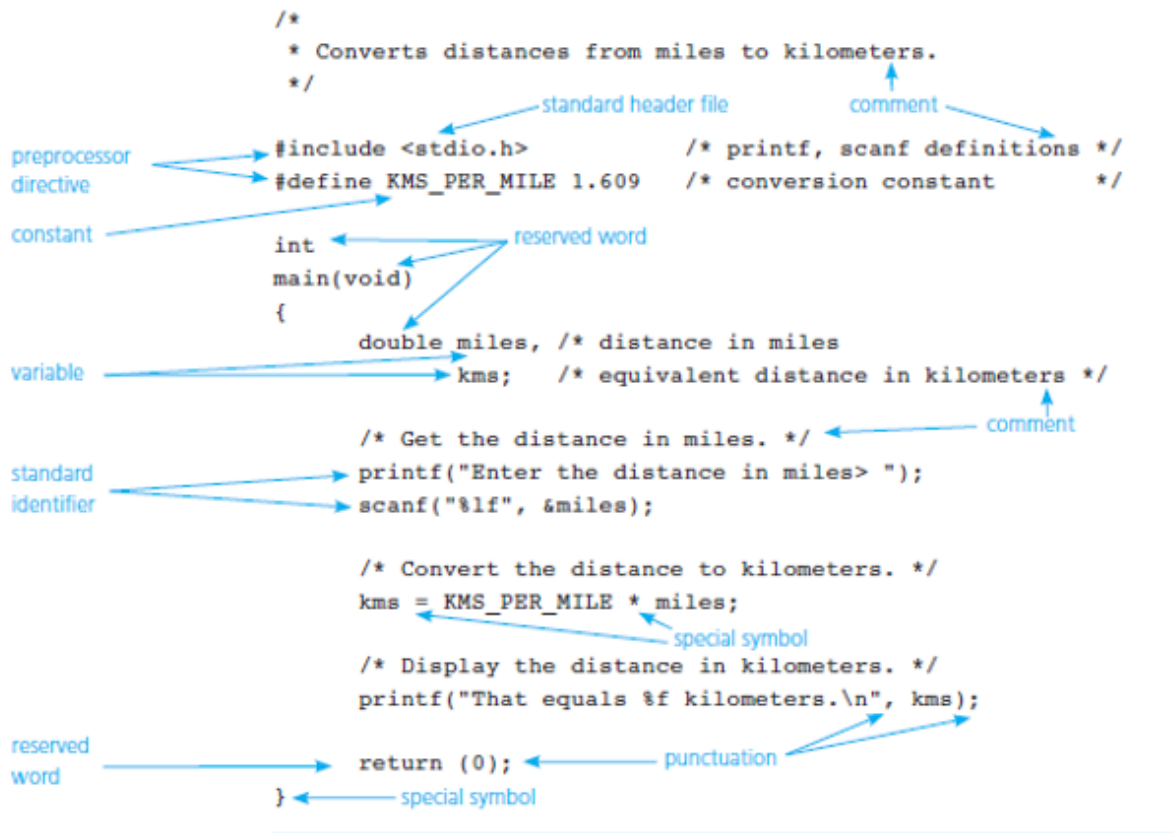
CS 210

August 30, 2016

In Class Assignment 1

Create a new project in C named *ConvertAcres* which inputs a number in acres and produces an equivalent number in square miles. (There are 640 acres in 1 square mile.) Add comments to the top of your program which give your name, the date, the assignment number, and a one or two sentence description of what the program does. Run your program and verify that it works. Print a copy of your source code and turn it in during class.

Figure 2.1 C Language Elements in
Miles-to-Kilometers Conversion Program



Variables in C – Naming rules:

- **Naming rules:**

1. An identifier must consist only of letters, digits and underscores.
2. An identifier cannot begin with a digit.
3. A C reserved word cannot be used as an identifier.
4. An identifier defined in a C standard library should not be redefined.

Data Types:

Data Types

- int
 - a whole number
 - 435
- double
 - a real number with an integral part and a fractional part separated by a decimal point
 - 3.14159
- char
 - an individual character value
 - enclosed in single quotes
 - 'A', 'z', '2', '9', '*', '!'

The `printf` Function

- format string
 - in a call to `printf`, a string of characters enclosed in quotes, which specifies the form of the output line

```
printf("That equals %f kilometers. \n", kms);
```



- print list
 - in a call to `printf`, the variables or expressions whose values are displayed
- placeholder
 - a symbol beginning with % in a format string that indicates where to display the output value

```
printf("That equals %f kilometers. \n", kms);
```



Placeholders in format string

Placeholder	Variable Type	Function Use
% c	char	<code>printf/scanf</code>
% d	int	<code>printf/scanf</code>
% f	double	<code>printf</code>
% lf	double	<code>scanf</code>

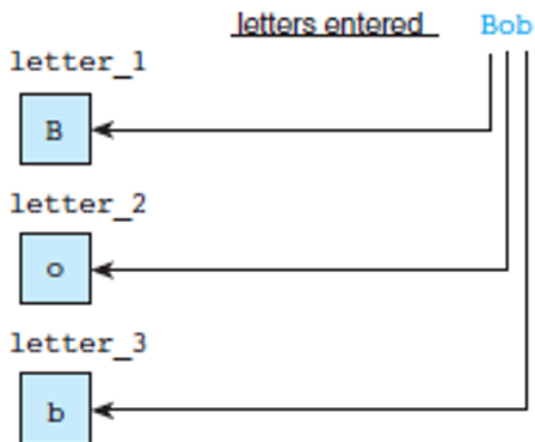
The scanf Function

- Copies data from the standard input device (usually the keyboard) into a variable.

scanf("%lf", miles);

scanf("%c%c%c", &letter_1, &letter_2, &letter_3);

Figure 2.7
Scanning Data Line **Bob**

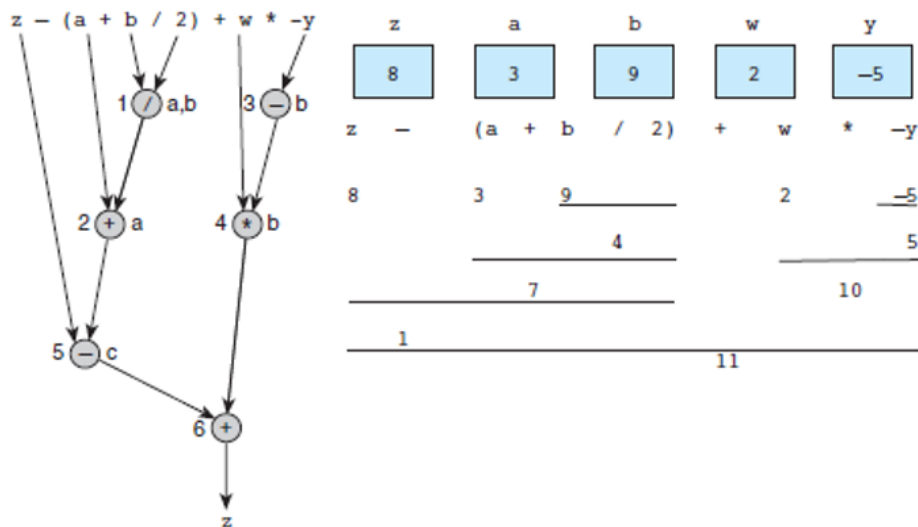


Arithmetic Operator	Meaning	Example
+	addition	5 + 2 is 7 5.0 + 2.0 is 7.0
-	subtraction	5 - 2 is 3 5.0 - 2.0 is 3.0
*	multiplication	5 * 2 is 10 5.0 * 2.0 is 10.0
/	division	5.0 / 2.0 is 2.5 5 / 2 is 2
%	remainder	5 % 2 is 1

Rules for Evaluating Expressions

- Parentheses rule
 - all expression must be evaluated separately
 - nested parentheses evaluated from the inside out
 - innermost expression evaluated first
- Operator precedence rule
 - unary +, - first
 - *, /, % next
 - binary +, - last
- Right Associativity
 - Unary operators in the same subexpression and at the same precedence level are evaluate right to left.
- Left Associativity
 - Binary operators in the same subexpression and at the same precedence lever are evaluated left to right.

Figure 2.12
Evaluation Tree and Evaluation for $z - (a + b / 2) + w * -y$



Supermarket Coin Value Program

```
1. /*
2.  * Determines the value of a collection of coins.
3.  */
4. #include <stdio.h>
5. int
6. main(void)
7. {
8.     char first, middle, last; /* input - 3 initials          */
9.     int pennies, nickels;      /* input - count of each coin type */
10.    int dimes, quarters;       /* input - count of each coin type */
11.    int dollars;               /* input - count of each coin type */
12.    int change;                /* output - change amount          */
13.    int total_dollars;          /* output - dollar amount          */
14.    int total_cents;           /* total cents                      */
15.
16.    /* Get and display the customer's initials. */
17.    printf("Type in your 3 initials and press return> ");
18.    scanf("%c%c%c", &first, &middle, &last);
19.    printf("\n%c%c%c, please enter your coin information.\n",
20.           first, middle, last);
21.
22.    /* Get the count of each kind of coin. */
23.    printf("Number of $ coins > ");
24.    scanf("%d", &dollars);
25.    printf("Number of quarters > ");
26.    scanf("%d", &quarters);
27.    printf("Number of dimes > ");
28.    scanf("%d", &dimes);
29.    printf("Number of nickels > ");
30.    scanf("%d", &nickels);
31.    printf("Number of pennies > ");
32.    scanf("%d", &pennies);
33.
34.    /* Compute the total value in cents. */
35.    total_cents = 100 * dollars + 25 * quarters + 10 * dimes +
36.                 5 * nickels + pennies;
37.
38.    /* Find the value in dollars and change. */
39.    total_dollars = total_cents / 100;
40.    change = total_cents % 100;
41.
42.    /* Display the credit slip with value in dollars and change. */
43.
44.    printf("\n\n%c%c%c Coin Credit\nDollars: %d\nChange: %d cents\n",
45.           first, middle, last, total_dollars, change);
46.    return (0);
47. }
```

Type in your 3 initials and press return> JRH
JRH, please enter your coin information.
Number of \$ coins > 2
Number of quarters> 14
Number of dimes > 12
Number of nickels > 25
Number of pennies > 131

JRH Coin Credit
Dollars: 9
Change: 26 cents

(Continued)

Formatting Values of Type int

Specifying the format of an integer value displayed by a C program is fairly easy. You simply add a number between the % and the d of the %d placeholder in the printf format string. This number specifies the **field width**—the number of columns to use for the display of the value. The statement

```
printf("Results: %3d meters = %4d ft. %2d in.\n",
      meters, feet, inches);
```

indicates that 3 columns will be used to display the value of `meters`, 4 columns will be used for `feet`, and 2 columns will be used for `inches` (a number between 0 and 11). If `meters` is 21, `feet` is 68, and `inches` is 11, the program output will be

```
Results:   21 meters =   68 ft.  11 in.
```

In this line, notice that there is an extra space before the value of `meters` (21) and two extra spaces before the value of `feet` (68). The reason is that the placeholder for `meters` (%3d) allows space for 3 digits to be printed. Because the value of `meters` is between 10 and 99, its two digits are displayed *right-justified*, preceded by one blank space. Because the placeholder for `feet` (%4d) allows room for 4 digits, printing its two-digit value right-justified leaves two extra blank spaces. We can use the placeholder %2d to display any integer value between -9 and 99. The placeholder %4d works for values in the range -999 to 9999. For negative numbers, the minus sign is included in the count of digits displayed.

Table 2.14 shows how two integer values are displayed using different format string placeholders. The character ■ represents a blank character. The last line shows that C expands the field width if it is too small for the integer value displayed.

Formatting Values of Type double

To describe the format specification for a type `double` value, we must indicate both the total *field width* needed and the number of *decimal places* desired. The total field width should be large enough to accommodate all digits before and after the decimal point. There will be at least one digit before the decimal point because a

TABLE 2.14 Displaying 234 and -234 Using Different Placeholders

Value	Format	Displayed Output	Value	Format	Displayed Output
234	%4d	■234	-234	%4d	-234
234	%5d	■■234	-234	%5d	■-234
234	%6d	■■■234	-234	%6d	■■-234
234	%1d	234	-234	%2d	-234

TABLE 2.15 Displaying *x* Using Format String Placeholder `%6.2f`

Value of <i>x</i>	Displayed Output	Value of <i>x</i>	Displayed Output
-99.42	-99.42	-25.554	-25.55
.123	0.12	99.999	100.00
-9.536	-9.54	999.4	999.40

zero is printed as the whole-number part of fractions that are less than 1.0 and greater than -1.0. We should also include a display column for the decimal point and for the minus sign if the number can be negative. The form of the format string placeholder is `%n.mf` where *n* is a number representing the total field width, and *m* is the desired number of decimal places.

If *x* is a type `double` variable whose value will be between -99.99 and 999.99, we could use the placeholder `%6.2f` to display the value of *x* to an accuracy of two decimal places. Table 2.15 shows different values of *x* displayed using this format specification. The values displayed are rounded to two decimal places and are displayed right-justified in six columns. When you round to two decimal places, if the third digit of the value's fractional part is 5 or greater, the second digit is incremented by 1 (-9.536 becomes -9.54). Otherwise, the digits after the second digit in the fraction are simply dropped (-25.554 becomes -25.55).

Table 2.16 shows some values that were displayed using other placeholders. The last line shows it is legal to omit the total field width in the format string placeholder. If you use a placeholder such as `%m.f` to specify only the number of decimal places, the value will be printed with no leading blanks.

TABLE 2.16 Formatting Type double Values

Value	Format	Displayed Output	Value	Format	Displayed Output
3.14159	<code>%5.2f</code>	3.14	3.14159	<code>%4.2f</code>	3.14
3.14159	<code>%3.2f</code>	3.14	3.14159	<code>%5.1f</code>	3.1
3.14159	<code>%5.3f</code>	3.142	3.14159	<code>%8.5f</code>	3.14159
.1234	<code>%4.2f</code>	0.12	-.006	<code>%4.2f</code>	-0.01
-.006	<code>%8.3f</code>	-0.006	-.006	<code>%8.5f</code>	-0.00600
-.006	<code>%3.f</code>	-0.006	-3.14159	<code>%4.f</code>	-3.1416

Finding the Area and Circumference of a Circle

```
1.  /*
2.   * Calculates and displays the area and circumference of a circle
3.   */
4.
5.  #include <stdio.h> /* printf, scanf definitions */
6.  #define PI 3.14159
7.
8.  int
9.  main(void)
10. {
11.     double radius;    /* input - radius of a circle */
12.     double area;      /* output - area of a circle */
13.     double circum;    /* output - circumference */
14.
15.     /* Get the circle radius */
16.
17.     /* Calculate the area */
18.     /* Assign PI * radius * radius to area. */
19.
20.     /* Calculate the circumference */
21.     /* Assign 2 * PI * radius to circum */
22.
23.     /* Display the area and circumference */
24.
25.     return (0);
26. }
```

Program Design


```

1.  /*
2.   * Calculates and displays the area and circumference of a circle
3.   */
4.
5.  #include <stdio.h> /* printf, scanf definitions */
6.  #define PI 3.14159
7.
8.  int
9.  main(void)
10. {
11.     double radius; /* input - radius of a circle */
12.     double area;   /* output - area of a circle */
13.     double circum; /* output - circumference */
14.
15.     /* Get the circle radius */
16.     printf("Enter radius> ");
17.     scanf("%lf", &radius);
18.
19.     /* Calculate the area */
20.     area = PI * radius * radius;
21.
22.     /* Calculate the circumference */
23.     circum = 2 * PI * radius;
24.
25.     /* Display the area and circumference */
26.     printf("The area is %.4f\n", area);
27.     printf("The circumference is %.4f\n", circum);
28.
29.     return (0);
30. }

```

Enter radius> 5.0
The area is 78.5397
The circumference is 31.4159

Program Implementation

Square Root Program

```
1.  /*
2.   * Performs three square root computations
3.   */
4.
5.  #include <stdio.h> /* definitions of printf, scanf */
6.  #include <math.h>  /* definition of sqrt          */
7.
8.  int
9.  main(void)
10. {
11.     double first, second,    /* input - two data values      */
12.            first_sqrt,      /* output - square root of first */
13.            second_sqrt,     /* output - square root of second */
14.            sum_sqrt;         /* output - square root of sum   */
15.
16.     /* Get first number and display its square root. */
17.     printf("Enter the first number> ");
18.     scanf("%lf", &first);
19.     first_sqrt = sqrt(first);
20.     printf("The square root of the first number is %.2f\n", first_sqrt);
```

TABLE 3.1 Some Mathematical Library Functions

Function	Standard Header File	Purpose: Example	Argument(s)	Result
<code>abs(x)</code>	<code><stdlib.h></code>	Returns the absolute value of its integer argument: if <code>x</code> is <code>-5</code> , <code>abs(x)</code> is <code>5</code>	<code>int</code>	<code>int</code>
<code>ceil(x)</code>	<code><math.h></code>	Returns the smallest integral value that is not less than <code>x</code> : if <code>x</code> is <code>45.23</code> , <code>ceil(x)</code> is <code>46.0</code>	<code>double</code>	<code>double</code>
<code>cos(x)</code>	<code><math.h></code>	Returns the cosine of angle <code>x</code> : if <code>x</code> is <code>0.0</code> , <code>cos(x)</code> is <code>1.0</code>	<code>double</code> (radians)	<code>double</code>
<code>exp(x)</code>	<code><math.h></code>	Returns e^x where $e = 2.71828\dots$: if <code>x</code> is <code>1.0</code> , <code>exp(x)</code> is <code>2.71828</code>	<code>double</code>	<code>double</code>
<code>fabs(x)</code>	<code><math.h></code>	Returns the absolute value of its type <code>double</code> argument: if <code>x</code> is <code>-8.432</code> , <code>fabs(x)</code> is <code>8.432</code>	<code>double</code>	<code>double</code>
<code>floor(x)</code>	<code><math.h></code>	Returns the largest integral value that is not greater than <code>x</code> : if <code>x</code> is <code>45.23</code> , <code>floor(x)</code> is <code>45.0</code>	<code>double</code>	<code>double</code>
<code>log(x)</code>	<code><math.h></code>	Returns the natural logarithm of <code>x</code> for <code>x > 0.0</code> : if <code>x</code> is <code>2.71828</code> , <code>log(x)</code> is <code>1.0</code>	<code>double</code>	<code>double</code>
<code>log10(x)</code>	<code><math.h></code>	Returns the base-10 logarithm of <code>x</code> for <code>x > 0.0</code> : if <code>x</code> is <code>100.0</code> , <code>log10(x)</code> is <code>2.0</code>	<code>double</code>	<code>double</code>
<code>pow(x, y)</code>	<code><math.h></code>	Returns x^y . If <code>x</code> is negative, <code>y</code> must be integral: if <code>x</code> is <code>0.16</code> and <code>y</code> is <code>0.5</code> , <code>pow(x,y)</code> is <code>0.4</code>	<code>double</code> , <code>double</code>	<code>double</code>
<code>sin(x)</code>	<code><math.h></code>	Returns the sine of angle <code>x</code> : if <code>x</code> is <code>1.5708</code> , <code>sin(x)</code> is <code>1.0</code>	<code>double</code> (radians)	<code>double</code>
<code>sqrt(x)</code>	<code><math.h></code>	Returns the nonnegative square root of <code>x</code> (\sqrt{x}) for <code>x ≥ 0.0</code> : if <code>x</code> is <code>2.25</code> , <code>sqrt(x)</code> is <code>1.5</code>	<code>double</code>	<code>double</code>
<code>tan(x)</code>	<code><math.h></code>	Returns the tangent of angle <code>x</code> : if <code>x</code> is <code>0.0</code> , <code>tan(x)</code> is <code>0.0</code>	<code>double</code> (radians)	<code>double</code>

P. 121 of the text

DrawCircle might look like this:

```
void DrawCircle()
```

```
{  
    printf("    * *  \n");  
    printf(" *      * \n");  
    printf(" *      * \n");  
    printf("    * *  \n");  
}
```

The DrawTriangle function can be broken down into two simpler pieces: DrawIntersectingLines and DrawBase. The complete program is shown below.

```

#include<stdio.h>
void DrawCircle();           //function prototypes
void DrawTriangle();
void DrawIntersectingLines();
void DrawBase();

```

```

void main(void)
{
    DrawCircle();
    DrawTriangle();
}
void DrawCircle()
{
    printf("  * * \n");
    printf(" *   * \n");
    printf(" *   * \n");
    printf("  * * \n");
}
void DrawTriangle()
{
    DrawIntersectingLines();
    DrawBase();
}
void DrawIntersectingLines()
{
    printf(" / \\ \n");
    printf(" /  \\ \n");
    printf("/   \\ \n");
}
void DrawBase()
{
    printf("-----\n");
}

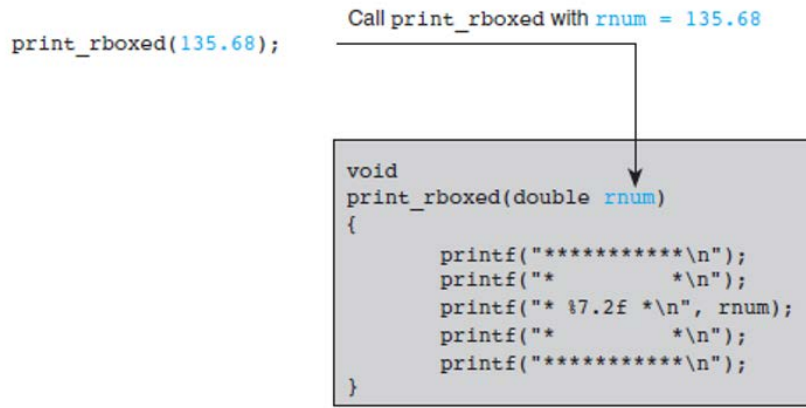
```

```

  * *
 *  *
 *  *
  * *
 / \
 /  \
 /   \
-----

```

Press any key to continue . . .



This function has a single input argument and returns nothing. Its return type is void.

```
#include "stdio.h"
void Fun1(double x); //Prototype - note semicolon at end.

int main()
{double x;
  x = 22;
  //Function call
  Fun1(x); //x here is an argument
  return 0;
}
//Function body
//This function takes one double argument and returns nothing.
//double x is called a formal parameter.
void Fun1(double x) //Function heading (no semicolon)
{double y; //Function body
  y = 3*x*x + 9.7;
  printf("if x = %f, y = %f \n", x, y);
}
```

Note that we could call x by a different name in the function since it is indeed a different variable.

We can change this function to one that accepts a double argument and returns a double result:

```
#include "stdio.h"
double Fun1(double x); //Prototype - note semicolon at end.

int main()
{
    double x, y;
    x = 22;
    //Function call
    y = Fun1(x); //x here is an argument
    printf("if x = %f, y = %f \n", x, y);

    return 0;
}
//Function body
//This function takes one double argument and returns a double.
//double x is called a formal parameter.
double Fun1(double x) //Function heading (no semicolon)
{
    double y; //Function body
    y = 3*x*x + 9.7;
    return y;
}
```

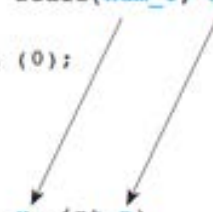
Example

Function scale

```
1. /*
2.  * Multiplies its first argument by the power of 10 specified
3.  * by its second argument.
4.  * Pre : x and n are defined and math.h is included.
5.  */
6. double
7. scale(double x, int n)
8. {
9.     double scale_factor;    /* local variable */
10.    scale_factor = pow(10, n);
11.
12.    return (x * scale_factor);
13. }
```

```
1. /*
2.  * Tests function scale.
3.  */
4. #include <stdio.h>          /* printf, scanf definitions */
5. #include <math.h>          /* pow definition */
6.
7. /* Function prototype */
8. double scale(double x, int n);
9.
10. int
11. main(void)
12. {
13.     double num_1;
14.     int num_2;
15.
16.     /* Get values for num_1 and num_2 */
17.     printf("Enter a real number> ");
18.     scanf("%lf", &num_1);
19.     printf("Enter an integer> ");
20.     scanf("%d", &num_2);
21.
22.     /* Call scale and display result. */
23.     printf("Result of call to function scale is %f\n",
24.           scale(num_1, num_2));      actual arguments
25.
26.     return (0);
27. }
28.
29.
30. double
31. scale(double x, int n)              formal parameters
32. {
33.     double scale_factor;          /* local variable - 10 to power n */
34.
35.     scale_factor = pow(10, n);
36.
37.     return (x * scale_factor);
38. }
```

Enter a real number> 2.5
Enter an integer> -2
Result of call to function scale is 0.025



information flow

Engr 123
Polynomial method

August 31, 2016

Write a console application that prompts the user for a value for x (a double). Evaluate and print the value of y where $y = x^4 - 3x^3 + 2x^2 + 1$. Put the function into a method called FindY which accepts an argument of type double and returns a double.

Turn in a printed copy of your source file.


Selection Structures IF and SWITCH

Relational and Equality Operators

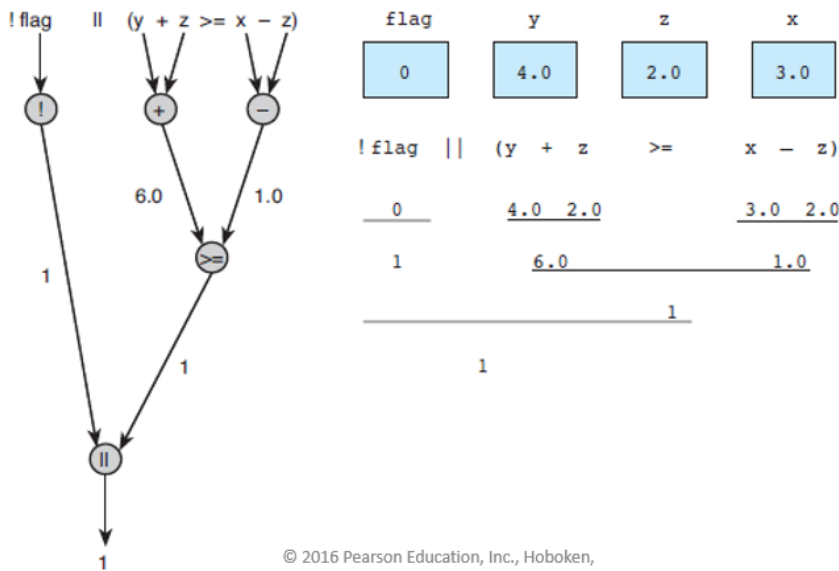
Operator	Meaning	Type
<	less than	relational
>	greater than	relational
<=	less than or equal to	relational
>=	greater than or equal to	relational
==	equal to	relational
!=	not equal to	equality

In addition to the relational operators there are also some logical operators

- logical expressions
 - an expression that uses one or more of the logical operators
 - && (and)
 - || (or)
 - ! (not)

Operator	Precedence
function calls	highest (evaluated first)
! + - & (unary operator)	
* / %	
+ -	
< <= >= >	
== !=	
&&	
=	lowest (evaluated last)

!flag || (y + z >= x - z)



Note that a logical expression stops being evaluated as soon as the outcome can be determined.

For example, suppose $x = 0$
 $(x \neq 0 \ \&\& \ y/x > 5)$ will be OK but
 $(y/x > t \ \&\& \ x \neq 0)$ will fail

Also, we can compare characters with logical expressions:

Expression	Value
'g' >= '0'	1 (true)
'a' < 'e'	1 (true)
'B' <= 'A'	0 (false)
'Z' == 'z'	0 (false)
'a' <= 'A'	System dependent
'a' <= ch && ch <= 'z'	1 (true) if ch is a lowercase letter

The IF statement is a selection structure that makes use of a logical expression. Its general form is:

```
if(logical expression)
{
    statements to do if
    logical expression is true
}
else
{
    statements to do if
    logical expression is false.
}
```

There are some variations and options on this general form:

If statement with one alternative

```
if (x != 0)
    product = product * x;
```

If statement with two alternatives

```
if (rest_heart_rate > 75)
    printf("Keep up your exercise program!\n");
else
    printf("Your hear is doing well!\n");
```



```

#include<stdio.h>
#include<math.h>
//Quad1 - finds the roots of the quadratic equation
//          if the roots are real
int main()
{
    double a, b, c;
    double discr;
    double root1, root2;
    printf("Enter three coefficients... ");
    scanf_s("%lf%lf%lf", &a, &b, &c);
    discr = b*b - 4*a*c;
    if(discr < 0)
        printf("discriminant is negative\n");
    else
    {
        root1 = (-b + sqrt(discr))/(2*a);
        root2 = (-b - sqrt(discr))/(2*a);
        printf("roots are %lf and %lf \n", root1, root2);
    }
}

```

Example:

Quadratic equation with three cases

```
#include<stdio.h>
#include<math.h>
//Quad2 - finds the roots of the quadratic equation
int main()
{
    double a, b, c;
    double discr;
    double root1, root2;
    double real, imag;
    printf("Enter three coefficients... ");
    scanf_s("%lf%lf%lf", &a, &b, &c);
    discr = b*b - 4*a*c;
    if(discr > 0)
    {
        printf("Roots are positive and real. \n");
        root1 = (-b + sqrt(discr))/(2*a);
        root2 = (-b - sqrt(discr))/(2*a);
        printf("roots are %lf and %lf \n", root1, root2);
    }
    else if(discr == 0)
    {
        printf("Roots are real and equal. \n");
        root1 = -b/(2*a);
        printf("roots are %lf and %lf \n", root1, root1);
    }
    else
    {
        printf("Roots are complex. \n");
        real = -b/(2*a);
        imag = (sqrt(-discr))/(2*a);
        printf("roots are %lf +/- i%lf \n", real, imag);
    }
}
```

Common programming problems with IF and logical expressions:

1. Be very careful not to confuse = with ==

The following statement is legal and gives no warnings:

```
else if(discr = 0)
{
    printf("Roots are real and equal. \n");
    root1 = -b/(2*a);
    printf("roots are %lf and %lf \n", root1, root1);
}
```

The logical expression uses = instead of == so it assigns discr to 0. It then checks to see if the result is true or false. A 0 is false so this expression will always be false.

2. Remember that the logical operators need two arguments (except for !).

```
if(5 < x < 10)
```

is incorrect. For example if $x = 6$ this will ask is $5 < 6$ which is true but that's a one so it will then ask if $1 < 10$ which is true.

You should write this as:

```
if(5 < x && x < 10)
```

3. Remember that C ignores spaces which can lead to a dangling else

```
if(x < 5)
    printf("%lf", x);
else
    if(x < 10)
        printf("%lf", x - 5);
else
    printf("%lf", x + 10);
```

At first glance this looks like the second else belongs to the first if because of the spacing. But it, in fact, belongs to the second if. This would be more clear if we used braces.

```
if(x < 5)
    printf("%lf", x);
else
{
    if(x < 10)
        printf("%lf", x - 5);
    else
        printf("%lf", x + 10);
}
```

The switch statement

- also used to select one of several alternatives
- useful when the selection is based on the value of
 - a single variable
 - or a simple expression
- values may of type int or char
 - not double

controlling expression



```
switch (controlling expression) {  
    label set1  
        statements1  
        break;  
    label set2  
        statements2  
        break;  
    .  
    .  
    .  
    label setn  
        statementsn  
        break;  
}
```

```

1.  /*
2.   * Reads serial number and displays class of ship
3.   */
4.
5.  #include <stdio.h>
6.
7.  int
8.  main(void)
9.  {
10.     char class;    /* input - character indicating class of ship */
11.
12.     /* Read first character of serial number */
13.     printf("Enter ship serial number> ");
14.     scanf("%c", &class);    /* scan first letter */
15.
16.     /* Display first character followed by ship class */
17.     printf("Ship class is %c: ", class);
18.     switch (class) {
19.     case 'B':
20.     case 'b':
21.         printf("Battleship\n");
22.         break;
23.     case 'C':
24.     case 'c':
25.         printf("Cruiser\n");
26.         break;
27.     case 'D':
28.     case 'd':
29.         printf("Destroyer\n");
30.         break;
31.     case 'F':
32.     case 'f':
33.         printf("Frigate\n");
34.         break;
35.     default:
36.         printf("Unknown\n");
37.     }
38.
39.     return (0);
40. }

```

Write a console application that prompts the user for a value for x (a double). Print the corresponding value of y according to the following table.

$x < 0$	$y = \text{absolute value of } x$
$0 \leq x < 12$	$y = x^2$
$12 \leq x \leq 50$	$y = (x - 12)^2$
$x > 50$	$y = \sqrt{x}$

Turn in a printed copy of your source file. Include your name, the date, and the assignment title (IF/ELSE).

Loops

Repetition in Programs

- loop
 - a control structure that repeats a group of steps in a program
- loop body
 - the statements that are repeated in the loop

Comparison of Loop Kinds

- counting loop
 - we can determine before loop execution exactly how many loop repetitions will be needed to solve the problem
 - while, for
- sentinel-controlled loop
 - input of a list of data of any length ended by a special value
 - while, for
- ~~endfile~~-controlled loop
 - input of a single list of data of any length from a data file
 - while, for
- input validation loop
 - repeated interactive input of a data value until a value within the valid range is entered
 - do-while
- general conditional loop
 - repeated processing of data until a desired condition is met
 - while, for

Counting Loops

- counter-controlled loop
 - a.k.a. counting loop
 - a loop whose required number of iterations can be determined before loop execution begins
- loop repetition condition
 - the condition that controls loop repetition
- loop control variable
 - the variable whose value controls loop repetition
- infinite loop
 - a loop that executes forever

while Statement Syntax

```
while (loop repetition condition)  
    statement;
```

```
/* display N asterisks. */  
count_star = 0  
while (count_star < N) {  
    printf("*");  
    count_star = count_star + 1;  
}
```

Computing a Sum or Product in a Loop

- accumulator
 - a variable used to store a value being computed in increments during the execution of a loop

Do Example of accumulator loop

```
#include<stdio.h>
int main()
{
    int i, sum;
    sum = 0;           //initialize accumulator value to zero
    i = 1;             //initialize loop control
    while(i < 100)
    {
        sum = sum + i;
        i = i + 2;     //update loop control
    }
    printf("sum of the odd numbers 1 to 100 is %d\n", sum);
}
```

Do Example of Fibonacci numbers

```
#include<stdio.h>
int main()
{
    int i, f, fn1, fn2;
    fn1 = 1;
    fn2 = 0;
    i = 2;
    printf("Fibonacci 0 = %d\n", fn2);
    printf("Fibonacci 1 = %d\n", fn1);
    while(i < 20)
    {
        f = fn1 + fn2;
        printf("Fibonacci %d = %d\n", i, f);
        fn2 = fn1;
        fn1 = f;
        i++;
    }
    return 0;
}
```

Do Example to evaluate $y = 3x^3 - 12x^2 + 4x - 3$ for values of x starting at 0 and continuing in steps of 0.01 until y is greater than 1000. Print the first value of y and the corresponding value of x for which $y > 1000$.

```
#include<stdio.h>
#include<math.h>
double FindY(double x);
int main()
{
    double x, y, xIncr;
    x = 0;
    xIncr = 0.01;
    y = FindY(x);
    while(y <= 1000)
    {
        x = x + xIncr;
        y = FindY(x);
    }
    printf("x = %lf y = %lf\n", x, y);
    return 0;
}
double FindY(double x)
{
    double y;
    y = 3*pow(x, 3) - 12*x*x + 4*x - 3;
    return y;
}
```

Example: The value of π can be approximated from the series given by

$$\pi = 4 \times \left\{ 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15} \dots \right\}$$

```
#include<stdio.h>
//Pi = 4*{1 - 1/3 + 1/5 - 1/7 + 1/9 - ...}
// The user enters the number of terms to use.
int main()
{int terms, i;
  double pi;
  printf("Enter the number of terms to use... ");
  scanf_s("%d", &terms);
  i = 0;
  pi = 0;
  while(i < terms)
  {if(i % 2 == 0)
    pi = pi + 1.0/(2*i+1); //Add even term.
    //Note that 2i+1 is always odd
    else
    pi = pi - 1.0/(2*i+1); //Subtract odd term
    i++;
  }
  pi = pi * 4;
  printf("For %d terms, pi = %lf\n", terms, pi);
  return 0;
}
```

Example: Write a program to find the integer square root of a number input from the user. The integer square root is the largest integer whose square is less than or equal to the number. Use a loop and do this program with an exhaustive search.

```
#include<stdio.h>
int main()
{
    int i, n;
    printf("Enter an integer greater than zero... ");
    scanf_s("%d", &n);
    i = 1;
    while(i*i <= n)
    {
        i = i + 1;
    }
    i = i - 1;
    printf("%d is the integer square root of %d\n", i, n);
}
```

Write a console application that prompts the user for a value which can be used to increment the value of x . Evaluate and print the value of y where $y = x^4 - 3x^3 + 2x^2 + 1$. Use a loop to allow the value of x to go from 0 to 10 while x is incremented by the value from the user.

For example, if the user inputs a value of 1, your program should print values for y for $x = 0, 1, 2, 3, 4, \dots 10$.

Put the function into a method called FindY which accepts an argument of type double and returns a double.

Turn in a printed copy of your source file.

General Conditional Loop

1. Initialize loop control variable.
2. As long as exit condition hasn't been met
3. Continue processing

Loop Control Components

- initialization of the loop control variable
 - test of the loop repetition condition
 - change (update) of the loop control variable
-
- the **for** loop supplies a designated place for each of these three components

The **for** Statement Syntax

```
for (initialization expression;  
    loop repetition condition;  
    update expression)  
statement;
```

```
/* Display N asterisks. */  
for (count_star = 0;  
    count_star < N;  
    count_star += 1)  
    printf("*");
```

Example Countable loop

TABLE 5.3 Compound Assignment Operators

Statement with Simple Assignment Operator	Equivalent Statement with Compound Assignment Operator
<code>count_emp = count_emp + 1;</code>	<code>count_emp += 1;</code>
<code>time = time - 1;</code>	<code>time -= 1;</code>
<code>total_time = total_time + times;</code>	<code>total_time += times;</code>
<code>product = product * item;</code>	<code>product *= item;</code>
<code>n = n * (x + 1);</code>	<code>n *= x + 1;</code>

The **for** Statement Syntax

```
for (initialization expression;  
    loop repetition condition;  
    update expression)  
    statement;
```

```
/* Display N asterisks. */  
for (count_star = 0;  
    count_star < N;  
    count_star += 1)  
    printf("*");
```

More Loops

General Conditional Loop

1. Initialize loop control variable.
2. As long as exit condition hasn't been met
3. Continue processing

Loop Control Components

- initialization of the loop control variable
 - test of the loop repetition condition
 - change (update) of the loop control variable
-
- the **for** loop supplies a designated place for each of these three components

The **for** Statement Syntax

```
for (initialization expression;  
    loop repetition condition;  
    update expression)  
statement;
```

```
/* Display N asterisks. */  
for (count_star = 0;  
    count_star < N;  
    count_star += 1)  
    printf("*");
```

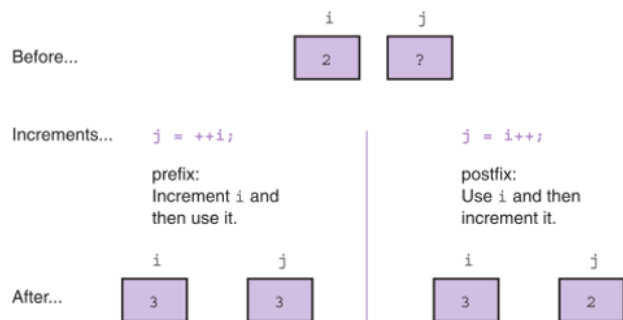
Example Countable loop

TABLE 5.3 Compound Assignment Operators

Statement with Simple Assignment Operator	Equivalent Statement with Compound Assignment Operator
<code>count_emp = count_emp + 1;</code>	<code>count_emp += 1;</code>
<code>time = time - 1;</code>	<code>time -= 1;</code>
<code>total_time = total_time + times;</code>	<code>total_time += times;</code>
<code>product = product * item;</code>	<code>product *= item;</code>
<code>n = n * (x + 1);</code>	<code>n *= x + 1;</code>

Increment and Decrement Operators

- `counter = counter + 1`
`count += 1`
`counter++`
- `counter = counter - 1`
`count -= 1`
`counter--`



Example of a for loop

```
#include<stdio.h>
int main()
{
    int i, f, fn1, fn2;
    fn1 = 1;
    fn2 = 0;
    printf("Fibonacci 0 = %d\n", fn2);
    printf("Fibonacci 1 = %d\n", fn1);
    for(i=2;i<20;i++)
    {
        f = fn1 + fn2;
        printf("Fibonacci %d = %d\n", i, f);
        fn2 = fn1;
        fn1 = f;
    }
    return 0;
}
```

```
int main()
{
    int i, f, fn1, fn2;
    fn1 = 1;
    fn2 = 0;
    i = 2;
    printf("Fibonacci 0 = %d\n", fn2);
    printf("Fibonacci 1 = %d\n", fn1);
    while(i < 20)
    {
        f = fn1 + fn2;
        printf("Fibonacci %d = %d\n", i, f);
        fn2 = fn1;
        fn1 = f;
        i++;
    }
    return 0;
}
```


Example Factorial $n! = n*(n-1)*...*1$

```
#include<stdio.h>
#define FACTMAX 20
int Factorial(int n);
int main()
{
    int i;
    for(i=0;i < FACTMAX;i++)
        printf("Factorial of %d = %d\n", i, Factorial(i));
    return 0;
}
int Factorial(int n)
{
    int i, fact;
    if(n == 0)
        return 1;
    fact = 1;
    for(i=1;i<=n;i++)
        fact *= i;
    return fact;
}
```

Example Print a table of Fahrenheit to Celsius conversion

<pre>#include<stdio.h> double ConvertToCelsius(double tF); int main() { int i; double tC, tF; tF = -40; printf("Fahrenheit Celsius\n"); for(i=0;i < 15;i++) { tC = ConvertToCelsius(tF); printf("%8.2f %8.2f\n", tF, tC); tF = tF + 10; } } double ConvertToCelsius(double tF) { return 5.0*(tF - 32)/9.0; }</pre>	<table><thead><tr><th>Fahrenheit</th><th>Celsius</th></tr></thead><tbody><tr><td>-40.00</td><td>-40.00</td></tr><tr><td>-30.00</td><td>-34.44</td></tr><tr><td>-20.00</td><td>-28.89</td></tr><tr><td>-10.00</td><td>-23.33</td></tr><tr><td>0.00</td><td>-17.78</td></tr><tr><td>10.00</td><td>-12.22</td></tr><tr><td>20.00</td><td>-6.67</td></tr><tr><td>30.00</td><td>-1.11</td></tr><tr><td>40.00</td><td>4.44</td></tr><tr><td>50.00</td><td>10.00</td></tr><tr><td>60.00</td><td>15.56</td></tr><tr><td>70.00</td><td>21.11</td></tr><tr><td>80.00</td><td>26.67</td></tr><tr><td>90.00</td><td>32.22</td></tr><tr><td>100.00</td><td>37.78</td></tr></tbody></table> <p>Press any key to continue . . .</p>	Fahrenheit	Celsius	-40.00	-40.00	-30.00	-34.44	-20.00	-28.89	-10.00	-23.33	0.00	-17.78	10.00	-12.22	20.00	-6.67	30.00	-1.11	40.00	4.44	50.00	10.00	60.00	15.56	70.00	21.11	80.00	26.67	90.00	32.22	100.00	37.78
Fahrenheit	Celsius																																
-40.00	-40.00																																
-30.00	-34.44																																
-20.00	-28.89																																
-10.00	-23.33																																
0.00	-17.78																																
10.00	-12.22																																
20.00	-6.67																																
30.00	-1.11																																
40.00	4.44																																
50.00	10.00																																
60.00	15.56																																
70.00	21.11																																
80.00	26.67																																
90.00	32.22																																
100.00	37.78																																

Conditional Loops

- used when there are programming conditions when you will not be able to determine the exact number of loop repetitions before loop execution begins

Example of a conditional loop

The trigonometric $\sin(x)$ function can be written as an infinite sum as:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

where x is a real number in radians. Write a program in C which will prompt the user for a value of x in degrees and an accuracy number and print the value of $\sin(x)$ to the required accuracy.

Take $\pi = 3.141592653589793$

```
#include<stdio.h>
#include<math.h>
#define PI 3.141592653589793
double Factorial(int n);
int main()
{
    int i;
    double degrees, x, term, acc, sinx;
    printf("This program computes the value of sin(x)\n");
    printf("Enter a value for x in degrees ... ");
    scanf_s("%lf", &degrees);
    printf("Enter a value for the accuracy ... ");
    scanf_s("%lf", &acc);
    x = degrees * PI/180;
    i = 0;
    term = pow(x, (2*i+1))/Factorial(2*i+1);
    sinx = term;
    while(term > acc)
    {
        i++;
        term = pow(x, (2*i+1))/Factorial(2*i+1);
        sinx += pow(-1, i)*term;
    }
    printf("Sin(%5.2f) = %10.8f\n", degrees, sinx);
}

double Factorial(int n)
{
    double fact;
    if(n == 0)
        return 1.0;
    fact = 1;
    while(n > 0)
    {
        fact = fact*n;
        n--;
    }
    return fact;
}
```

Loop Design

- Sentinel-Controlled Loops
 - sentinel value: an end marker that follows the last item in a list of data
- Endfile-Controlled Loops
- Infinite Loops on Faulty Data

Sentinel Loop Design

- Correct Sentinel Loop
 1. Initialize **sum** to **zero**.
 2. Get first **score**.
 3. while **score** is not the sentinel
 4. Add **score** to **sum**.
 5. Get next **score**

Sentinel Loop Design

- Incorrect Sentinel Loop
 1. Initialize **sum** to **zero**.
 2. while **score** is not the sentinel
 3. Get **score**
 4. Add **score** to **sum**.

Example: find the average of numbers input from the keyboard.

```
#include<stdio.h>
int main()
{
    int n, sum, count;
    double avg;
    count = 0;
    sum = 0;
    printf("Enter numbers to find the average.\n");
    printf("Enter 0 to end.\n");
    printf("Enter the first number ... ");
    scanf_s("%d", &n);
    while(n != 0)
    {
        count++;
        sum += n;
        printf("Enter the next number ... ");
        scanf_s("%d", &n);
    }
    if(count > 0)
    {
        avg = (double)sum/count;
        printf("The average is %3.4f\n", avg);
    }
    return(0);
}
```

```
Enter numbers to find the average.
Enter 0 to end.
Enter the first number ... 1
Enter the next number ... 2
Enter the next number ... 3
Enter the next number ... 0
The average is 2.0000
Press any key to continue . . .
```

End Of File (EOF) loops are similar to sentinel loops except the program reads a file until it comes to an EOF marker. We will do this later.

Nested Loops

- Loops may be nested just like other control structures
- Nested loops consist of an outer loop with one or more inner loops
- Each time the outer loop is repeated, the inner loops are reentered, their loop control expressions are reevaluated, and all required iterations are performed

Example of nested loops:

Print the sequence

000
001
010
011
100
101
110
111

Notice that the least significant digit changes on every line, the next on every other line, and the final only on every fourth line.

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

```
int main()
```

```
{
```

```
    int i, j, k;
```

```
    for(i=0;i<2;i++)
```

```
    {
```

```
        for(j=0;j<2;j++)
```

```
        {
```

```
            for(k=0;k<2;k++)
```

```
                printf("%d%d%d\n", i, j, k);
```

```
        }
```

```
    }
```

```
}
```

000
001
010
011
100
101
110
111

Press any key to continue . . .

How many line of print do each of the following produce? The variables i, j, and k are integers.

```
i = 5;
while(i>1)
{printf("1\n";
j = 3;
while(j < 5)
{printf("2\n";
j++;
}
i--;
}
```

Lines of Print _____

```
B)int i, j, k;
i = -9;j = 1;k = 7;
while(i < 0)
{while(j > -2)
{printf("%d\n", k);
j--;
k = k - 2;
}
i++;
}
```

Lines of Print _____

The following function takes two arguments x and y and produces one returned variable z. All inputs and outputs are integers. Write a program that contains a nested loop which will use the function to calculate values of z as x goes from 0 to 5 and y goes from 0 to 3. Print each value of z on a new line.

```
int FindZ(int x, int y)
{
    if(x >= 0 && y >= 0)
        return x*x + y*y
    return 0;
}
```

How many line of print do each of the following produce? The variables i, j, and k are integers.

```
i = 52;
while(i > 5)
{printf("%d", i);
  i--;
}

for(j=-1; j<3; j++)
{printf("%d", j);
  for(k=1; k<5; k++)
    printf("%d", k);
}
```

do-while Statement

- For conditions where we know that a loop must execute at least one time
 1. Get a *data value*
 2. If *data value* isn't in the acceptable range, go back to step 1.

do-while Syntax

do

statement;

while (loop repetition condition);

Do while example:

Write a function to prompt the user to enter a positive even number from the keyboard.

```
#include <stdio.h>
int GetEven();
int main()
{
    int even;
    even = GetEven();
    printf("%d", even);
    return(0);
}
int GetEven()
{
    int status;
    int even;
    do
    {
        printf("Enter a positive even integer... ");
        status = scanf_s("%d", &even);
    }while(status > 0 && even % 2 != 0);
    return even;
}
```

Notes:

1. Do while must execute statements *at least* once since the loop condition is not checked until the end of the loop.
2. The scanf function returns the number of characters that was read and stored. If an error occurs or end-of-file is reached before any items could be read, it will return EOF. For example if you try to read a char with the %d format specifier the status will come back as 0.
3. The while statement at the end of a do-while loop ends in a semicolon. The normal while loop does not.

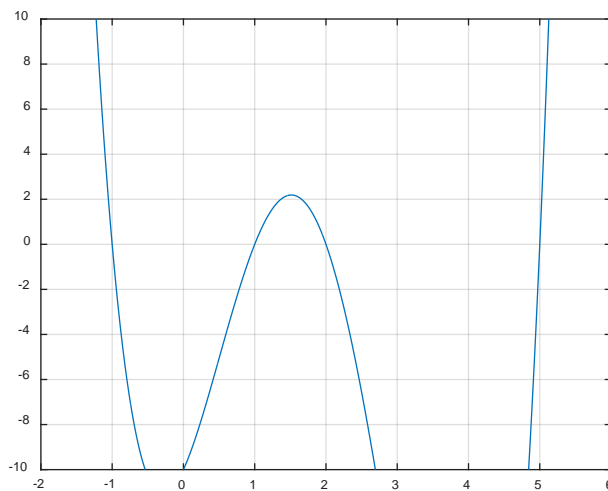
Iterative Approximations

Iterative approximations use the computer to effectively guess at results until it gets an answer within a suitable range of accuracy.

Example

Write a program to find the real roots of an equation over a given interval by examining all of the points in the interval where the function crosses zero.

The function is $y = x^4 - 7x^3 + 9x^2 + 7x - 10$



This program examines a function looking for real roots between START and END.

Roots are found when there is a zero crossing.

We start with a large increment of 0.1. When we find a zero crossing we back up, decrease the increment by 10 and continue searching until the increment gets to be less than the tolerance.

To find a zero crossing we evaluate a function at the left and right end of the interval whose length is xIncr.

```
while(left*right > 0 && x < END)
{
    x = x + xIncr;
    left = FindY(x);
    right = FindY(x + xIncr);
}
```

When we drop out of this loop we are either at the end or we have found a zero crossing. This loop goes inside a larger loop which decreases the size of the increment, backs up to the start of the interval, and finds a zero crossing to a better accuracy.

```
while(xIncr >= tolerance && x < END)
{
    while(left*right > 0 && x < END)
    {
        x = x + xIncr;
        left = FindY(x);
        right = FindY(x + xIncr);
    }
    x = x - xIncr;           //back up
    xIncr = xIncr/10;       //decrease increment
    left = FindY(x);
    right = FindY(x + xIncr);
}
```

When we drop out of this loop we are either at the end or we have found a root to the required tolerance. We can print the root and continue searching for more roots by setting the new interval left side immediately to the right of the root.

```

#include<stdio.h>
#include<math.h>
#define START -10
#define END 10
double FindY(double x);
int main()
{
    double tolerance, x, xIncr;
    double left, right;
    tolerance = 0.001;
    x = START;
    while(x < END)
    {
        x = x + tolerance;
        xIncr = 0.1;
        left = FindY(x);
        right = FindY(x + xIncr);
        while(xIncr >= tolerance && x < END)
            {while(left*right > 0 && x < END)
                {
                    x = x + xIncr;
                    left = FindY(x);
                    right = FindY(x + xIncr);
                }
                x = x - xIncr;
                xIncr = xIncr/10;
                left = FindY(x);
                right = FindY(x + xIncr);
            }
        x = x + xIncr*10;
        if(x < END)
            printf("Root is between %10.6f and %10.6f\n", x, x+tolerance);
    }
}

double FindY(double x)
{
    return pow(x, 4) - 7*pow(x, 3) + 9*x*x + 7*x -10;
}

```

Numerical integration

In numerical integration using rectangles we break a function up into small rectangles and add up the area of each rectangle to get a value for the integral over the range.

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

const double INCR = 0.01;
double F(double x);
int main()
{
    double newX, oldX;
    double sumRec = 0, sumTrap = 0;
    newX = INCR;
    while (newX <= 10)
    {
        sumRec = sumRec + INCR*F(newX);
        newX = newX + INCR;
    }
    printf("The true value of the integral is...%10.6f \n ", 14356.66667);
    printf("The integral for rectangular integration is... %10.6f\n ",
sumRec);
    //
    return 0;
}
//
double F(double x)
{
    return pow(x, 4) - pow(x, 3) -10*x*x +3*x + 4;
}
```

Write a C-program which will find an integer solution to the two equations given by:

$$a + b + c = 1000$$

$$a^2 + b^2 = c^2$$

Write the program by doing an exhaustive search through all of the triples from 1 to 998.