CHAPTER 2

SIMPLE C PROGRAMS

Program Structure

Let us analyze the structure of the simple C program in Chapter 1

•The first lines of the program contain comments that document its purpose:

```
//  Purpose: This program computes the distance between two points
//  Input(s): two points
//  Output(s): distance between points
//  Written by: He
//  Date: 8/12
```

•Comments begin with /* and end with */ characters. If only one line will be used then may use //

Program Structure

- **Preprocessor directives** provide instructions that are performed before the program is compiled:

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

- These directives specify that statements in the files *stdio.h* and *math.h* should be included in place of these two statements

Program Structure

- Every C program contains a set of statements called a **main** function. The keyword **int** indicates that the function returns an integer value. The keyword **void** indicates that the function is not receiving any information from the operating system:

```
int main(void)
{
```

Program Structure

- The main function contains two types of commands:
  1. **Declarations**
  2. Statements

```
// Declare and initialize variables
double x1=1, y1=5, x2=4, y2=7, side1, side2, distance;
```

- Declarations define memory locations and may or may not give initial values to be stored in memory

Program Structure

- The main function contains two types of commands:
  1. Declarations
  2. **Statements**

```
//  Compute the sides of right triangle
side1 = x2 - x1;
side2 = y2 - y1;
distance = sqrt(side1*side1 + side2*side2);

//  Print distance
printf("The distance between points is %5.2f",distance);
getch();
```

- Statements specify the operations to be performed in the program

Program Structure

•To end execution of the program and return control to the operating system, we use a return0; statement

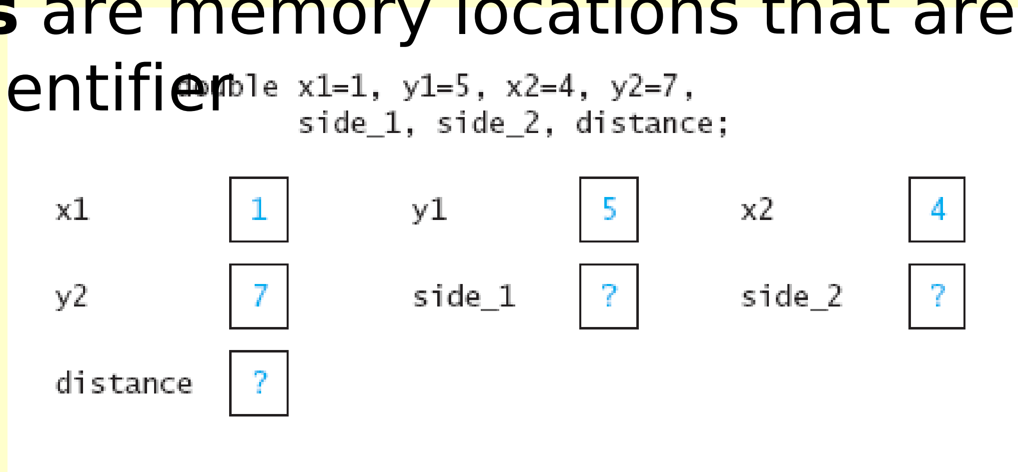
```
    //  Exit program
    return 0;
}
```

•This statement return a value of 0 to the OS.

•The body of the main function then ends with the **right brace** on a line by itself

Constants and Variables

- **Constants** are specific values that we include in C programs
- **Variables** are memory locations that are assigned a name or identifier



- Valid variable names must:
  - Begin with an alphabetic character
  - Can contain letters but not as the first character
  - Can be of any length, but the first 31 characters must

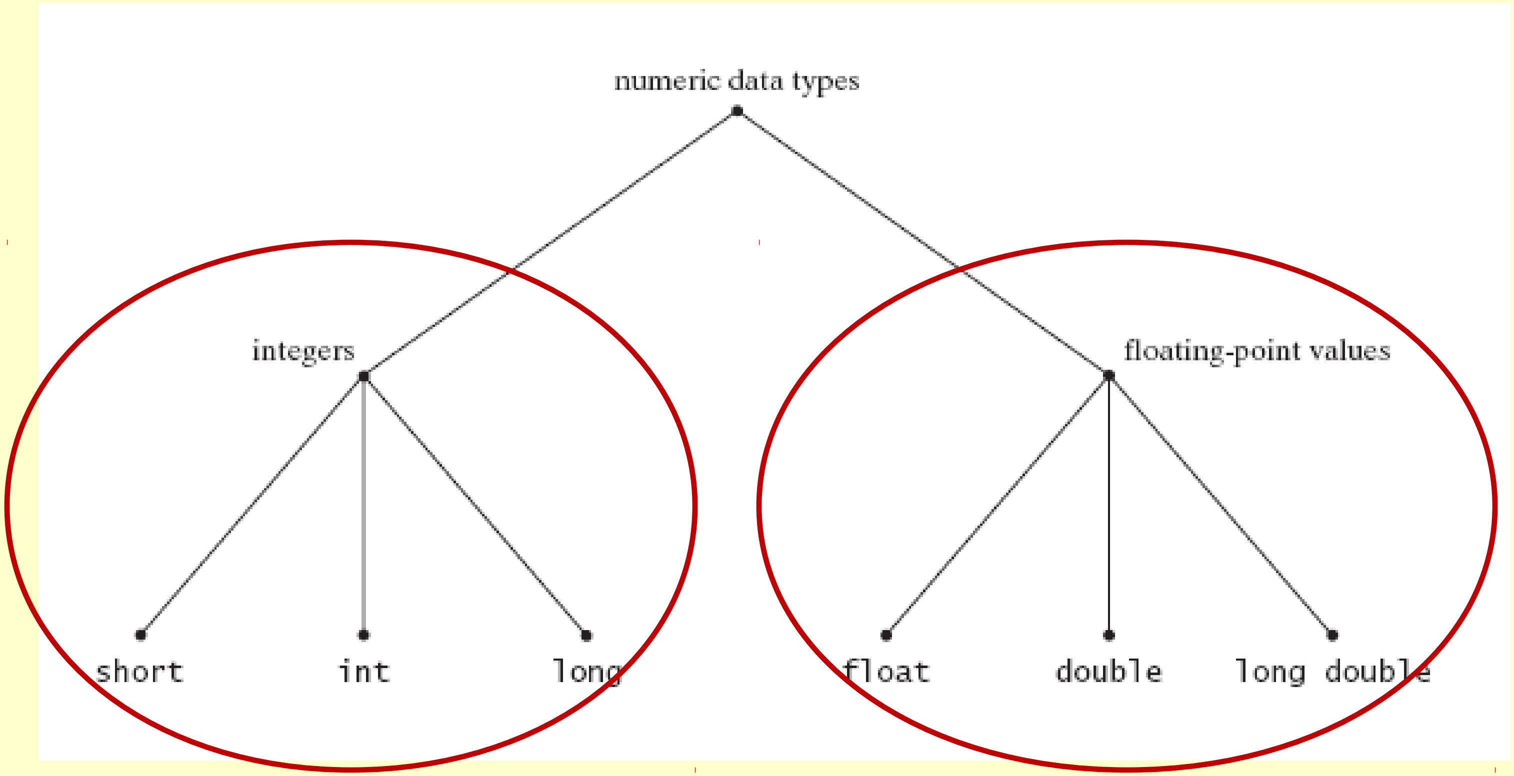
Keywords

•C also includes **keywords** with special meaning to the C compiler that <u>cannot</u> be used for identifiers

auto	double	ints	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Numeric Data Types

•Numeric data types are used to specify the types of numbers that will be contained in variables



Numeric Data Types

•The type specifiers for floating-point values are float (single precision), double (double precision), and long double (extended precision).

•The following statement in our sample program defines seven variables that are double-precision floating-point values:

```
//  Declare and initialize variables
double x1=1, y1=5, x2=4, y2=7, side1, side2, distance;
```

Example Data-Type Limits

•The difference between the float and double, and long double types relate to the precision (accuracy) and range of the values represented

Integers	
`short`	Maximum = 32,767
`int`	Maximum = 2,147,483,647
`long`	Maximum = 2,147,483,647
Floating Point	
`float`	6 digits of precision Maximum exponent 38 Maximum value 3.402823e+38
`double`	15 digits of precision Maximum exponent 308 Maximum value 1.797693e+308
`long double`	15 digits of precision Maximum exponent 308 Maximum value 1.797693e+308

*Microsoft Visual C++ 6.0 compiler.

Character Data

•Character data is a type of information used to represent and manipulate characters

Character	ASCII Code	Integer Equivalent
newline, \n	0001010	10
%	0100101	37
3	0110011	51
A	1000001	65
a	1100001	97
b	1100010	98
c	1100011	99

Symbolic Constants

•A symbolic constant is defined with a preprocessor directive that assigns an identifier to the constant

•A directive can appear anywhere in a C program; the compiler will replace each occurrence of the directive identifier with the constant value.   Examples of these are:

```
#define PI 3.141516
```

•Statements that need to use the value of π then would use the symbolic constant PI:

```
Area = PI*radius*radius
```

Arithmetic Operators

- An assignment statement can be used to assign the results of an arithmetic operator to a variable

```
Area_square = side*side
```

- Where * indicates multiplication. The symbols + and – are used to indicate addition and subtraction, respectively, and / is used for division

- The following are valid statements

```
Area_triangle = 0.5*base*height;
Area_triangle = (base*height)/2;
```

Priority of Arithmetic Operators

- In an expression that contains more than one arithmetic operator, we need to be concerned about the order in which the operations are performed

- The following table shows the precedence of arithmetic operators

Precedence	Operator	Associativity
1	Parentheses: ( )	Innermost first
2	Unary operators: + − (type)	Right to left
3	Binary operators: * / %	Left to right
4	Binary operators: + −	Left to right

Overflow and Underflow

- If the results of a computation exceeds the range of allowed values, an error occurs.

- For example, suppose we execute the following statements;

```
x = 2.5e30;
y = 1.0e30;
z = x*y;
```

- The values of x and y are within allowable range, but not z (which should be 2.5e60). This error is called **exponent overflow**

Increment and Decrement Operators

•The C language contains **unary operators** for incrementing and decrementing variables.

•For example:

```
x --;
y ++;
```

•The first statement decrements the variable x by 1, and the second statement increments the variable y by 1

•Other combinations are possible. For example x = x + 3 and x += 3 are equivalent statements

Priority of Arithmetic and Assignment Operators

Precedence	Operator	Associativity
1	Parentheses: ( )	Innermost first
2	Unary operators: + − ++ −− (type)	Right to left
3	Binary operators: * / %	Left to right
4	Binary operators: + −	Left to right
5	Assignment operators: = += −= *= /= %=	Right to left

Standard Input and Output

- To use input/output statements in a C program, we must include the following preprocessor directive:

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

- The **printf** statement function allows us to print values and text to the screen
-

```
printf("Angle=%f radians \n",angle);
```

- If the value of the angle is 2.84, the output generated by the previous statement will be

```
Angle=2.840000 radians
```

Specifiers for Output Statements

Variable Type	Output Type	Specifier
	Integer Values	
short, int	int	%i, %d
int	short	%hi, %hd
long	long	%li, %ld
int	unsigned int	%u
int	unsigned short	%hu
long	unsigned long	%lu
	Floating-Point Values	
float, double	double	%f, %e, %E, %g, %G
long double	long double	%LF, %Le, %LE, %Lg, %LG
	Character Values	
char	char	%c

- To print a short or an int, use an %i (integer) or %d (decimal)
- To print a float or a double, use an %f (floating-point), %e (exponential), or %E.

Standard Input and Output

- The **scanf** statement function allows us to enter values in to the program

```
scanf("%i",&year);
```

- If we wish to read more than one value from the keyboard, we can use the following statement

```
scanf(%lf %c",&distance,&unit_length);
```

- To read a double variable use %lf specifier

- To read a character type variable, then use %c

Specifiers for Input Statements

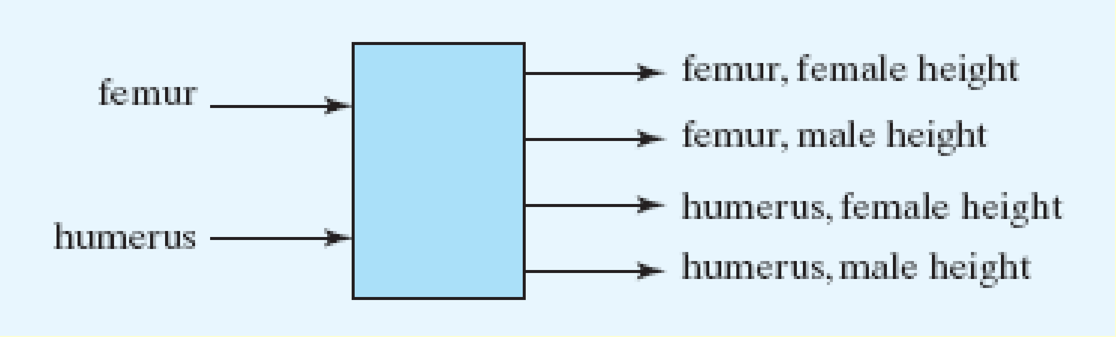
Variable Type	Specifier
Integer Values	
int	%i, %d
short	%hi, %hd
long int	%li, %ld
unsigned int	%u
unsigned short	%hu
unsigned long	%lu
Floating-Point Values	
float	%f, %e, %E, %g, %G
double	%lf, %le, %lE, %lg, %lG
long double	%Lf, %Le, %LE, %Lg, %LG
Character Values	
char	%c

Problem Solving Applied: Estimating Height from Bone Lengths

1.Problem Statement:

Estimate a person’s height from the length of the femur and from that of the humerus

2.Input/Output Description:



Problem Solving Applied: Estimating Height from Bone Lengths

3.Hand Example:

Suppose that the length o the femur is 15, and the length of the humerus is 12 in. The height estimates are:

femur_height_female=femur_lengthx1.94+28.7=57.8 in

humerus_height_female=humerus_lengthx2.8+28.2=61.8 in

Problem Solving Applied: Estimating Height from Bone Lengths

4. Algorithm Development:
   Decomposition Outline
   1. Read the lengths of the femur and humerus
   2. Compute the height estimates
   3. Print the height estimates

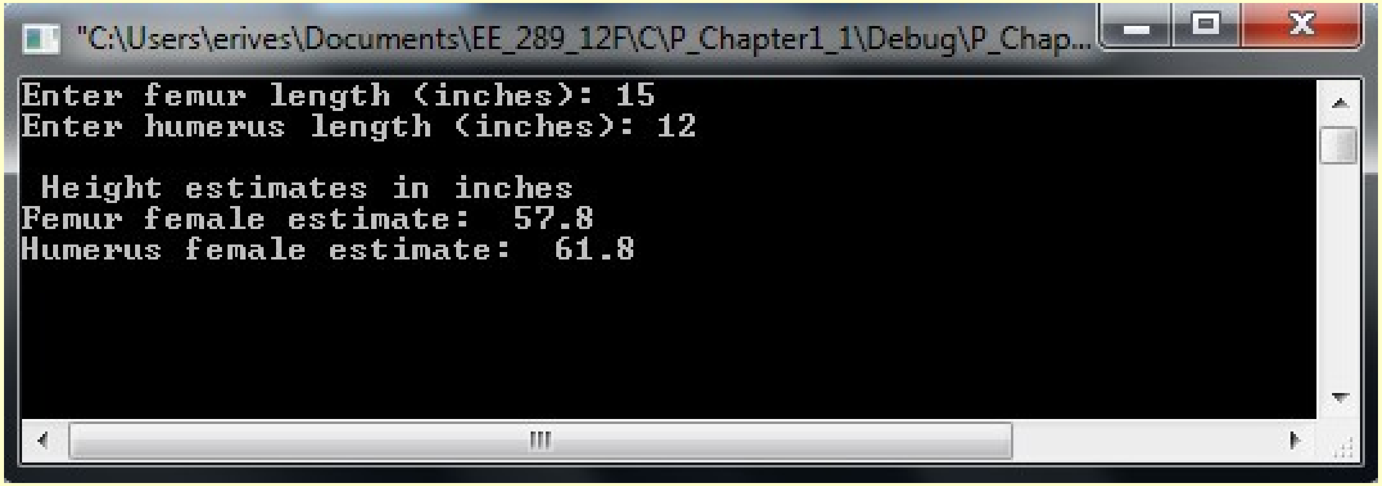
Problem Solving Applied: Estimating Height from Bone Lengths

```
/*
    Purpose: estimates a female height from length of femur and humerus
    Input(s): femur and humerus length
    Output(s): female height
    Written by: HE
    Date: 8/12
*/

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

int main(void)
{
   /*  Declare variables.  */
   double femur, humerus;

   /*  Get user input from the keyboard.  */
   printf("Enter femur length (inches): "); scanf("%lf",&femur);
   printf("Enter humerus length (inches): "); scanf("%lf",&humerus);

   /*  Print heights   */
   printf("\n Height estimates in inches \n");
   printf("Femur female estimate: %5.1f \n",femur*1.94+28.7;);
   printf("Humerus female estimate: %5.1f \n",humerus*2.8+28.2);

   /*  Exit program.  */
   return 0;
}
/*-----------------------------------------------------------*/
```

Problem Solving Applied: Estimating Height from Bone Lengths

5.Testing

We test the program and generates the following output



The answer matches the hand example

Mathematical Functions

- Arithmetic expressions that solve engineering problems often require computations other than additions, subtraction, multiplication, and division

- The following preprocessor directive should be used in program referencing mathematical functions

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

- The preprocessor should be used if for example one wants to compute a sine function

Mathematical Functions

- The following statement computes the sine of the angle theta which is <u>in radians</u>

```
b=sin(theta);
```

- A function reference can also be a part of the argument of another function reference (just like in MATLAB)

```
b=log(fabs(x));
```

- Where fabs(x) returns the absolute value of x, and log (x) returns the natural logarithm of x

Elementary Mathematical Functions

fabs(x) Absolute value of x
sqrt(x) Square root of x
pow(x,y) Computes the value of x to the y power
ceil(x) rounds x to the nearest integer toward ∞
floor(x) rounds x to the nearest integer toward -∞
exp(x) computes the value of exp(x)
log(x) return natural log of x
log10(x) returns log 10 of x

Trigonometric Functions

sin(x) sine of x in radians
cos(x) cosine of x in radians
tan(x) tangent of x in radians
asin(x) arcsine or inverse sine of x
acos(x) arcosine or inverse sine of x
atan(x) arctangent of x. The function returns an angle in radians
in the range [-π/2,π/2]
atan2(x) arctangent of x. The function returns an angle in radians
in the range [-π,π]

Character Comparisons

•The standard C library contains additional functions for use with characters

•The following preprocessor directive should be used in programs referencing these character functions:

#include <ctype.h>

•Some character functions include:

isdigit(ch) returns a nonzero value if ch is a decimal value

isalpha(ch) returns a nonzero value if ch ia an upper/lowercase letter

.

.

Homework on Chapter 2 is posted on the website:

http://www.ee.nmt.edu/~erives/289_F12/EE289.html

Homework is due in a week