BIL 104E Introduction to Scientific and Engineering Computing

Lecture 1

Introduction to C programming, Program Structure

Introduction

- **As engineers and scientists why do we need computers?**

  - We use computers to solve a variety of problems ranging from evaluation of simple functions to solving nonlinear equations.

- **Why C?**

  - C has powerful commands and data structures.

  - Can easily be used for system-level operations.

- **ANSI C**

  - C standards developed by the American National Standards Institute

  - Portable code that can be transferred from one computer platform to another.

Computing Systems

- A **computer** is a machine that performs operations that are specified with a set of instructions called a **program**.

- **Hardware** is the computer equipment, such as keyboard, mouse, terminal, hard disk, and printer.

- **Software** refers to the programs that describe the steps that we want the computer to perform.

- **Processor** is the part of the computer that controls all other parts.

- **Arithmetic Logic Unit (ALU)** performs arithmetic calculations and logic decisions.

- The processor and ALU together are called the **Central Processing Unit (CPU)**.

- **Operating System** provides an interface between the user and the hardware by providing a convenient and efficient environment in which the user can select and execute the software on the system.

Computer Languages

- **Machine language** is written using two symbols which are usually represented using the digits 0 and 1 (binary). Machine language is tied closely to the design of the computer hardware so the machine language for a Sun computer is different from the machine language for a Silicon Graphics computer.

- **Assembly language** is also unique to a specific computer design. But instructions are written in English-like statements instead of binary. Writing programs can be tedious.

- **High level languages** have English-like commands and instructions. Include C, FORTRAN etc. Writing programs is easier.

- Although C is defined as a high level language, many people like to describe C as a **mid-level language** because it allows access to low level routines and is often used to define programs that are converted to assembly language.

C Programming Language

- C evolved from two languages BCPL and B. In 1972, Dennis Ritchie developed and implemented the first C compiler at Bell Laboratories.

- In 1989 ANSI C standard was approved which provided a machine independent and unambiguous definition of C.

General Form

```
preprocessing directives
main()
{
  declarations;
  statements;
}
```

Examples of C Programs: Our First Program

```
1   /* A first program in C */
2   #include <stdio.h>
3
4   int main()
5   {
6       printf( "Welcome to C!\n" );
7
8       return 0;
9   }
```

- **Comments**
  - Text surrounded by /* and */ is ignored by computer
  - Used to describe the program

- **#include <stdio.h>**
  - Preprocessor directive
  - Tells computer to load contents of a certain file
  - **<stdio.h>** allows standard input/output operations

- **int main()**
  - C programs contain one or more functions, exactly one of which must be **main**
  - Parenthesis used to indicate a function
  - **int** means that main "returns" an integer value
  - Braces **({** and **})** indicate a block. The bodies of all functions must be contained in braces

- **printf("Welcome to C!\n");**
  - Instructs computer to perform an action. Specifically, prints the string of characters within quotes **(" ")**
  - Entire line called a **statement**. All statements must end with a semicolon **(;)**
  - Escape character **(\)** iindicates that **printf** should do something out of the ordinary. **\n** is the **newline** character

- **return 0;**
  - A way to exit a function. **return 0**, in this case, means that the program terminated normally

- **Right brace }**
  - Indicates end of **main** has been reached

Examples of C Programs: Our First Program

```
1   /* A first program in C */
2   #include <stdio.h>
3
4   int main()
5   {
6       printf( "Welcome to C!\n" );
7
8       return 0;
9   }
```

- **Comments**
  - Text surrounded by /* and */ is ignored by computer
  - Used to describe the program

- **#include <stdio.h>**
  - Preprocessor directive
  - Tells computer to load contents of a certain file
  - **<stdio.h>** allows standard input/output operations

- **int main()**
  - C programs contain one or more functions, exactly one of which must be **main**
  - Parenthesis used to indicate a function
  - **int** means that main "returns" an integer value
  - Braces **({** and **})** indicate a block. The bodies of all functions must be contained in braces

- **printf("Welcome to C!\n");**
  - Instructs computer to perform an action. Specifically, prints the string of characters within quotes **(" ")**
  - Entire line called a **statement**. All statements must end with a semicolon **(;)**
  - Escape character **(\)** iindicates that **printf** should do something out of the ordinary. **\n** is the **newline** character

- **return 0;**
  - A way to exit a function. **return 0**, in this case, means that the program terminated normally

- **Right brace }**
  - Indicates end of **main** has been reached

Examples of C Programs: Our First Program

```
1   /* A first program in C */
2   #include <stdio.h>
3
4   int main()
5   {
6       printf( "Welcome to C!\n" );
7
8       return 0;
9   }
```

- **Comments**
  - Text surrounded by /* and */ is ignored by computer
  - Used to describe the program

- **#include <stdio.h>**
  - Preprocessor directive
  - Tells computer to load contents of a certain file
  - **<stdio.h>** allows standard input/output operations

- **int main()**
  - C programs contain one or more functions, exactly one of which must be **main**
  - Parenthesis used to indicate a function
  - **int** means that main "returns" an integer value
  - Braces **({** and **})** indicate a block. The bodies of all functions must be contained in braces

- **printf("Welcome to C!\n");**
  - Instructs computer to perform an action. Specifically, prints the string of characters within quotes **(" ")**
  - Entire line called a **statement**. All statements must end with a semicolon **(;)**
  - Escape character **(\)** iindicates that **printf** should do something out of the ordinary. **\n** is the **newline** character

- **return 0;**
  - A way to exit a function. **return 0**, in this case, means that the program terminated normally

- **Right brace }**
  - Indicates end of **main** has been reached

Examples of C Programs: Our First Program

```
1   /* A first program in C */
2   #include <stdio.h>
3
4   int main()
5   {
6       printf( "Welcome to C!\n" );
7
8       return 0;
9   }
```

- **Comments**
  - Text surrounded by /* and */ is ignored by computer
  - Used to describe the program

- **#include <stdio.h>**
  - Preprocessor directive
  - Tells computer to load contents of a certain file
  - **<stdio.h>** allows standard input/output operations

- **int main()**
  - C programs contain one or more functions, exactly one of which must be **main**
  - Parenthesis used to indicate a function
  - **int** means that main "returns" an integer value
  - Braces (**{** and **}**) indicate a block. The bodies of all functions must be contained in braces

- **printf("Welcome to C!\n");**
  - Instructs computer to perform an action. Specifically, prints the string of characters within quotes (**" "**)
  - Entire line called a **statement**. All statements must end with a semicolon (**;**)
  - Escape character (****) iindicates that **printf** should do something out of the ordinary. **\n** is the **newline** character

- **return 0;**
  - A way to exit a function. **return 0**, in this case, means that the program terminated normally

- **Right brace }**
  - Indicates end of **main** has been reached

Examples of C Programs: Our First Program

```
1   /* A first program in C */
2   #include <stdio.h>
3
4   int main()
5   {
6       printf( "Welcome to C!\n" );
7
8       return 0;
9   }
```

- **Comments**
  - Text surrounded by /* and */ is ignored by computer
  - Used to describe the program

- **#include <stdio.h>**
  - Preprocessor directive
  - Tells computer to load contents of a certain file
  - **<stdio.h>** allows standard input/output operations

- **int main()**
  - C programs contain one or more functions, exactly one of which must be **main**
  - Parenthesis used to indicate a function
  - **int** means that main "returns" an integer value
  - Braces (**{** and **}**) indicate a block. The bodies of all functions must be contained in braces

- **printf("Welcome to C!\n");**
  - Instructs computer to perform an action. Specifically, prints the string of characters within quotes (**" "**)
  - Entire line called a **statement**. All statements must end with a semicolon (**;**)
  - Escape character (****) iindicates that **printf** should do something out of the ordinary. **\n** is the **newline** character

- **return 0;**
  - A way to exit a function. **return 0**, in this case, means that the program terminated normally

- **Right brace }**
  - Indicates end of **main** has been reached

Examples of C Programs: Our First Program

```
1   /* A first program in C */
2   #include <stdio.h>
3
4   int main()
5   {
6      printf( "Welcome to C!\n" );
7
8      return 0;
9   }
```

- **Comments**
  - Text surrounded by /* and */ is ignored by computer
  - Used to describe the program

- **#include <stdio.h>**
  - Preprocessor directive
  - Tells computer to load contents of a certain file
  - **<stdio.h>** allows standard input/output operations

- **int main()**
  - C programs contain one or more functions, exactly one of which must be **main**
  - Parenthesis used to indicate a function
  - **int** means that main "returns" an integer value
  - Braces **({** and **})** indicate a block. The bodies of all functions must be contained in braces

- **printf("Welcome to C!\n");**
  - Instructs computer to perform an action. Specifically, prints the string of characters within quotes **(" ")**
  - Entire line called a **statement**. All statements must end with a semicolon **(;)**
  - Escape character **(\)** iindicates that **printf** should do something out of the ordinary. **\n** is the **newline** character

- **return 0;**
  - A way to exit a function. **return 0**, in this case, means that the program terminated normally

- **Right brace }**
  - Indicates end of **main** has been reached

Examples of C Programs: Our First Program

```
1
2  /* A first program in C */
3  #include <stdio.h>
4
5  int main()
6  {
7     printf( "Welcome to C!\n" );
8
9     return 0;
10 }
```

- Save your code with a suitable name, such as *file1.c*

- Compile your code and obtain the executable program and then run it
  1. Windows: Use Visual C, Turbo C, Borland C, **Dev-C++** etc.
  2. Linux: use cc or gcc
     > **gcc** file1.c **–o** file1.x
     > **.**/file1.x

Compiling, Linking, Executing

- A program written in a high-level language must be translated into machine language by using a **compiler**, before the instructions can be executed by the computer.

- If any errors are detected by the compiler during compilation, corresponding error messages are printed. The errors identified at this stage are called **compile errors**. The program statements should be corrected and the compilation step should be performed again (**debugging**).

- The original C program is referred to as the **source program**, and the machine language version is called an **object program**.

- Before **execution** the object program should be **linked** to other machine language statements and the program should be **loaded** to the memory.

- **Execution errors** or **logic errors** may be identified at this stage. Execution errors often cause termination of a program. Some execution errors do not stop the program but cause incorrect results to be computed.

- Many C programming environments include **debugger** programs which allow the programmer to see values stored in variables at different points in a program and to step through the program line by line.

An example using math.h file

```
1   /* Third program in C */
2   #include <stdio.h>
3   #include <stdlib.h>
4   #include <math.h>
5   main()
6   {
7       float x=125;
8       double y;
9       y=pow(x,3);
10      printf("y=%f\n",y);
11      exit(0);
12  }
```

- Compile your code by linking math library
  > gcc –lm file3.c –o file3.x
  > ./file3.x

Problem Solving Methodology

1. **State the problem clearly:** Should be clear and avoid any misunderstanding.

2. **Describe the input and output information:** Information that is given and the values to be computed.

3. Work a simple example by hand

4. **Develop an algorithm and convert it to a computer program:** Algorithm is a step-by-step outline of the problem solution, in other words simple operations performed one after another.

5. **Test the solution with a variety of data:** Data from the hand example can be used for testing.

Example

- **Problem statement**:

  "*Compute the distance between two points in a plane.*"
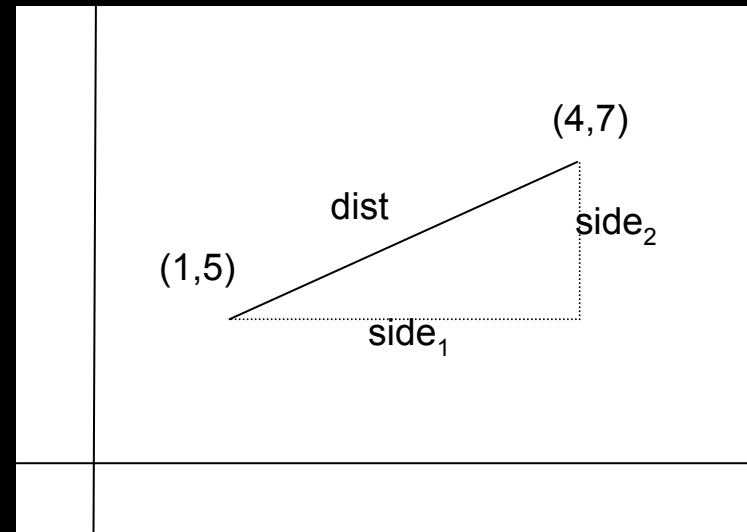
- **Input/Output Description :**
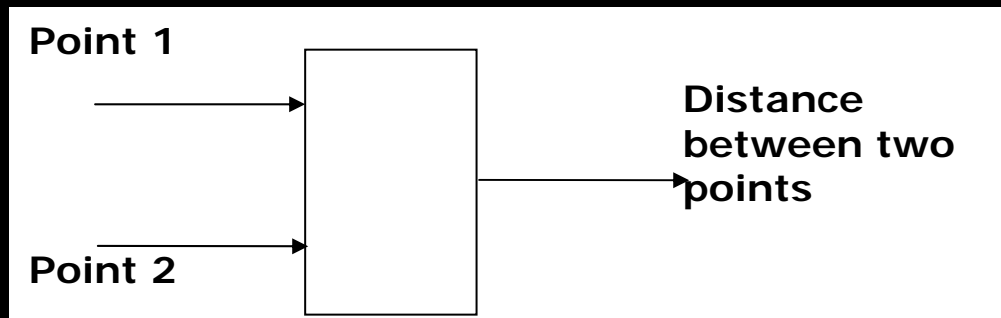
Example (cont.)

- **Hand Example** :

$$p_1 = (1,5)$$
$$p_2 = (4,7)$$
$$dist = \sqrt{(side_1)^2 + (side_2)^2}$$
$$= \sqrt{(4-1)^2 + (7-5)^2}$$
$$= \sqrt{13}$$
$$= 3.61$$

Example (cont.)

- **Algorithm**:

  1. Give values to the points.

  2. Compute the lengths of the sides of the right triangle. (side1 and side2).

  3. Compute the hypotenuse of the tringle which is also the distance between the two points. (dist)

  4. Print the value of distance.

Example (cont.)

```c
/*--This program computes the distance between two points-----*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
main()
{
  /*----- Declare and initialize variables----- */
  double x1=1, y1=5, x2=4, y2=7;
  double side1, side2, dist ;

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

  /*----- Print distance on the screen--------- */
  printf("The distance between the two points is %5.2f \n",dist);

/*----- Exit program ----------------------- */
return EXIT_SUCCESS;
}
```

Example (cont.)

- **Test** :

  The output of the program is:

  ```
  The distance between the two points is 3.61
  ```

Program Structure

- **Comments**: Begin with /* and end with */ . Can be extended over several lines.

- **Preprocessor directives**: Give instructions to the compiler that are performed before the program is compiled.

  - **#include** is the most common directive and it inserts additional statements in the program.

  - **#include** is followed by the name of the file containing the additional statements.

  - **<** and **>** indicate that the files are included with the **Standard C library**. This library is contained in the files that accompany an ANSI C compiler.

  - **stdio.h** contains info related to input and output statements.
  - **stdlib.h** contains the constant that we used when exiting the program.
  - **math.h** contains information related to the square root function.
  - **.h** extension on the file names specifies that they are header files.

Program Structure

- **main function**: Every C program should contain a main function. The body of the function is enclosed by braces '{' and '}'.

- **Declarations**:
    - Define the variables that will be used by statements.
    - Must end with a semicolon.
    - They must precede the statements.
    - May or may not give initial values.
    - If initialization is not made than the values <u>should not</u> be assumed as zero.
    - double indicates that variables will store double-precision floating-point values.

- **Statements**:
    - Specify the operations to be performed, such as arithmetic computations, function calls.
    - Must end with a semicolon.
    - printf statement is used to print output to the screen.
    - return statement is used to exit the program.
    - EXIT_SUCCESS constant, defined in stdlib.h, indicates a successful exit from the program.

Style

- Comments are optional but use them to improve the readability of your program.

- Too long lines can be spitted and by indenting the continuation of the remaining lines can be indicated.

- Use blank lines to make a program more readable and easier to modify.