

# Computer Programming

1

# 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.

# 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.

# Problem Solving Methodology

1. State the problem clearly
2. Describe the input and output information
3. Work a simple example by hand
4. Develop an algorithm and convert it to a computer program
5. Test the solution with a variety of data

# Example

## Problem statement:

- Should be clear and avoid any misunderstanding.

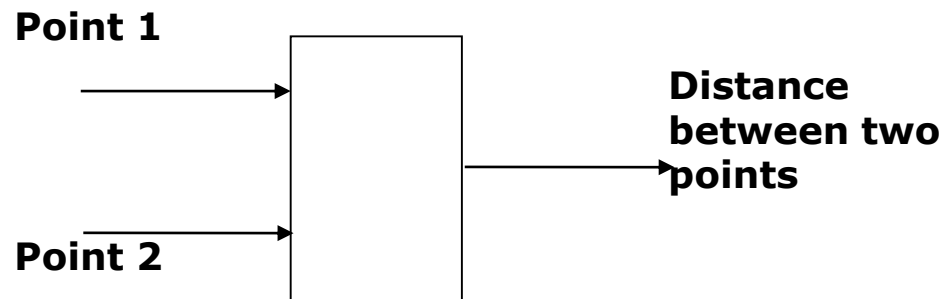
*“Compute the distance between two points in a plane.”*



# Example

## Input/Output Description :

- Information that is given and the values to be computed.



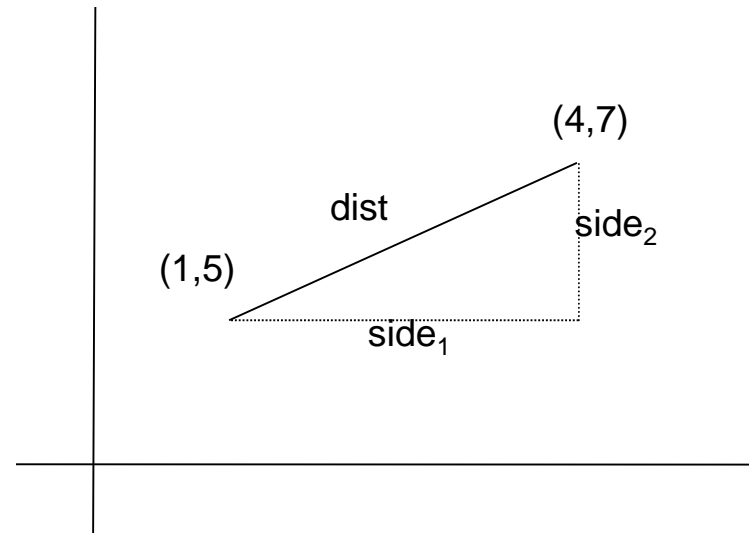
# Example

## Hand Example :

$$p_1 = (1,5)$$

$$p_2 = (4,7)$$

$$\begin{aligned} dist &= \sqrt{(side_1)^2 + (side_2)^2} \\ &= \sqrt{(4-1)^2 + (7-5)^2} \\ &= \sqrt{13} \\ &= 3.61 \end{aligned}$$



# Example

## Algorithm Development :

- Algorithm is a step-by-step outline of the problem solution, in other words simple operations performed one after another.
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 triangle which is also the distance between the two points. (dist)
  4. Print the value of distance.

# Example

```
/*--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

## Testing :

- Data from the hand example can be used for testing.

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 then the values should not 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 splited 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.



# General Form

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

# *Constants and Variables*

- Constants and variables represent values that we use in our programs.
- **Constants** are specific values such as 2, 3.14, or -5.78.
- **Variables** are memory locations that are assigned a **name** or **identifier**.
- **Identifier** is used to reference the value stored in the memory location.
- The values of variables that were not given initial values are unspecified. These values are called **garbage values** because they are values left in memory from previous program.

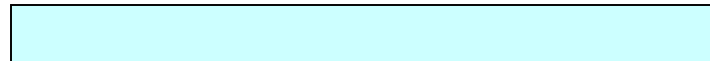
# Rules for selecting a valid identifier

- Begin with an **alphabetic character** or the underscore character **\_**.
- Alphabetical characters can be lowercase or uppercase. (C is **case sensitive**)
- **Digits** can be used but not as the first character.
- Can be of any length but first **31 characters** should be unique.
- **Keywords** with special meanings to the compiler should not be used.
- The name should reflect the **content** of the variable.

# 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  }
```

Output:



- **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 **(\)** indicates 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 }
```

Output:

- **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 (`\`) indicates 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 }
```

Output:

- **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 (`\`) indicates 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  }
```

Output:

- **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 (`\`) indicates 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  }
```

Output:

- **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 **(\)** indicates 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  }
```

Output:

```
Welcome to C!
```

- **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 **(\)** indicates 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**

# 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

# Numeric Data Types

- In C, numeric values are either integers or floating-point values. There are also non-numeric data types (such as characters) which will be discussed later.
- **Integers**: Specified by **short**, **int** and **long** according to the required range. Ranges of values are **system dependent**.
- C also allows **unsigned** qualifier where unsigned integer represents only positive values. Signed and unsigned integers represent same number of values but the ranges are different.

# For most systems ranges are:

INTEGERS	Min	Max
short	-32768	32767
int	-32768	32767
long	-2147483648	-2147483647
unsigned short	0	65535

# Numeric Data Types

## Floating point numbers:

Specified by **float** (single-precision), **double** (double-precision), and **long double** (extended precision) according to the required precision and range which are also system dependent.

# Floating point numbers

<b>FLOATING POINT NUMBERS</b>	<b>Precision</b>	<b>Max Exponent</b>	<b>Maximum Value</b>
<b>float</b>	6 digits	38	3.402823e+38
<b>double</b>	15 digits	308	1.797693e+308
<b>long double</b>	19 digits	4932	1.189731e+4932

# ***Specifiers for Output***

	<b>Variable Type</b>	<b>Output Type</b>	<b>Specifier for output</b>
<b>INTEGER VALUES</b>	short, int	int	<b>%i</b> (integer) , <b>%d</b> (decimal)
	int	short	<b>%hi</b> , <b>%hd</b>
	long	long	<b>%li</b> , <b>%ld</b>
	int	unsigned int	<b>%u</b>
	int	unsigned short	<b>%hu</b>
	long	unsigned long	<b>%lu</b>
<b>FLOATING-POINT VALUES</b>	float, double	double	<b>%f</b> (floating-point), <b>%e</b> (exponential form), <b>%E</b> (exponential form) , <b>%g</b> (general), <b>%G</b> (general)
	long double	long double	<b>%Lf</b> , <b>%Le</b> , <b>%LE</b> , <b>%Lg</b> , <b>%LG</b>



# ***Specifiers for Input***

<b>Variable Type</b>	<b>Specifier of Input</b>
int	%i , %d
short	%hi, %hd
long int	%li, %ld
unsigned int	%u
unsigned short	%hu
unsigned long	%lu
float	%f, %e, %E, %g, %G
double	%lf, %le, %lE, %lg, %lG
long double	%Lf, %Le, %LE, %Lg, %LG

# Character Data

- **Numeric information** is represented in a C program as **integers** or **floating-point** values. Numeric values are often used in arithmetic computations.
- **Nonnumeric information** may consist of **alphabetic characters**, **digits**, and **special characters**.
- Each character corresponds to a **binary code** value. The most commonly used binary codes are **ASCII** (American Standard Code for Information Interchange) and **EBCDIC** (Extended Binary Coded Decimal Interchange Code).
- A total of 128 characters can be represented in the ASCII code.

# *char Data Type*

Character	ASCII Code	Integer Equivalent
newline, \n	0001010	10
%	0100101	37
3	0110011	51
A	1000001	65

- Note that the binary representation for a character digit is not equal to the binary representation for an integer digit.
- Nonnumeric information can be represented by constants or by variables.
  - A character constant is enclosed in single quotes, as in '**A**' , '**b**' , and '**3**' .
  - A variable that is going to contain a character can be defined as an integer or as a character data type (**char**).

# *getc() and getchar()*

- The **int getc(FILE \*stream)** function reads the next character from a file stream, and returns the integer value of the character as the function value.
- The **int getchar(void)** function reads a character from the standard input and returns the integer value of the character as the function value. It is equivalent to getc(stdin).
- Example:

```
#include <stdio.h>
main()
{
    int ch1, ch2;
    printf("Enter two characters from the keyboard:\n ");
    ch1=getc(stdin);
    ch2=getchar();
    printf("The first character you entered is: %c\n",ch1);
    printf("The second character you entered is: %c\n",ch2);
    return 0;
}
```

# *putc() and putchar()*

- The **int putc(int c, FILE \*stream)** function prints the character that corresponds to the integer argument to the specified file stream. It then returns the same character as the function value.
- The **int putchar(int)** function prints the character that corresponds to the integer argument to the computer screen. It then returns the same character as the function value.
- Example:

```
#include <stdio.h>
main()
{
    int ch1=65, ch2=98;
    printf("The character that has numeric value of %d is:
",ch1);
    putc(ch1,stdout); putc('\n',stdout);
    printf("The character that has numeric value of %d is:
",ch2);
    putchar(ch2); putchar('\n');
    return 0;
}
```