LECTURE – 1 & 2

Programming language and structure programming

- In simple word A **programming language** is a type of written **language** that tells computers what to do in order to work. There are multiple types, technique, levels and generations of programming language.

- **Structure programming** is a technique for organizing and coding computer programs . It allows a program to be spilitted into multiple blocks of execution. It enables a programmer to understand a program and detect the errors of a program easily. Which saves a lots of time.

- ALGOL, Pascal, C, etc. can be considered as Structured programming language.

C programming

- **C** is called **structured programming language** because a program in **c language** can be divided into small logical functional modules or structures with the help of function procedure. So, it is easy to understand and write a program with it.
- It is **procedure-oriented** programming language.
- C **is mother language** of all programming language.
- Previously, C was considered as the high level language but today many programmers consider it as a low level language as it supports only scalar operations.

Analyzing C Program Structure

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

void main(){
        clrscr();
        printf("This is my first C program \n");
        getch();
}
```

Preprocessor directives and Header File

As mentioned earlier, C programs are divided into modules or functions. Some functions are written by users and many others are stored in C library.

Library functions are grouped category-wise and stored in different files known as header files. If we want to access the functions stored in library, it is necessary to tell the compiler about the files to be accessed. This is achieved by using the **preprocessor directive.**

Preprocessor directives are lines included in a program that begin with the character #, which make them different from a typical source code text. They are invoked by the compiler to process some programs before compilation.

#include, #define, #ifndef etc. are the preprocessor directives in C. Pre processor directives are placed at the beginning of a program.

Preprocessor directives and Header File

```
#include <file_name>
```

Here file name is the name of the **library file** that contains the required function definitions.

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

Here **stdio.h** refers to the **standerd I/O header file** containing standard input and output functions like **printf()**, **scanf()** etc. and **#include** preprocessor directive says that this header file has been included to the program. To compile *library function* **printf(),** it is required.

Preprocessor directives and Header File

```
#include <conio.h>
```

Here **conio**. **h** is a C header file used mostly by MS-DOS compilers to provide console input/output. It **stands** for console input output header file.

clrscr(), getch(),getche(),textcolor() etc. library functions are included in this header file.

```
clrscr()
```

- This function is used to clear the output screen.

Preprocessor directives and Header File

`getch()`

- It reads character from keyboard.
- When the program compiles and provides an output, it waits for getting a character from the keyboard
- When it gets, it returns from the output screen to source code

Both had been used in the given program. For this the program's output screen will be cleared after it reads a character.

void main(){….}

main() is a special kind of function used by the C system to tell the computer where the program should start. Every C program have to have exactly one main function. If we use more than one main function the compiler won't understand where to mark the beginning of the program.

The empty pair of **parenthesis()** immediately following the main indicates that the function main does not have any arguments or parameters.

void before main indicates this function does not have any return type. Though the main function might have return types.

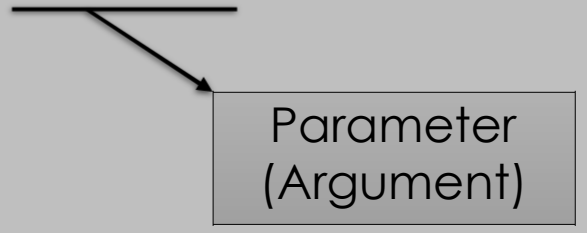
The { indicates the **beginning** of the main function and the } indicates the **end** of the function. Everything between these two braces is called the **body** of the function.

Statements and Arguments

- A computer program is made up of a series of **statements**. In C a **statement** is a command given to the computer that instructs the computer to take a specific action, such as display to the screen, or collect input. A statement is terminated by a semicolon ';'.

- **C** functions exchange information by means of **parameters** and **arguments**. The term **parameter** refers to any declaration within the parentheses following the function name in a function declaration or **definition**; the term **argument** refers to any expression within the parentheses of a function call.

  - Function_name(int a, int b)



printf("This is my first C program \n");

- *It's a C statement.*
- This statement has a **library function** called printf( ).
- printf() is a pre-defined standard C function that **writes the outputs** from stdin(standard
- Output) stream.
- printf() takes a *string* inside of it within two ***quotation marks*.**
- Whatever is in between them, will be printed on the screen.
- The ***backslash (\)*** character is called ***Escape Character.***
- The escape sequence **\n** means ***new line.***

```
/* This program takes two integers
and provides the sum to the user */

#include <stdio.h>     // header file
#include <conio.h>     //header file

// main function starts

void main(){
  clrscr();     // clearing the output screen
  int a,b,sum;  // declaring three integers a, b and sum
  printf("Enter integer one: \n");    // printing on output
  scanf ("%d",&a);     //taking the input value into a
  printf("Enter integer two: \n");    // printing on output
  scanf ("%d",&b);     // taking the input value into b
  sum=a+b;      // taking their summation into sum
  printf("Sum is: %d\n", sum); // displaying the sum
  getch();      //waiting for a character to return
}
```

Comments

Sometimes in a program we need to write lines that won't be compiled but will help someone to understand the program. These lines are called comment in C. its makes the program more readable and easy to analyze the code. In **C** there are two **types of comments** :

- Single line **comment**: The characters // is use in single line commenting.
- Multi-line **comment**: The characters /* … … … */ is use in multi line commenting .

➢When C finds this character, it **omits** what is inside that and goes to the next instruction.

Comments

In the given program :

/* This program takes two integers
and provides the sum to the user */

This is a **multi-line** comment.

// main function starts

This and every other line starting with these character are called **single line** comments.

```
scanf ("%d",&a);
scanf ("%d",&b);
```

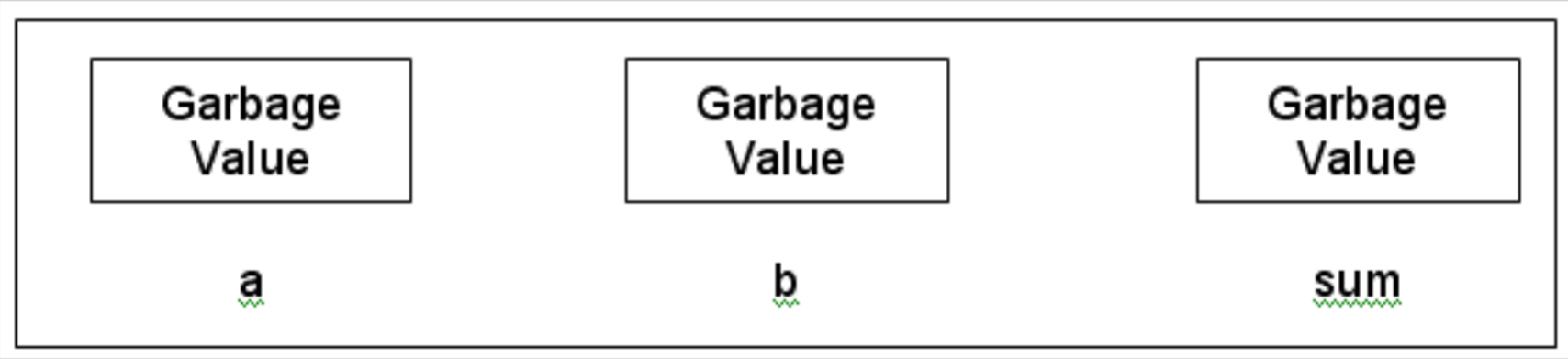
- scanf() is a pre-defined standard C function that **reads the inputs** from stdin(standard input) stream.
- scanf () is a library function to **obtain a value from the user**
- The scanf () here has two *arguments*. and **&*a*.**
- The **% is Escape Character** and **%d is Escape Sequence**
- The second argument **& (ampersand)-** called ***address operator*** in C- followed by the variable name- tells scanf() **the location in memory** in which the variable a is stored.
- The computer then **stores** the values for *a* at that location

```
sum=a+b;
```

- It calculates the sum of variables ***a* and *b*** and assigns the result to variable **sum** using the **assignment** operator **=**
- The **+** and **=** operators are called ***binary*** operators as they require two ***operands***
- The two operands of **+** are **a and b**
- The two operands of **=** are **sum and a+b**

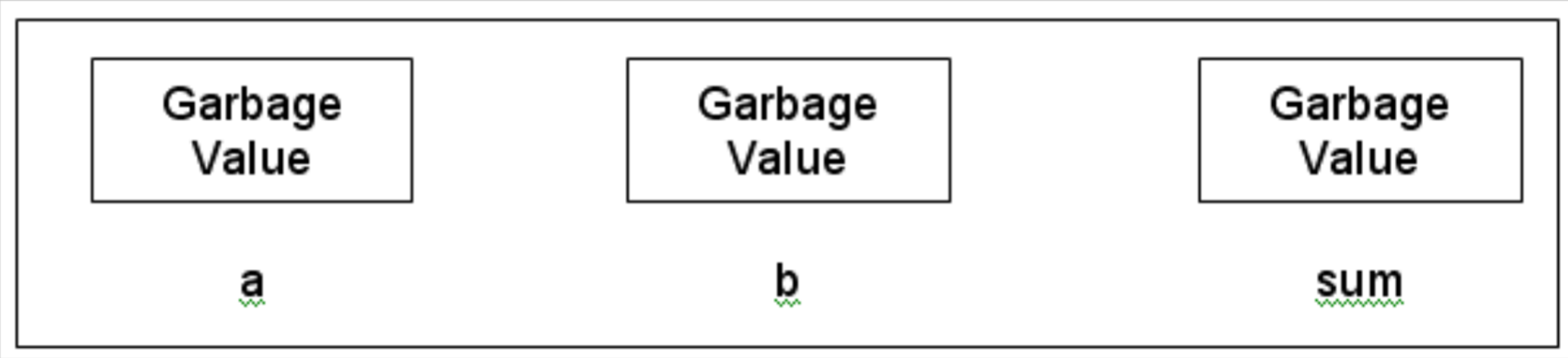
Memory Concept

- int a,b,sum;



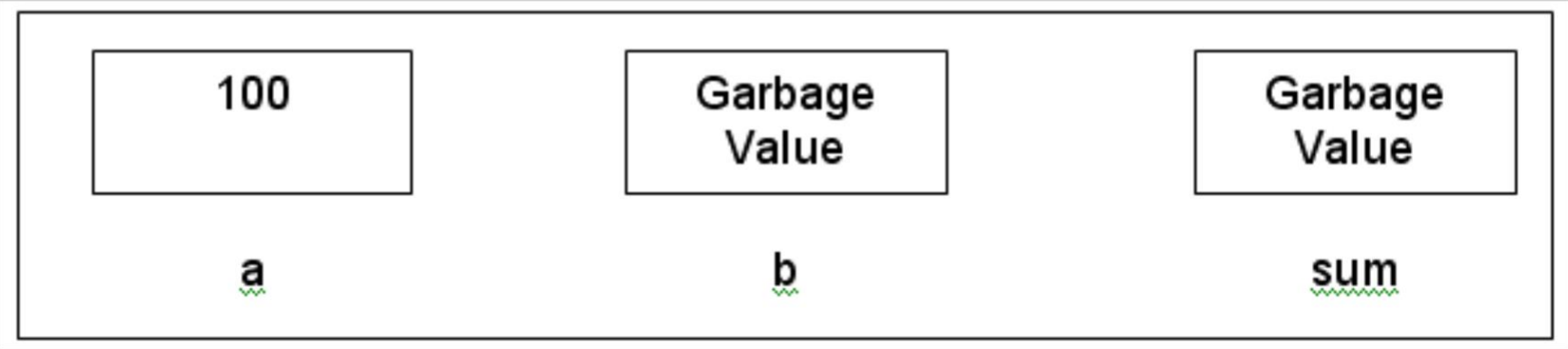
Memory Concept

- Scanf("%d",&a);



Memory Concept

- Scanf("%d",&a);
- User inputs a number. Suppose 100.

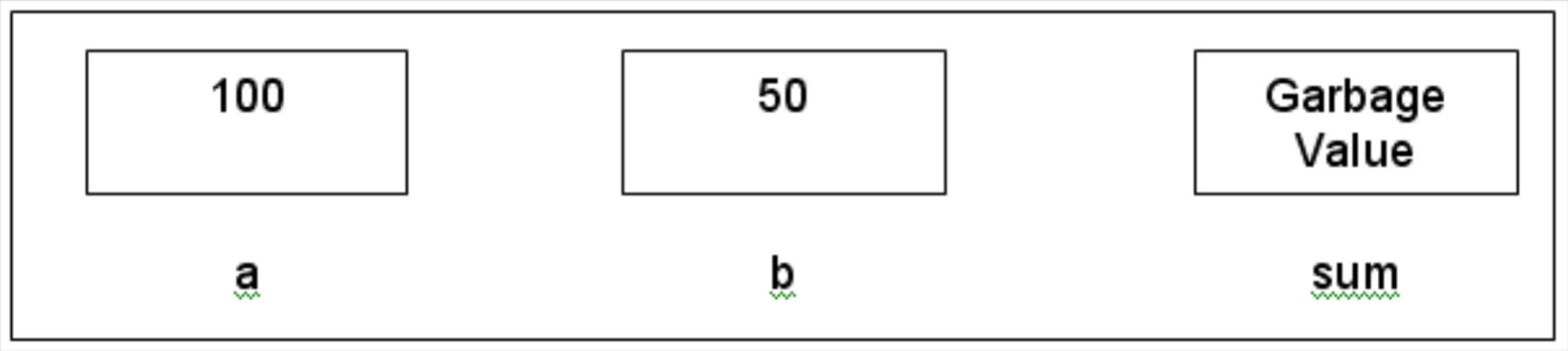


Memory Concept

- Scanf("%d",&b);

Memory Concept

- Scanf("%d",&b);
- User inputs a number. Suppose 50.



Memory Concept

- Sum = a + b;