# C PROGRAMMING

## Introduction to C Programming

C is a general-purpose, high-level language that was originally developed by Dennis M. Ritchie to develop the UNIX operating system at Bell Labs. C was originally first implemented on the DEC PDP-11 computer in 1972.

In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.

The UNIX operating system, the C compiler, and essentially all UNIX application programs have been written in C. C has now become a widely used professional language for various reasons −

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computer platforms

## Facts about C

- C was invented to write an operating system called UNIX.
- C is a successor of B language which was introduced around the early 1970s.
- The language was formalized in 1988 by the American National Standard Institute (ANSI).
- The UNIX OS was totally written in C.
- Today C is the most widely used and popular System Programming Language.
- Most of the state-of-the-art software have been implemented using C.
- Today's most popular Linux OS and RDBMS MySQL have been written in C.

## Why use C?

C was initially used for system development work, particularly the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as the code written in assembly language.

Some examples of the use of C might be −

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Databases
- Language Interpreters
- Utilities

**Example:/*Program to Display a message*/**

```
#include<stdio.h>
main()
{   clrscr();
    printf("Hello,World");
return(0);
}
```

Let us take a look at the various parts of the above program −

- The first line of the program *#include <stdio.h>* is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
- The next line *int main()* is the main function where the program execution begins.
- The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
- The next line *printf(...)* is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
- The next line **return 0;** terminates the main() function and returns the value

## C - Basic Syntax

You have seen the basic structure of a C program, so it will be easy to understand other basic building blocks of the

C programming language.

## Tokens in C:

```
printf("Hello, World! \n");
```

```
printf
(
"Hello, World! \n"
);
```

A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following C statement consists of five tokens −

The individual tokens are −

**Semicolons:**

```
printf("Hello, World! \n");
return 0;
```

In a C program, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

Given below are two different statements −

**Comments:**

```
/* my first program in C */
```

Comments are like helping text in your C program and they are ignored by the compiler. They start with /* and terminate with the characters */ as shown below −

You cannot have comments within comments and they do not occur within a string or character literals.

## Identifiers:

A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9).

```
mohd       zara    abc move_name a_123
myname50   _temp   j     a23b9     retVal
```

C does not allow punctuation characters such as @, $, and % within identifiers. C is a **case-sensitive** programming language. Thus, *Manpower* and *manpower*are two different identifiers in C. Here are some examples of acceptable identifiers −

**Keywords:**

The following list shows the reserved words in C. These reserved words may not be used as constants or variables or any other identifier names.

| | | | |
|---|---|---|---|
| Auto | else | Long | switch |
| Break | enum | register | typedef |
| Case | extern | return | union |
| Char | float | short | unsigned |
| Const | for | signed | void |
| Continue | goto | sizeof | volatile |
| Default | If | static | while |
| Do | int | struct | _Packed |
| Double | | | |

## White space in C

A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it.

Whitespace is the term used in C to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as

int, ends and the next element begins. Therefore, in the following statement −

```
int age;
```

there must be at least one whitespace character (usually a space) between int and age for the compiler to be able to

```
fruit = apples + oranges;  // get the total fruit
```

distinguish them. On the other hand, in the following statement −

no whitespace characters are necessary between fruit and =, or between = and apples, although you are free to include some if you wish to increase readability.

## C - Data Types

Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

## C - Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive. Based on the basic types explained in the previous chapter, there will be the following basic variable types −

| Type | Description |
|---|---|
| char | Typically a single octet(one byte). This is an integer type. |
| int | The most natural size of integer for the machine. |
| float | A single-precision floating point value. |
| double | A double-precision floating point value. |
| void | Represents the absence of type. |

C programming language also allows to define various other types of variables, which we will cover in subsequent chapters like Enumeration, Pointer, Array, Structure, Union, etc. For this chapter, let us study only basic variable types.

## Variable Definition in C

```
type variable_list;
```

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition

```
int    i, j, k;
char c, ch;
float  f, salary;
double d;
```

specifies a data type and contains a list of one or more variables of that type as follows −

Here, **type** must be a valid C data type including char, w_char, int, float, double, bool, or any user-defined object; and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here −

## C - Constants & Literals

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

## C - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators

## Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands. | A + B = 30 |
| − | Subtracts second operand from the first. | A − B = 10 |
| * | Multiplies both operands. | A * B = 200 |
| / | Divides numerator by de-numerator. | B / A = 2 |
| % | Modulus Operator and remainder of after an integer division. | B % A = 0 |
| ++ | Increment operator increases the integer value by one. | A++ = 11 |
| -- | Decrement operator decreases the integer value by one. | A-- = 9 |

## Relational Operators

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then −

| Operator | Description | Example |
|---|---|---|
| == | Checks if the values of two operands are equal or not. If yes, then the condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true. | (A <= B) is true. |

## Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

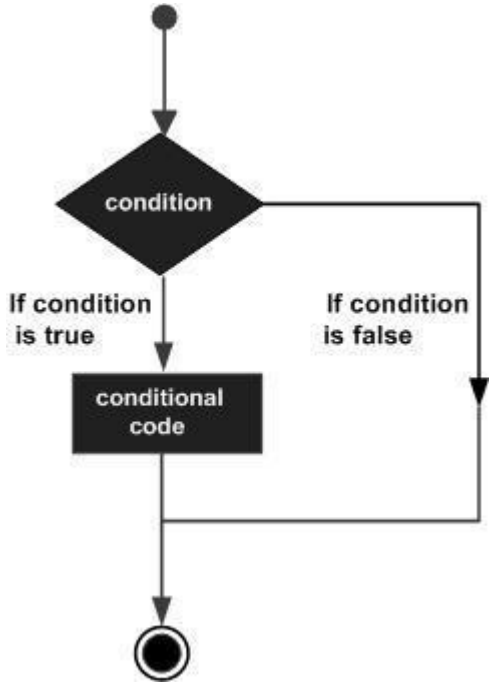
| Operator | Description | Example |
|---|---|---|
| && | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false. |
| \|\| | Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true. | (A \|\| B) is true. |
| ! | Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. | !(A && B) is true. |

**Assignment Operators:** The following table lists the assignment operators supported by the C language −

| Operator | Description | Example |
|---|---|---|

| | | |
|---|---|---|
| = | Simple assignment operator. Assigns values from right side operands to left side operand | C = A + B will assign the value of A + B to C |
| += | Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand. | C -= A is equivalent to C = C – A |
| *= | Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand. | C *= A is equivalent to C = C * A |

**C**




**- Decision Making**

Decision making structures require that the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Show below is the general form of a typical decision making structure found in most of the programming languages −

C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**,then it is assumed as **false** value.

C programming language provides the following types of decision making statements.

- Simple if statements
- If. Else statements
- Ladder if statements
- Nested if statements
- Switch statements

**The?: Operator**

```
Exp1? Exp2: Exp3;
```

We have covered **conditional operator? :** In the previous chapter which can be used to replace **if...else** statements.It has the following general form −

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

The value of a? Expression is determined like this −

- Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire? Expression.
- If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

**C - Loops**

You may encounter situations, when a block of code needs to be executed several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

C programming language provides the following types of looping statements

- While loop
- Do while loop
- For loop

**C - Functions**

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main ()**, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

The C standard library provides numerous built-in functions that your program can call. For example, **strcat ()** to concatenate two strings, **memcpy ()** to copy one memory location to another location, and

```
return_type function_name (parameter list)
   {body of the function
}
```

manymore functions.

A function can also be referred as a method or a sub-routine or a procedure, etc.

**Defining a Function**

The general form of a function definition in C programming language is as follows −

A function definition in C programming consists of a *function header* and a*function body*. Here are all the parts of a Function −

- **Return Type** − A function may return a value. The **return_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the

```
/* function returning the max between two numbers */
int max (int num1, int num2) {
   /* local variable declaration */
   int result;
   if (num1 > num2)
      result = num1;
   else
      result = num2;
   return result;
}
```

  return_type is the keyword **void**.
- **Function Name** − this is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters** − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** − the function body contains a collection of statements that define what the function does.

**Example**

Given below is the source code for a function called **max ()**. This function takes two parameters num1 and num2 and returns the maximum value between the two −

**Function Declarations**

A function **declaration** tells the compiler about a function name and how to call the function. The actual

```
return_type function_name (parameter list);
```

body of the function can be defined separately.

A function declaration has the following parts −

For the above defined function max (), the function declaration is as follows −

```
int max (int num1, int num2);
```

```
int max(int, int);
```

```
#include <stdio.h>
 /* function declaration  */
int max(int num1, int num2);
int main () {
   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;
   /* calling a function to get max value */
   ret = max(a, b);
   printf( "Max value is : %d\n", ret );
   return 0;}
/* function returning the max between two numbers */
int max(int num1, int num2) { /* local variable declaration */
   int result;
   if (num1 > num2)       result = num1;
   else       result = num2;     return result; }
```

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration −

Function declaration is required when you define a function in one source file and you call that function in another File. In such case, you should declare the function at the top of the file calling the function.

**Calling a Function**

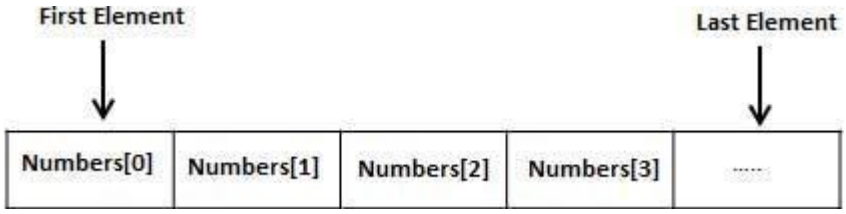
While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. For example −

**C - Arrays**

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.




Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

**Declaring Arrays**

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows −

```
type array Name [ array Size ];
```

This is called a *single-dimensional* array. The **array Size** must be an integer constant greater than zero and **type** can

Be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement −

```
double balance[10];
```

Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.

**Initializing Arrays**

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

You can initialize an array in C either one by one or using a single statement as follows −

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

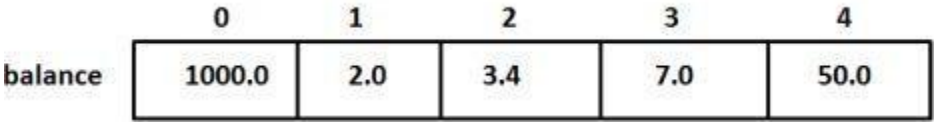
If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

```
balance[4] = 50.0;
```

You will create exactly the same array as you did in the previous example. Following is an example to assign a Single element of the array −

The above statement assigns the 5th element in the array with a value of 50.0. All arrays have 0 as the index of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above −




**Accessing Array Elements**

```
double salary = balance[9];
```

```
#include <stdio.h>
int main () {
   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;
   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = i + 100; /* set element at location i to i + 100 */
   }
   /* output each array element's value */
   for (j = 0; j < 10; j++) {
      printf ("Element[%d] = %d\n", j, n[j] );
   }
   return 0;
}
```

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

The above statement will take the 10th element from the array and assign the value to salary variable. The following Example shows how to use all the three above mentioned concepts viz. declaration, assignment, and accessing arrays −

When the above code is compiled and executed, it produces the following result −

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
```

```
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

```
#include <stdio.h>
int main () {
   int  var1;
   char var2[10];
   printf("Address of var1 variable: %x\n", &var1 );
   printf("Address of var2 variable: %x\n", &var2 );
   return 0;
}
```

**C - Pointers**

Pointers in C are easy and fun to learn. Some C programming tasks are performed more easily with pointers, and other tasks, such as dynamic memory allocation, cannot be performed without using pointers. So it becomes necessary to learn pointers to become a perfect C programmer. Let's start learning them in simple and easy steps.

As you know, every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory. Consider the following

```
Address of var1 variable: bff5a400
Address of var2 variable: bff5a3f6
```

example, which prints the address of the variables defined −

When the above code is compiled and executed, it produces the following result −

**What are Pointers?**

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is −

Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer Variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer

```
Type *var-name;
```

declarations −

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

**C - Strings**

Strings are actually one-dimensional array of characters terminated by a **null**character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of

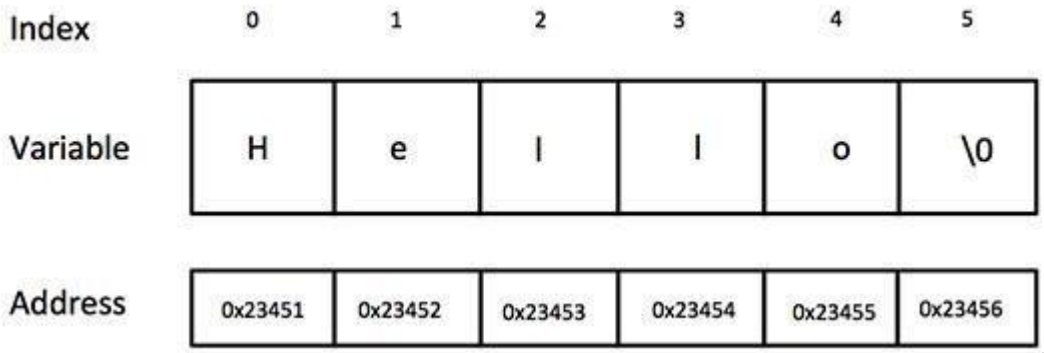
characters in the word "Hello."

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```
char greeting[] = "Hello";
```

If you follow the rule of array initialization then you can write the above statement as follows −

Following is the memory presentation of the above defined string in C/C++ −




```
#include <stdio.h>
int main () {
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
   printf("Greeting message: %s\n", greeting );
   return 0;
}
```

```
Greeting message: Hello
```

Actually, you do not place the *null* character at the end of a string constant. The C compiler automatically places the'\0' at the end of the string when it initializes the array. Let us try to print the above mentioned string −

When the above code is compiled and executed, it produces the following result −

C supports a wide range of functions that manipulate null-terminated strings −

| S.N. | Function & Purpose |
|---|---|
| 1 | **strcpy(s1, s2);**<br>Copies string s2 into string s1. |
| 2 | **strcat(s1, s2);**<br>Concatenates string s2 onto the end of string s1. |
| 3 | **strlen(s1);**<br>Returns the length of string s1. |
| 4 | **strcmp(s1, s2);**<br>Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2. |
| 5 | **strchr(s1, ch);**<br>Returns a pointer to the first occurrence of character ch in string s1. |
| 6 | **strstr(s1, s2);**<br>Returns a pointer to the first occurrence of string s2 in string s1. |

The following example uses some of the above-mentioned functions −

```
#include <stdio.h>
#include <string.h>
int main () {
   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   int  len ;
   /* copy str1 into str3 */
   strcpy(str3, str1);
   printf("strcpy( str3, str1) :  %s\n", str3 );
   /* concatenates str1 and str2 */
   strcat( str1, str2);
   printf("strcat( str1, str2):  %s\n", str1 );
  /* total lenghth of str1 after concatenation */
```

```
   len = strlen(str1);
   printf("strlen(str1) : %d\n", len );
   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
strcpy( str3, str1) :  Hello
strcat( str1, str2): HelloWorld
strlen(str1) :  10
```

**C - Structures**

Arrays allow to define type of variables that can hold several data items of the same kind. Similarly **structure** is another user defined data type available in C that allows to combine data items of different kinds.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −

- Title
- Author
- Subject
- Book ID

```
struct Books
   { char title[50];
   char  author[50];
   char subject[100];
   int  book_id;
} book;
```

**Defining a Structure**

To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows −

The **structure tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure −

**Accessing Structure Members:**

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword **struct** to define variables of structure type. The following example shows how to use a structure in a program −

```
#include <stdio.h>
#include <string.h>
struct Books {
   char  title[50]; char
   author[50]; char
   subject[100]; int
   book_id;
};
struct [structure tag]
   {member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
int main( ) {
   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */
   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;
   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;
   /* print Book1 info */
   printf( "Book 1 title : %s\n", Book1.title);
   printf( "Book 1 author : %s\n", Book1.author);
   printf( "Book 1 subject : %s\n", Book1.subject);
   printf( "Book 1 book_id : %d\n", Book1.book_id);
   /* print Book2 info */
   printf( "Book 2 title : %s\n", Book2.title);
   printf( "Book 2 author : %s\n", Book2.author);
   printf( "Book 2 subject : %s\n", Book2.subject);
   printf( "Book 2 book_id : %d\n", Book2.book_id);
   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

1) // program to print the message.#include<stdio.h> #include<conio.h>
   void main ()
   {
   printf (“welcome to c program”);getch ();}
2)// program to find sum and average of twonumbers.

```
#include<stdio.h> #include<conio.h>void main ()
{
int a, b;
float sum, avg;clrscr ();
printf (“enter the value of a and b :”);scanf (“%d%d”, &a, &b);
sum=a+b; avg=sum/2;
printf (“sum: =%f”, sum); printf (“\n average: =%f”, avg);getch ();
}
```

3) // program to find area and perimeter of rectangle.

```
#include<stdio.h> #include<conio.h> void main ()
{
int l, b;
float area, peri;clrscr ();
printf (“enter the value of l and b :”);scanf (“%d%d”, &l, &b);
area=l*b; peri=2*(l+b);
printf (“area: =%f”, area);
printf (“\n perimeter: =%f”, peri);getch ();}
```

4) // program to find area and circumference of circle.

```
#include<stdio.h> #include<conio.h> void main ()
{
int r;
float area, cir;clrscr ();
printf (“enter the value of r :”);scanf (“%d”, &r); area=3.142*r*r;
cir=2*3.142*r;
printf (“area: =%f”, area);
printf (“\n circumference: =%f”, cir);getch ();}
```

5) // program to find maximum of two no’s usingconditional operator.

```
#include<stdio.h> #include<conio.h> void main ()
{int a, b, max;
clrscr ();
printf (“enter the value of a and b :”);
scanf (“%d%d”, &a, &b);
max=(a>b) ? a: b;
printf (“%d= maximum of two no’s”, max);
getch ();}
```

6) // program to find even or odd no’s.

```
#include<stdio.h> #include<conio.h> void main ()
{int a; clrscr ();
printf (“enter the value of a :”);
scanf (“%d”, &a);
if (a%2==0)
printf (“%d is even”, a);if (a%2! =0)
printf (“%d is odd”, a);getch ();}
```

7) // program to find biggest of two numbers.

```
#include<stdio.h>#include<conio.h> void main ()
{int a, b; clrscr ();
printf (“enter the value of a and b :”);scanf (“%d%d”, &a, &b);
if (a>b)
printf (“%d is big”, a);
else
printf (“%d is big”, b);
getch ();}
```

8) // program to find biggest of three no’s.

```
#include<stdio.h>  #include<conio.h>void main ()
{int a, b, c;
clrscr ();
printf (“enter the value of a, b and c :”);
scanf (“%d%d%d”, &a, &b, &c);
if (a>b){if (a>b)
```

```
        printf (“%d is big”, a);else
printf (“%d is big”, c);}else{if (b>c)
printf (“%d is big”, b);else
printf (“%d is big”, c);}getch ();}
```

9)// program to display n natural numbers using dowhile.

```
#include<stdio.h> #include<conio.h> void main ()
{int i=1 , n;clrscr ();
printf (“enter the value of n:”);
scanf (“%d”, &n);
do{printf (“%d\n”, i);i++;
} while (i<=10);
getch ();}
```

10)// program to display sum of n naturalnumbers using do while.

```
#include<stdio.h> #include<conio.h> void main ()
{int i=1, n, sum=0;
clrscr ();
printf (“enter the value of n:”);
scanf (“%d”, &n);
do{
printf (“%d\n”, i);
sum=sum+i;
i++;} while (i<=10);
printf (“sum=%d”, sum);
getch ();}
```

11) // program to generate to print first n fibonaccinumber.

```
#include<stdio.h>  #include<conio.h> #include<math.h> void main ()
{
int fst=0, snd=1, n, count=2, next;
clrscr ();
printf (“enter the value of n :”);
scanf (“%d”, &n);
printf (“%d\n%d\n”, fst, snd);
while (count<n)
{next=fst+ snd;
printf (“%d\n”, next);
count++;
fst=snd; snd=next;
}getch ();}
```

12) // program to find sum & avg of arrayelements.

```
#include<stdio.h> #include<conio.h>void main ()
{
int a [100], i, n, sum=0, avg=0;
clrscr ();
printf (“enter the value of n :”);
scanf (“%d”, &n);
printf (“enter the elements of array :”);
for (i=1; i<=n; i++)
scanf (“%d”, &a[i]);
printf (“elements of array are\n”);
for (i=1; i<=5; i++)
{sum=sum + a [i];}
avg= sum/n;
printf (“\n sum=%d”, sum);
printf (“\n average=%d”, avg);
getch ();}
```

**What is C++:** **C++**

C++ is an object-oriented programming language. It is an extension to C programming. C++ programming language was developed in 1980 by **BjarneStroustrup** at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A. C++ runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX..

**BjarneStroustrup** is known as the **founder of C++ language.**

**Object-Oriented Programming (OOPs)**

C++ supports the object-oriented programming, the four major pillar of object oriented programming used in C++ are:

**1. Inheritance 2.Polymorphism 3.Encapsulation 4.Abstraction**

**Usage of C++**

By the help of C++ programming language, we can develop different types of secured and robust applications:

- ✓ Window application
- ✓ Client-Server application
- ✓ Device drivers
- ✓ Embedded firmware etc.

## C vs. C++

| No. | C | C++ |
|---|---|---|
| 1) | C follows the **procedural style Programming.** | C++ is multi-paradigm. It supports Both **procedural and object oriented.** |
| 2) | Data is less secured in C. | In C++, you can use modifiers for class members To make it inaccessible for outside users. |
| 3) | C follows the **top-down approach.** | C++ follows the **bottom-up approach.** |
| 4) | C does not support function overloading, Operator Overloading etc. OOP concepts. | C++ supports function overloading, Operator Overloading etc. OOP concepts.. |
| 5) | In C, you can't use functions in structure. | In C++, you can use functions in structure. |
| 6) | In C, **scanf() and printf()** are mainly used for Input/output. | C++ mainly uses stream **cin and cout** to perform Input and output operations. |

## C++ Features:

## C++ is object oriented programming language. It provides a lot of features that are given below.

- ➢ The main focus remains on data rather than procedures.
- ➢ Object-oriented programs are segmented into parts called objects.
- ➢ Data structures are designed to categorize the objects.
- ➢ Data member and functions are tied together as a data structure.
- ➢ Data can be hidden and cannot be accessed by external functions using access specifier .
- ➢ Objects can communicate among themselves using functions.
- ➢ New data and functions can be easily added anywhere within a program whenever required.
- ➢ Since this is an object-oriented programming language, it follows a bottom up approach, i.e. the execution of codes starts from the main which resides at the lower section and then based on the member function call the working is done from the classes.

**C++ Program:** Basically, a C++ program involves the following section:

```
Documentation
Pre-processor Statements
Global Declarations
The main ()
function{Local
Declarations
Program Statements & Expressions}
```

## Example : //program to display a line of text.

```
#include<iostream.h> #include<conio.h>
void  main(){clrscr();
 cout<<"welcome to c++ programming language";
 getch();}
```

**Standard output stream (cout):** The **cout** is a predefined object of **ostream** class. It is connected with the standard output device, which is usually a display screen. The cout is used in conjunction with stream **insertion operator (<<)** to display the output on a console

**Standard input stream (cin):**The **cin** is a predefined object of **istream** class. It is connected with the standard input device, which is usually a keyboard. The cin is used in conjunction with stream **extraction operator (>>)** to read the input from a console.

**Standard end line (endl):** The **endl** is a predefined object of **ostream** class. It is used to insert a new line characters and flushes the stream.

**C++ Variable:** A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times. **Data type variable list;**

The example of declaring variable is given below: **1) int x; 2) float y; 3) char z;**

Here, x, y, z are variables and int, float, char are data types.

We can also provide values while declaring the variables as given below:

**1) Int x=5, b=10; //declaring 2 variable of integer type 2) float f=30.8; 3) char c='A';**

- ✓ A variable can have alphabets, digits and underscore.
- ✓ A variable name can start with alphabet and underscore only. It can't start with digit.
- ✓ No white space is allowed within variable name.
- ✓ A variable name must not be any reserved word or keyword e.g. chars, float etc.

## C++ Data Types: There are 4 types of data types in C++ language.

| Types | Data Types |
|---|---|
| Basic Data Type | int, char, float, double, etc |
| Derived Data Type | array, pointer, etc |
| User Defined Data Type | structure |

## C++ Keywords

A keyword is a reserved word. You cannot use it as a variable name, constant name etc.

Example: if, else,switch,case,do,while,for,class etc

**C++ Operators:** Operators are special type of functions that takes one or more arguments and produces a new value.

- o Arithmetic Operators [+ ,- ,* ,/ ,%]
- o Relational Operators ]> ,>= ,< ,<= ,== ,!=]
- o Logical Operators [&& , ||, ! ]
- o Assignment Operator [=]
- o Unary operator [++ ,--]
- o Ternary or Conditional Operator [ ? , : ]
- o sizeof operator [sizeof(nt)]
- o pointer to member operator [ .* and ->*]
- o **scope resolution** operator [ :: ]
- o **address** operator [ & ]
- o **new o**perator (memory allocation operator)
- o **delete** operator(memory release operator)

## C++ Control Statementif-else

In C++ programming, if statement is used to test the condition. There are various types of if statements in C++.

- ➢ **if statement**
- ➢ **if-else statement**
- ➢ **if-else-if ladder**

**IF Statement**: The C++ if statement tests the condition. It is executed if condition is true.

**Syntax: if(condition) { //code to be executed}**

Example: #include <iostream.h> #include<conio.h>void main ()
{int num = 10;if (num % 2 == 0) { cout<<"It is even number"; }getch();}

**If-else Statement:** The C++ if-else statement also tests the condition. It executes if block if condition is trueotherwise else block is executed.

Syntax: if(condition) { //code if condition is true }
else { //code if condition is false }

Example:

```
#include <iostream.h>#include<conio.h> void main () {int num;
  cout<<"Enter a Number: ";cin>>num;
  if (num % 2 == 0) {  cout<<"It is even number"<<endl;  }else  {
        cout<<"It is odd number"<<endl;          }
  getch();
  }
```

**if-else-if ladder :** statement executes one condition from multiple statements.

## Syntax:

## Example: if(condition1){ //code to be executed if condition1 is true } else if(condition2){ //code to be executed if condition2 is true } else if(condition3){ //code to be executed if condition3 is true }

...

```
else{//code to be executed if all the conditions are false }
#include<iostream.h>
#include<conio.h> void main()
{
                        int avg;
                        clrscr();
                        cout<<"Enter average marks obtained in 5 subjects :";
                        cin>>avg;
                        cout<<"Your Grade is ";
                        if(avg>80)
                        { cout<<"A";}
                        else if(avg>60 && avg<=80)
                        { cout<<"B";}
                        else if(avg>40 && avg<=60)
                        { cout<<"C";}
                        else
                        { cout<<"D";}
                        getch();}
```

**Switch:** The C++ switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement in C++.

## Syntax:

```
switch (expression)
  {     case label1: statements  ;
                        break;
   case label2: statements;
                        break;
```

## Example:……

```
        default: staements;
                 break;          }
```

**// Program to build a simple calculator using switch Statement**

```
#include <iostream.h>
#include<conio.h> void main()
{
                        char op;
                        float num1, num2;
                        clrscr();
                        cout << "Enter an operator (+, -, *, /): ";
                        cin >> op;
                        cout << "Enter two operands: ";
                        cin >> num1 >> num2;
                        switch (op)
                        {
                           case '+':
                           case '-':
                           case '*':
                           case '/':default:}
cout << num1 << " + " << num2 << " = " << num1+num2;break;
cout << num1 << " - " << num2 << " = " << num1-num2;break;
cout << num1 << " * " << num2 << " = " << num1*num2;break;
cout << num1 << " / " << num2 << " = " << num1/num2;break;
// operator is doesn't match any case constant (+, -, *, /)cout << "Error! operator is not
correct";
break;
```

## Looping: getch();}

Loops are used in programming to repeat a specific block until some end condition is met. There are three types of loops in C++ programming:

1) for loop
2) while loop
3) do while loop

## While loop:

**Syntax:** **while(condition){**
**//code to be executed }**

## Example:

```
// C++ Program to compute factorial of a number
// Factorial of n = 1*2*3...*n
#include <iostream.h>
#include<conio.h>
void  main()
{
int number, i = 1, factorial = 1;
clrscr();
   cout << "Enter a positive integer: ";
   cin >> number;
   while ( i <= number)
    {
      factorial *= i;      //factorial = factorial * i;
      ++i;}
   cout<<"Factorial of "<< number <<" = "<< factorial;
   getch();
}
```

## Do-While Loop :

**Syntax:**

```
do
{ //code to be executed
}while(condition);
```

**Example** :

```
//C++ Program to display natural numbers up to 10.
#include <iostream>
#include<conio.h>
void  main()
 {
```

## Example: int i = 1;do{

```
     cout<<i<<"\n";i++;
  } while (i <= 10) ;getch();
}
```

**Syntax:** for(initialization; condition; incr/decr) {
//code to be executed }

**Example:** **//program to display your name for specified number of time**
#include <iostream.h>#include<conio.h> void main()
{

## C++ Functions: int time,i; clrscr();

cout << "how many times you want to display your name";cin >>time;
for(i=0;i<time;i++)
{
0 Cout<<"\n KEONICS";
} getch(); }

A function is a group of statements that together perform a task. Every C++ program has at least one function,which is main().

## Advantages of using functions in a program

- You can divide your program in logical blocks. ...
- Use of function avoids typing same pieces of code multiple times. ...
- Individual functions can be easily tested.
- In case of any modification in the code you can modify only the function without changing the structureof the program.

Types of Functions

There are two types of functions in C programming:

**1. Library Functions:** are the functions which are declared in the C++ header files such as ceil(x), floor(x),sqrt(x),sin(x) etc.

**2. User-defined functions:** are the functions which are created by the C++ programmer, so that he/she can use itmany times. It reduces complexity of a big program and optimizes the code.

## Declaration of a function: Syntax: Example: return_type function_name( parameter list ) {body of the function}

```
#include<iostream.h>
#include<conio.h>
// function declaration
int max(int num1, int num2);//function Declaration
void main () {
  // local variable declaration:int
  a = 100;
  int b = 200;int
  ret;
  // calling a function to get max value.ret = max(a, b);
                          cout << "Max value is : " << ret << endl;getch();}
                        // function returning the max between two numbersint max(int num1,
                        int num2) {
                          // local variable declarationint result;
                          if (num1 > num2)result = num1;
                          else
  result = num2;
  return (result);
                        }
```

**Inline functions**

When a function is declared as inline, the compiler places a copy of the code of that specific function at eachpoint where the function is called at compile time.

**Syntax:**

inline return_type function_name(arguments...)
{

## Example: //function_code //return_value;

}

```
//Simple Inline Function Example
Program in C++
#include<iostream.h>
#include<conio.h> inline int square(int x);void main() {
  int a = 100, b = 200;
  cout << "Simple Inline Function Example
  Program in C++\n"; cout << "\nSquare value
  for " << a << " is:" << square(a);cout <<
  "\nSquare value for " << b << " is:" <<
  square(b);getch();}
// Inline square functioninline int
square(int x) {
  return (x * x);}
```

## C++ Arrays

Array is a group of similar types of elements that have contiguous memory location.
In C++, array index starts from **0** **to** **(n-1)**


***Advantages of C++ Array***

- Code Optimization (less code)
- Random Access
- Easy to traverse data
- Easy to manipulate data
- Easy to sort

data etc. ***Disadvantages of C++ Array*** **->** Fixed size

## Array Types :

There are 2 types of arrays in C++ programming:

1. Single Dimensional Array
2. Multidimensional Array

## Single Dimensional Array:

A one-dimensional array is a group of elements having the same data type and same name.

**Declaration: data_type array_name[array_size];**
**Initialization: data_type array_name[array_size]=comma_separated_element_list};**
**Example :** // C++ One Dimensional Array

```
#include<iostream.h>#include<conio.h> void
main()
{
    clrscr();
    int arr[5] = {1, 2, 3, 4, 5};
    int i;
    for(i=0; i<5; i++)
    { cout<<"arr["<<i<<"] =
    "<<arr[i]<<"\n";}getch();
}
```

## C++ Passing Array to Function

**functionname(arrayname); //passing array to function**

## Example:

```
#include
<iostream.h>
#include<conio.h>
void printArray(int arr[5]);
void main()
{
        int arr1[5] = { 10, 20, 30, 40, 50 };
        int arr2[5] = { 5, 15, 25, 35, 45 };
        printArray(arr1); //passing
        array to function
        printArray(arr2);
}
void printArray(int arr[5])
{
   cout << "Printing array
   elements:"<< endl;for (int
   i = 0; i < 5; i++)
   {
            cout<<arr[i]<<"\n";
   }
getch();
}
```

## C++ Multidimensional Arrays:

**Declare Two Dimensional Array->**data_type array_name[row_size][column_size];

## Initialization:

data_type array_name[row_size][column_size] = { comma_separated_value_list} };

## Example :

```
// C++ Two Dimensional Array#include<iostream.h>
#include<conio.h>
void main()
{
        clrscr();
        int arr[5][2] = { {1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6} };
        int i, j;
        for(i=0; i<5; i++)
        {
                for(j=0; j<2; j++)
                {
                        cout<<"arr["<<i<<"]["<<j<<"] = "<<arr[i][j]<<"\n";
                }}getch();}
```

## C++ OOPs Concepts

The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language. Object Oriented Programming is a paradigm that provides many concepts such as **inheritance, data binding, polymorphism etc.**

The programming paradigm where everything is represented as an object is known as truly object-oriented programming language.

## OOPS (Object Oriented Programming System):

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

## Object
Any entity that has state and behaviour is known as an object.
For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.Class
**Collection of objects** is called class. It is a logical entity.

## Inheritance
**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. Itprovides code reusability. It is used to achieve runtime polymorphism.

## Polymorphism
When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.In C++, we use Function overloading and Function overriding to achieve polymorphism.

## Abstraction
**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don'tknow the internal processing.In C++, we use abstract class and interface to achieve abstraction.

## Encapsulation
**Binding (or wrapping) code and data together into a single unit is known as encapsulation.**
For example: capsule, it is wrapped with different medicines.

***Advantage of OOPs over Procedure-oriented programming language***

- OOPs makes development and maintenance easier where as in Procedure-oriented programminglanguage it is not easy to manage if code grows as project size grows.
- OOPs provide data hiding whereas in Procedure-oriented programming language a global data can beaccessed from anywhere.
- OOPs provide ability to simulate real-world event much more effectively. We can provide the solution ofreal word problem if we are using the Object-Oriented Programming language.

## C++ Object and Class
Since C++ is an object-oriented language, program is designed using objects and classes in C++.

## C++ Object:
Object is a runtime entity, it is created at runtime. Object is an instance of a class. All the members of the classcan be accessed through object.

## C++ Class
In C++, object is a group of similar objects.

**Syntax to create class:**

```
Class class_name
{
Data Members;Methods;
}
```

**Where class-|>keyword**
**Class Name->any valid identifies**

**Example:**

```
class A
{
  public:
  double length; //
  Length of a box
  double breadth; //
  Breadth of a box
  double height; //
```

Height of a box
}

- ✓ **Private**, **Protected**, **Public** is called visibility labels.
- ✓ The members that are declared private can be accessed only from within the class.
- ✓ Public members can be accessed from outside the class also.
- ✓ In C++, data can be hidden by making it private.

## C++ Object and Class Example

Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the objectand prints the object value.

```
#include <iostream.h>#include <conio.h> class
Student {
  public:
    int id;//data member (also
    instance variable) string
    name;//data member(also
    instance variable)
};
void main() {
   Student s1; //creating an
   object of Students1.id =
   201;
   s1.name = "Sonoo Jaiswal";cout<<s1.id<<endl;
   cout<<s1.name<<endl; getch();
}
```

## C++ Class Example: Store and Display Employee Information

```
#include <iostream.h>#include <conio.h> class Employee {
  public:
     int id;//data member (also instance variable)
     float salary; //data member(also
     instance variable)void insert(int i,
     float s)
      {
         id = i; salary = s;
      }
     void display()
      {
         cout<<"id="<<id<<"\nsalary= "<<salary<<endl;
      }
};
```

## C++ Constructor void main() {

```
Employee e1; //creating an object of Employee
Employee e2; //creating an object of Employee
e1.insert(201,15000);
e2.insert(202, 29000);
e1.display();
e2.display();
getch(); }
```

In C++, constructor is a special method which is invoked automatically at the time of object creation. It is used toinitialize the data members of new object generally. The constructor in C++ has the same name as class or structure.

## There can be two types of constructors in C++.

1. Default constructor

2. Parameterized constructor

**C++ Default Constructor** : A constructor which has no argument is known as default constructor. It is invoked atthe time of creating object.

Example:

```
#include <iostream.h>#include <conio.h>
   class Employee
{     public:          Employee()          {
  cout<<"Default Constructor Invoked"<<endl;          }
   };
void main()
   {
      Employee e1; //creating an
      object of EmployeeEmployee
      e2;
getch();                                  }C++ Destructor
```

A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in aclass. Like constructors, it is invoked automatically.

A destructor is defined like constructor. It must have same name as class. But it is prefixed with a **tilde sign (~).**

## C++ friend function

If a function is defined as a friend function in C++ then the protected and private data of a class can be accessed using the function. By using the keyword **friend** compiler knows the given function is a friend function .For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword **friend.**

## Declaration of friend function in C++Syntax:

```
class class_name
{
   friend data_type function_name(arguments);
};
```

**Example**:

```
 #include<iostream.h>
#include<conio.h>
class base {
int val1, val2;public:
   void get() {
      cout << "Enter two values:";cin >> val1>>val2;       }
   friend float mean(base ob);
};
float mean(base ob) {
   return
float(ob.val1 +
ob.val2) / 2;}void
main() {
   clrscr(); base obj;obj.get();
   cout << "\n Mean value is :
   " << mean(obj);getch(); }
```

## C++ Inheritance

In C++, inheritance is a process in which one object acquires all the properties and behaviours of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviours which are defined in other class.

In C++, the class which inherits the members of another class is called derived class and the class whosemembers are inherited is called base class. The derived class is the specialized class for the base class.

## Forms of inheritance in c++:

| | |
|---|---|
| Single inheritance | When one class inherits another class, it is known as single level inheritance |
| Multiple inheritance | One derived class with several base classes is called multiple inheritance |
| Hierarchical inheritance | One class inherits more than one derived class is known as hierarchical inheritance |
| Multilevel inheritance | When one class inherits another class which is further inherited by another class, it is known as multi-level inheritance in C++. Inheritance is transitive so the last derived class acquires all the members of all its base classes. |
| Hybrid inheritance | Combination of all forms of inheritance is known as hybrid inheritance. |

## Creation of derived class:

**Syntax:** class derived-class: visibility-mode base-class

```
{
   Class body
}
```

## C++ Single Inheritance Example:

```
#include<iostream.h>
#include<conio.h>
class emp {
        public:
          int eno;
          char name[20], des[20];void get() {
             cout << "Enter the employee number:";cin>>eno;
             cout << "Enter the employee name:";cin>>name;
             cout << "Enter the designation:";cin>>des;
                  }};
class salary : public emp
{
   float bp, hra, da, pf, np;
            public:
              void get1() {
                 cout << "Enter the basic pay:";cin>>bp;
                 cout << "Enter the HRA:";cin>>hra;
  cout << "Enter the DA :";cin>>da;
  cout << "Enter the PF";cin>>pf;
                                    }
void calculate() {
np = bp + hra + da - pf;
}void display() {
cout << eno << "\t" << name << "\t" << des << "\t" << bp << "\t" << hra << "\t" << da << "\t" <<pf
<< "\t" << np << "\n";

};
void  main()
 {}

  int i, n; char ch;
  salary s[10];clrscr();
                  cout << "Enter the number of employee:";cin>>n;
                     for (i = 0; i < n; i++)
```

```
                                        {   s[i].get();
                                            s[i].get1();
                                            s[i].calculate();
                                        }
                                cout << "\ne_no \t e_name\t des \t bp \t hra \t
                                da \t pf \t np \n";for (i = 0; i < n; i++)
                                        { s[i].display();    }getch();
                }
```

## C++ Multiple Inheritance Example:

```
                        #include<iostream.h>
                        #include<conio.h>
                         class student
                        {
      private:
      int rno, m1, m2;public:
      void get() {
         cout << "Enter the Roll no :";cin>>rno;
         cout << "Enter the two marks :";cin >> m1>>m2;
class sports
                         {
                        };
 };
    private:
         int sm; // sm = Sports markpublic:
            void getsm() {
            cout << "\nEnter the sports mark :";cin>>sm;
             }
                        class statement : public student, public sports
                        {
       int tot, avg;public:
       void display() {
       tot = (m1 + m2 + sm);avg = tot / 3;
                                    cout << "\n\n\tRoll No    : " << rno <<
                                    "\n\tTotal                : " << tot;cout <<
                                    "\n\tAverage              : " << avg;
                           }
                        };
                        void  main() {

                                }
```

## C++ Hierarchical Inheritance:

```
clrscr(); statement obj;obj.get();
obj.getsm(); obj.display();
getch();
```

**Syntax:**

```
Class A{              public void methodA()
                     {//Do Something  }
                }
Class B : public A      {
                public void methodB()
                {//Do Something          }
              }
Class C : public A        {
                 public void methodC()
                  {//Do Something      }          }
```

## C++ Multi Level

## Inheritance

**Syntax:**

```
                    c
lass
A{                  }
;
                  class B: public A{.....};
                  class C: public B{.....};
```

## C++ Polymorphism:

## The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms. It is a Greek word.

- **There are two types of polymorphism in C++:**
- ✓ **Compile time polymorphism:** It is achieved by function overloading and operator overloading which isalso known as static binding or early binding.
- ✓ **Runtime polymorphism:** It is achieved by method overriding which is also known as dynamic bindingor late binding.

## C++ Overloading (Function and Operator)

If we create two or more members having same name but different in number or type of parameter, it is known asC++ overloading.

Types of overloading in C++ are:

- ✓ **Function overloading**
- ✓ **Operators overloading**

### Function Overloading

Having two or more function with same name but different in parameters, is known as function overloading in C++.The **advantage** of Function overloading is that it increases the readability of the program because you don't need to use different names for same action.

**Example:**

- ✓ int test() { }
- ✓ int test(int a) { }
- ✓ float test(double a) { }
- ✓ int test(int a, double b) { }

## Function Overloading Example

```
#include <iostream>class Cal {
                public:  int add(int a,int b)
                         {    return (a + b);     } int
                         add(int a, int b, int c)
                         {     return (a + b + c);    }
                };
          void  main(void)
          {
                    Cal C;
                    cout<<C.add(10, 20)<<endl;
                   cout<<C.add(12, 20, 23); getch();
}
```

### Operators Overloading

Operator overloading is used to overload or redefine most of the operators available in C++. It is used to performoperation on user define data type.

The advantage of Operators overloading is to perform different operations on the same operand.

**Following are list of operators, which cannot be overloaded:**

**:: Scope resolution operator**
**.* Pointer to member operator**
**. Dot member5ship operator**

**?: conditional operatorSizeof() operator**

# Programs:

1) // program to print simple msg.

```
#include<iostream.h> #include<conio.h> void main()
{clrscr();
cout<<"\nwelcome\n to \nkeonics"<<endl<<"hubli";
getch();}
```

2)// program to demonstrate sum of two numbers..

```
#include<iostream.h>#include<conio.h>void main()
{int a=5,b=4,sum;
clrscr();
cout<<"\nsum of a and b:"<<a+b;
getch();}
```

3)// program to demonstrate arithmetic opertaion.

```
#include<iostream.h>#include<conio.h>void main()
{
int a=5,b=4,sum;
clrscr();
sum=a+b;
cout<<"\nsum of a and b:"<<sum;
getch();}
```

4)// program to demonstrate arithmetic opertaion.

```
#include<iostream.h>#include<conio.h>void main()
{
int a,b,sum;
clrscr();
cout<<"\nEnter the value of a and b:";
cin>>a>>b;
sum=a+b;
cout<<"\nsum of a and b:"<<sum;
getch();}
```

5)// program to print simple msg.

```
#include<iostream.h>#include<conio.h>void main()
{clrscr();
cout<<"\nwelcome\n to \nkeonics"<<endl<<"hubli";
getch();}
```

6)// program to demonstrate sum of two numbers..

```
#include<iostream.h>#include<conio.h>void main()
{
int a=5,b=4,sum;
clrscr();
cout<<"\nsum of a and b:"<<a+b;
getch();}
```

7)// program to demonstrate arithmetic opertaion.

```
#include<iostream.h>#include<conio.h>void main()
{
int a=5,b=4,sum;
clrscr();
sum=a+b;
cout<<"\nsum of a and b:"<<sum;
getch();}
```

8)// program to demonstrate area and perimeter of square.

```
#include<iostream.h>#include<conio.h>void main()
{
```

```
int a,area,peri;
clrscr();
cout<<"\nEnter the value of a:";
cin>>a;
area=a*a;
peri=4*a;
cout<<"\narea of square:"<<area;
cout<<"\nperimeter of square:"<<peri;
getch();}
```

9)// program to demonstrate area and perimeter of rectangle.

```
#include<iostream.h>#include<conio.h>void main()
{
int l,b,area,peri;
clrscr();
cout<<"\nEnter the value of l and b:";
cin>>l>>b;
area=l*b;
peri=2*(l+b);
cout<<"\narea of rectangle:"<<area;
cout<<"\nperimeter of rectangle:"<<peri;
getch();}
```

10) // program to demonstrate area and perimeter of traingle.

```
#include<iostream.h>#include<conio.h>void main()
{
int b,h,area;
clrscr();
cout<<"\nEnter the value of base and Height:";
cin>>b>>h;
area=0.5*b*h;
cout<<"\narea of Traingle:"<<area;
getch();}
```

11) // program to demonstrate area of circle.

```
#include<iostream.h>#include<conio.h>void main()
{
float r,area;
clrscr();
cout<<"\nEnter the value of base and Height:";
cin>>r;
area=3.142*r*r;
cout<<"\narea of Circle:"<<area;
getch();}
```

12) // program to find simple interest.

```
#include<iostream.h>#include<conio.h>void main()
{
float p=5000,t=3,r=3.5,si=0;
clrscr();
si=p*t*r/100;
cout<<"\nSimple Interest is:"<<si;
getch();}
```

13) // program to find simple interest.

```
#include<iostream.h>#include<conio.h>void main()
{
float p,t,r,si;
clrscr();
cout<<"\n Enter The Value of P,T and R:";
```

```
cin>>p>>t>>r;
si=p*t*r/100;
cout<<"\nSimple Interest is:"<<si;
getch();}
```

14)// program to find simple interest.

```
#include<iostream.h>#include<conio.h>void main()
{
float p,t,r,si;
clrscr();
cout<<"\n Enter The Value of Principle Amount:";
cin>>p;
cout<<"\n Enter The Value of Time:";
cin>>t;
cout<<"\n Enter The Rate of Interest:";
cin>>r;
si=p*t*r/100;
cout<<"\nSimple Interest is:"<<si;
getch();}
```

15)// program to Demonstrate Student Marks Sheet.

```
#include<iostream.h>#include<conio.h>void main()
{
clrscr();
char name[10];
int kan,eng,hin,total,rollno;
float per;
clrscr();
cout<<"\n          Enter the Rollno:";
cin>>rollno;
cout<<"\n            Enter the Name:";
cin>>name;
cout<<"\n Enter the Subject Kannada:";
cin>>kan;
cout<<"\n Enter the Subject English:";
cin>>eng;
cout<<"\n   Enter the Subject Hindi:";
cin>>hin;
total=kan+eng+hin;
per=total/3;
cout<<"\n    Total Marks of Student:"<<total;
cout<<"\n     Percentage of Student:"<<per;
getch();}
```

16) // program to Demonstrate Employee Details.

```
#include<iostream.h>#include<conio.h>void main()
{
clrscr();
char name[10],design[10],remarks[15],group[10];
int ecode;
float basic,hra,da,gross,pf,esi,tax,ded,net;
clrscr();
cout<<"\nEnter the Employee Code:";
cin>>ecode;
cout<<"\nEnter the Name of Employee:";
cin>>name;
cout<<"\nEnter the Employee Designation:";
cin>>design;
```

```
cout<<"\nEnter the Basic Salary:";
cin>>basic;
{
hra=basic*32/100;
da=basic*16/100;
gross=basic+hra+da;
pf=basic*12/100;
tax=basic*18/100;
esi=basic*1.75/100;
ded=pf+esi+tax;
net=gross-ded;
}
cout<<"\nEmployee Code:"<<ecode;
cout<<"\nName of Employee:"<<name;
cout<<"\nEmployee Designation:"<<design;
cout<<"\nEmployee Basic Salary:"<<basic;
cout<<"\nEmployee House Rent Allownce:"<<hra;
cout<<"\nEmployee Dearness Allownce:"<<da;
cout<<"\nEmployee Gross Salary:"<<gross;
cout<<"\nEmployee Service Helth Insurance:"<<esi;
cout<<"\nEmployee Providend Fund:"<<pf;
cout<<"\nEmployee Tax:"<<tax;
cout<<"\nEmployee Deduction:"<<ded;
cout<<"\nEmployee Net Salary:"<<net;
{
cout<<"\n     Remarks of Employee:"<<remarks;
}
{
if(net>=75000)
cout<<"Richest";
else if(net>=60000 && net<75000)
cout<<"Rich";
else if(net>=50 && net<60)
cout<<"Middle class";
else if(net>=35 && net<50)
cout<<"Normal";
else
cout<<"Poor";
}{
cout<<"\nGroup of Employee:"<<group<<endl;
}{
if(net>=75000)
cout<<"A+ Group";
else if(net>=60000 && net<75000)
cout<<"A Group";
else if(net>=50000 && net<60000)
cout<<"B Group";
else if(net>=35000 && net<50000)
cout<<"C Group";
else
cout<<"D Group";}
getch();}
```

17) // program to Demonstrate Student Marks Sheet using if else.

```
#include<iostream.h>#include<conio.h>void main()
{clrscr();
```

```
char name[10],grade,remarks[15],result[5];
int kan,eng,hin,total,rollno;
float per;
clrscr();
cout<<"\n          Enter the Rollno:";
cin>>rollno;
cout<<"\n            Enter the Name:";
cin>>name;
cout<<"\n Enter the Subject Kannada:";
cin>>kan;
cout<<"\n Enter the Subject English:";
cin>>eng;
cout<<"\n Enter the Subject Hindi:";
cin>>hin;
{
total=kan+eng+hin;
per=total/3;
cout<<"\n    Total Marks of Student:"<<total;
cout<<"\n     Percentage of Student:"<<per<<endl;
}{
cout<<"\n     Result of Student:"<<result<<endl;
}{
if(kan>=35 && eng>=35 && hin>=35)
cout<<"Pass";else
cout<<"Fail";
}{
cout<<"\nGrade of Student:"<<grade<<endl;
}{
if(per>=75)
cout<<"A";
else if(per>=60 && per<75)
cout<<"B";
else if(per>=50 && per<60)
cout<<"C";
else if(per>=35 && per<50)
cout<<"Pass";
else
cout<<"Fail";
}{
cout<<"\n     Remarks of Student:"<<remarks;
}{
if(per>=75)
cout<<"Outstanding";
else if(per>=60 && per<75)
cout<<"Excellent";
else if(per>=50 && per<60)
cout<<"Good";
else if(per>=35 && per<50)
cout<<"Pass";
else
cout<<"Fail";
}getch();}
```

18)// prog to find biggest of two no's using if condition.

```
#include<iostream.h>#include<conio.h>void main()
{int a,b;
```

```
clrscr();
cout<<"\n\nenter the two numbers:";
cin>>a>>b;
if(a>b)
cout<<a<<"=A is Bigger than B";
if(a<b)
cout<<b<<"=B is Bigger than A";
getch();}
```

19)// prog to find biggest of two no's using if condition.

```
#include<iostream.h>#include<conio.h>void main()
{int a=4,b=5;
clrscr();
//cout<<"\n\nenter the two numbers:";
//cin>>a>>b;
if(a>b)
cout<<endl<<a<<"=A is Bigger than B";
if(a<b)
cout<<endl<<b<<"=B is Bigger than A";
getch();}
```

20)// prog to find biggest of two no's using if condition.

```
#include<iostream.h>#include<conio.h>void main()
{int a=4,b=5;
clrscr();
if(a>b)
cout<<"\n\nA is:"<<a<<" Bigger than B";
if(a<b)
cout<<"\n\nB is:"<<b<<" Bigger than A";
getch();}
```

21)// prog to find even or odd no's.. using if.

```
#include<iostream.h>#include<conio.h>void main()
{int a;
clrscr();
cout<<"\n\nEnter the Value of A:";
cin>>a;
if(a%2==0)
cout<<"\n\nA is:"<<a<<"is  Even";
if(a%2!=0)
if(a%2==1)
cout<<"\n\nA is:"<<a<<" is Odd";
getch();}
```

22)// prog to find even or odd no's.. using if.

```
#include<iostream.h>#include<conio.h>void main()
{int a=5;
clrscr();
if(a%2==0)
cout<<"\n\nA is:"<<a<<"is  Even";
if(a%2!=0)
if(a%2==1)
cout<<"\n\nA is:"<<a<<" is Odd";
getch();}
```

23)// prog to find even or odd no's.. using if else.

```
#include<iostream.h>#include<conio.h>void main()
{
int a;
clrscr();
```

```
cout<<"\n\nEnter the Value of A:";
cin>>a;
if(a%2==0)
cout<<"\n\nA is:"<<a<<"is  Even";
else
cout<<"\n\nA is:"<<a<<" is Odd";
getch();}
```
24)// prog to find -ve or +ve no's.. using if .
```
#include<iostream.h>#include<conio.h>void main()
{int a=-5;
clrscr();
if(a>0)
cout<<"\n\nA is:"<<a<<"is  +ve'n";
else
cout<<"\n\nA is:"<<a<<" is -ve'n";
getch();}
```
25)// prog to find -ve or +ve no's.. using if else.
```
#include<iostream.h>#include<conio.h>void main()
{int a;
clrscr();
cout<<"\n\nEnter the Value of A:";
cin>>a;
if(a>0)
cout<<"\n\nA is:"<<a<<"is  +ve'n";
else
cout<<"\n\nA is:"<<a<<" is -ve'n";
getch();
}
```
26)// prog to find Eligible or not eligible for voting.
```
#include<iostream.h>#include<conio.h>void main()
{int age,n=1;
clrscr();
cout<<"\n\nEnter the Nationality if 1 india:";
cin>>n;
cout<<"\n\nEnter the age:";
cin>>age;
if(age>18)
if(n==1)
cout<<"\n\nyour natinality is: "<<n<<"\nyou are indian"<<"\nYour age is:"<<age<<"\nyou are
Eligible for Voting";
else
cout<<"\n\nyour natinality is: "<<n<<"\nyou are not indian"<<"\nYour age is:"<<age<<"\nyou are
Not Eligible for Voting";
getch();
}
```

# Java

Java was developed by James Gosling and his team, from Sun Microsystems in 1995 as an object-orientedlanguage for general-purpose business applications and for interactive, Web-based Internet applications

## Features of JAVA:

- **Object Oriented** − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** − unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable** − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust** − Java makes an effort to eliminate error prone situations by emphasizing mainly on compiletime error checking and runtime checking.
- **Multi-threaded** − With Java's multi-threaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted** − Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** − with the use of Just-In-Time compilers, Java enables high performance.
- **Distributed** − Java is designed for the distributed environment of the internet.
- **Dynamic** − Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

## Tools Need:

Computer with a minimum of 64 MB of RAM (128 MB of RAM recommended).Also need the following software −

- Linux 7.1 or Windows xp/7/8 operating system
- Java JDK 8
- Microsoft Notepad or any other text editor

| Comparison Between c++ and Java | |
|---|---|
| **C++** | **Java** |
| C++ is platform-Dependent. | Java is platform-independent. |
| C++ is mainly used for system Programming. | Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications. |
| C++ supports multiple Inheritance. | Java doesn't support multiple inheritance through class. It can be achieved by interfaces In java. |
| C++ supports operator Overloading. | Java doesn't support operator overloading. |

| | |
|---|---|
| C++ supports pointers. You can Write pointer programin C++. | Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java. |
| C++ uses compiler Only. | Java uses compiler and interpreter both. |
| C++ supports both call by value and call By reference. | Java supports call by value only. There is no call by reference in java. |
| C++ supports Structures and unions. | Java doesn't support structures and unions. |

How to write the simple program of java. We can write a simple hello java program easily after installing theJDK.

To create a simple java program, you need to create a class that contains main method **in any text editor(notepad, notepad++, word pad etc.)**

## Creating hello java example

```
class Simple{
   public static void
    main(String
    args[ ]){ System.out.pri
    nt("Hello Java");                    } }
```

## Program Execution:

Save the above Program as **Simple.java,** To compile and run this program, you need to open commandprompt by **start menu -> All Programs -> Accessories -> command prompt**.

**To compile:** javac Simple.java
**To execute:** java Simple

## Parameters used in first java program

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require creating object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[ ] args** is used for command line argument.
- **System.out.println()** is used print statement.

## Basic Structure of JAVA Program:

A Java program involves the following sections:

## How to set path in Java:

To set the temporary path of JDK, you need to follow following steps:

- Open command prompt
- Copy the path of jdk/bin directory
- Write in command prompt: set path=copied path

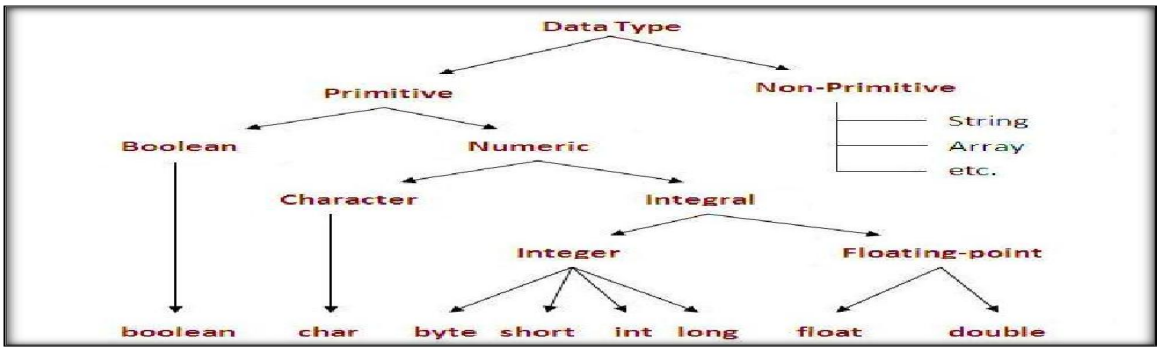
## For Example:

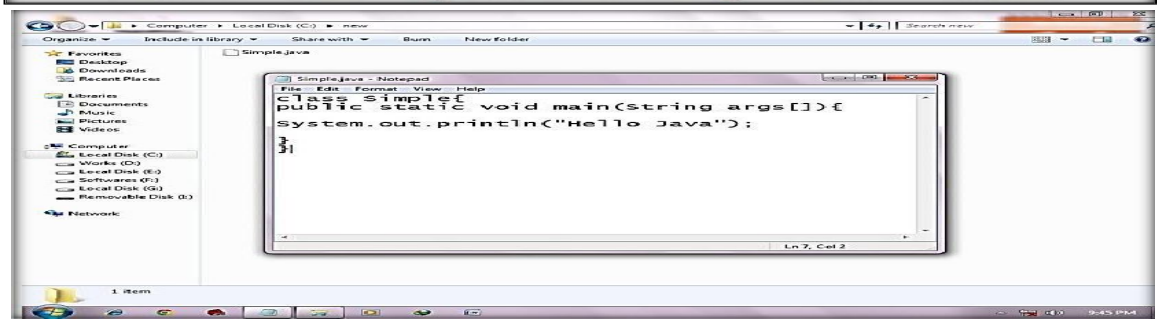
**Set path=C: \Program Files\Java\jdk1.6.0_23\bin**

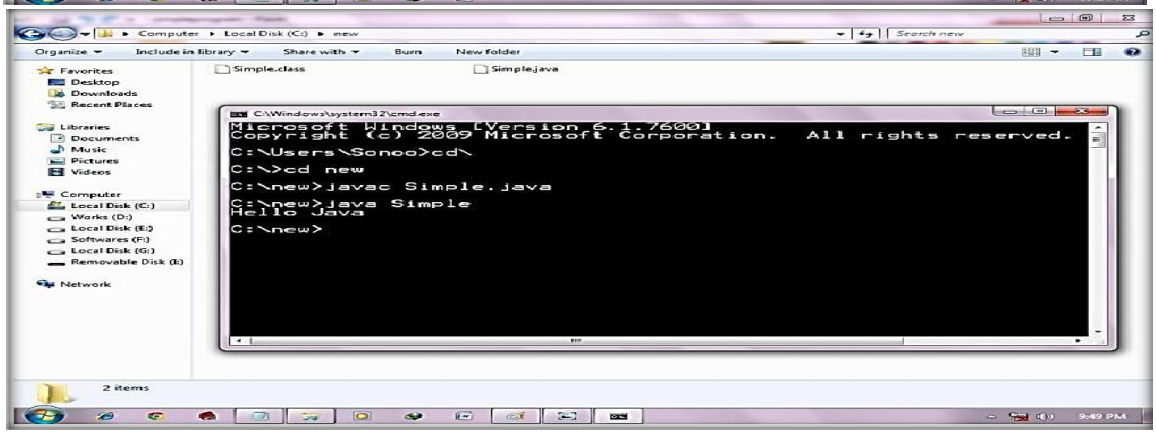
## Variables and Data Types in Java

A variable is a container which holds the value while the java program is executed. A variable is assigned with adatatype.

There are two types of data types in java: **primitive and non-primitive.**




# Operators in java:

**Operator** in java is a symbol that is used to perform operations. There are many types of operators in java which are given below:

- Unary Operator [ ++, -- ]
- Arithmetic Operator, [ +, - , * , /, %]
- Relational Operator [>, >=, <, <=, = =,! =]
- Logical Operator [&& , || , !]
- Ternary Operator [ ? :]
- Assignment Operator [ =]

## Java If-else Statement

The Java *if statement* is used to test the condition. It checks Boolean condition: *true* or *false*. There are varioustypes of if statement in java.

- if statement
- if-else statement
- if-else-if ladder
- nested if statement

**Java if Statement:** The Java if statement tests the condition. It executes the if block if condition is true.

**Syntax:** **if(condition) { //code to be executed }**

**Example**:
```
public class IfExample
                    {  public static void main(String[] args)
{     int age=20;
      if(age>18){     System.out.print("Eligible to VOTE");      }
} }
```

**Java if-else Statement:** The Java if-else statement also tests the condition. It executes the *if block* if condition istrue otherwise *else block* is executed.

**Syntax:**
```
if(condition){     //code if condition is true }
else{  //code if condition is false }
```

**Example:**
```
public class IfElseExample {
 public static void
    main(String[] args)
    {int number=13;
    if(number%2==0){
              System.out.println("even number");     }else{
    System.out.println("odd number");               }
} }
```

**Java if-else-if ladder Statement:** The if-else-if ladder statement executes one condition from multiplestatements.

**Syntax:**
```
if (condition1)
{/ /code to be executed if condition1 is true
} else if (condition2) {
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

**Example:**
```
public class Test {
  public static void main(String args[]) {int x =
    30;
    if( x == 10 ) {
      System.out.print ("Value of X is 10");
    } else if( x == 20 ) { System.out.print("Value
      of X is 20");
    } else if(x == 30) {System.out.print ("Value of
      X is 30");
    } else {
      System.out.print ("This is else statement");
                    }
                  }
                }
```

## Java Switch Statement

**Syntax:**
```
switch (expression){case value1:
          //code to be executed;break;
case value2:
          //code to be executed;break;
.....
   default:
          code to be executed if all cases are not matched;
}
```

**Example:**
```
public class Test {
   public static void
```

```
                                    main(String args[])
                                    {char grade = 'C';

switch(grade) {case 'A' :
                                        System.out.println("Excellent!");break;
                                    case 'B' :
                                    case 'C' :
                                        System.out.println("Well done");
                                        break;
                                case 'D' :
System.out.println("You passed");case 'F' :
          System.out.println("Better try again");break;
       default :
          System.out.println("Invalid grade");
    }
          System.out.println("Your grade is " + grade);
  }}
```

## Loops in Java

In programming languages, loops are used to execute a set of instructions/functions repeatedly when someconditions become true. There are three types of loops in java.

- for loop
- while loop
- do-while loop

## Java For Loop

**Syntax:**

```
for(initialization;condition;incr/decr){
//code to be executed
}
```

**Example:**

```
public class ForExample {
    public static void
            main(String[]
            args) {for(int
            i=1;i<=10;i+
            +){
                    System.out.println(i);
            }
    } }
```

## Java While Loop

**Syntax:**

```
while(condition){
//code to be executed  }
```

**Example:**

## Java Do While Loop Syntax:

Example:

```
public class While Example {
public static void main(String[] args) {int i=1;
   while(i<=10){ System.out.printl
      n(i);
   i++;
   } }  }
```

```
do{
//code to be executed
}while(condition);
```

```
public class DoWhileExample  { public static
```

```
void main(String[] args) {
   int i=1;do{
      System.out.println (i);i++;
   } while (i<=10);
                              } Java Comments
```

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be usedto hide program code for specific time.

## Types of Java Comments: There are 3 types of comments in java.

1. Single Line Comment //This is single line comment
2. Multi Line Comment
3. Documentation Comment

## *Example Programs:

1. // Fibonacci series program in java

```
without using recursion
class Fibonacci Example{
public static void main(String args[])
{  int n1=0,n2=1,n3,i,count=10;
 System.out.print (n1+" "+n2);//printing 0 and 1
 for (i=2;i<count;++i)//loop starts from 2 because 0 and 1 are already printed
 {    n3=n1+n2; System.out.print (" "+n3);n1=n2;
  n2=n3;
 } }}
```

2./* prime number program in java. In this java program, we will take a number variable and check whether thenumber is prime or not.*/

```
public class PrimeExample{
 public static void
  main(String
  args[]){int
  i,m=0,flag=0;
  int n=3;//it is the
  number to be checked
  m=n/2;
  if(n==0||n==1){
   System.out.print (n+" is not prime number");
  }else{ for(i=2;i<=m;i++){if(n%i==0){
     System.out.print (n+" is not
     prime number");flag=1;
     break;       }       }
   if(flag==0)  { System.out.print(n+" is prime number"); }
  }//end of else
 } }
```

3.// factorial Program
using loop in java.

```
class FactorialExample{
                public static void
                 main(String
                 args[]){int i,fact=1;
                 int number=5;//It is the number to
                 calculate factorial
                 for(i=1;i<=number;i++){
                    fact=fact*i;
                 }
                 System.out.print ("Factorial of "+number+" is: "+fact);
                }
```

}

## OOPs (Object Oriented Programming System)

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodologyor paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

**Object:** Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard,bike etc. It can be physical and logical.

**Class:** **Collection of objects** is called class. It is a logical entity.

**Inheritance:** **When one object acquires all the properties and behaviors of parent object** i.e. known asinheritance. It provides code reusability. It is used to achieve runtime polymorphism.

**Polymorphism:** When **one task is performed by different ways** i.e. known as polymorphism. For example: toconvince the customer differently, to draw something e.g. shape or rectangle etc. In java, we use method overloading and method overriding to achieve polymorphism.

**Abstraction:** **Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

**Encapsulation:** **Binding (or wrapping) code and data together into a single unit is known as encapsulation**.For example: capsule, it is wrapped with different medicines.

## Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects arecreated.

## Syntax to declare a class:

```
class <class_name>{ field;
  method;
```

Example: **1)**}

```
class Student{
        int id =100;//field or data member or instance variableString
        name;

 public static void main(String args[]){
  Student s1=new Student();//creating an object of Student
  System.out.println(s1.id);//accessing member through reference variable
  System.out.println(s1.name);  }  }Example: Rectangle
   class Rectangle{int length;
    int width;
```

## Object and Class

```
void insert(int l, int w){length=l;
                          width=w;
                         }
                         void calculateArea(){System.out.println(length*width);}
                        }
                        class TestRectangle1{
                         public static void
                          main(String
                          args[]){Rectangle
```

```
r1=new Rectangle();
Rectangle r2=new
Rectangle();
r1.insert(11,5);
r2.insert(3,15);
r1.calculateArea();
r2.calculateArea(); } }
```

## Inheritance in Java

**Inheritance in java** is a mechanism in which one object acquires all the properties and behaviors of parentobject. It is an important part of OPPs (Object Oriented programming system).

## Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint fromwhich objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class,extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is alsocalled a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methodsalready defined in previous class.

## Syntax of deriving a new class:

```
Class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

## Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.[In javaprogramming, multiple and hybrid inheritance is supported through interface only.]

***Note: Multiple inheritance is not supported in java through class.***

Java Inheritance

## Single Inheritance

When a single class gets derived from its base class, then this type of inheritance is termed as singleinheritance.

## Example:

```
class Teacher { void teach() {
 System.out.println ("Teaching subjects");
 }}
class Students
 extends
 Teacher {void
 listen() {
 System.out.println ("Listening to teacher");
 }}
class CheckForInheritance {
 public static void
 main(String args[])
 {Students s1 = new
 Students(); s1.teach();
 s1.listen ();}}
```

1. **Multi-Level Inheritance:** In this type of inheritance, a derived class gets created from another derived classand can have any number of levels.

## Example:

```
class Shape {
public void display() { System.out.print("Inside display"); }}
                        class Rectangle extends Shape { public void area() {
System.out.print("Inside area"); }}
class Cube extends Rectangle { public void volume() { System.out.print("Inside volume"); }}
public class Tester { public static void main(String[] arguments) { Cube cube = new Cube();
cube.display(); cube.area(); cube.volume(); }}
```

2. **Hierarchical Inheritance:** In this type of inheritance, there are more than one derived classes which getcreated from one single base class.

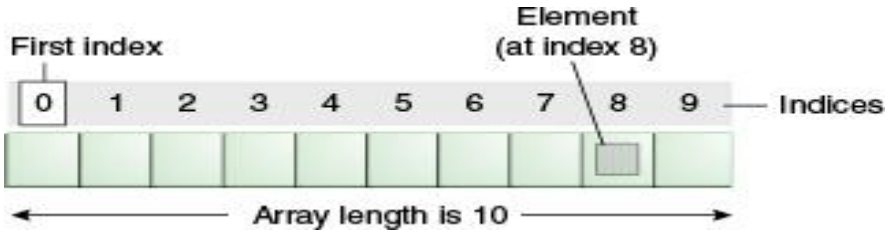
## Example:

```
class A
{  public void methodA()
 {      System.out.println ("method
of Class A"); }}class B extends
A{  public void methodB()
  {    System.out.println ("method
of Class B"); } }class C extends
A
{  public void methodC()  {    System.out.println("method
of Class C"); }}class D extends A
{  public void methodD()  {     System.out.println("method
of Class D"); }}class Java Example
{ public static void
    main(String args[])
    {B obj1 = new B();
    C obj2 = new C();
    D obj3 = new D();     //All classes can access the
    method of class Aobj1.methodA();
    obj2.methodA();
    obj3.methodA()
    ; }}
```

## Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package.There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc. **Simple example of java package**

The **package keyword** is used to create a package in java.




```
//save as Simple.javapackage mypack; public class
Simple {
 public static void main(String
   args[]){ System.out.print("Welco
   me to package");                                  } }
```

## Access Modifiers in java

**There are 4 types of java access modifiers:**

1. **Private:** The private access modifier is accessible only within class.
2. **Default:** If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package.
3. **Protected:** The **protected access modifier** is accessible within package and outside the package but through inheritance only. The protected access modifier can be applied on the data member, method and constructor.It can't be applied on the class.
4. **Public:** The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

## Java Array

Java array is an object the contains elements of similar data type. It is a data structure where we store similarelements. We can store only fixed set of elements in a java array. Array in java is index based, first element of the array is stored at 0 index.

## Advantage of Java Array

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.
- **Random access:** We can get any data located at any index position.

## Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

## Single dimensional array in java Syntax to Declare an Array in java

Data Type [ ] arr; (or) data Type [] arr; (or) data Type arr [];

## Instantiation of an Array in java

ArrayRefVar=new data type [size];

## Example:

```
class Test array {public static void main (String args []) {
    int a []=new
    int[5];//declaration and
    instantiation
    a[0]=10;//initialization
    a[1]=20;
    a[2]=70;
    a[3]=40;
    a[4]=50;
    //printing array
    for(int i=0;i<a.length;i++)//length is the property of array
    System.out.println(a[i]);
    }}
```

## Declaration, Instantiation and Initialization of Java Array

```
class Testarray1{ public static void main(String args[]){
int a[]={33,3,4,5};//declaration, instantiation and initialization
//printing array
for (int i=0;i<a.length;i++) //length is the property of array
System.out.println(a[i]);
}}
```

## Multidimensional array in java

## Syntax to Declare Multidimensional Array in java

Data Type [ ][ ] arrayRefVar; (or) data Type [ ][ ]arrayRefVar; (or) data Type arrayRefVar[][]; (or)

## Instantiation of a Multi-Dimensional Array in java

**int [][] arr=new int[3][3];//3 row and 3 column**

## Example:

```
class Testarray3 {
  Public static void main (String args []) {
  //declaring and initializing 2D array int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
  //printing 2D array for (int i=0; i<3; i++) {for (int j=0; j<3; j++) {
    System.out.print (arr[i][j]+" ");
   } System.out.print () ;}}}
```

## Java String

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence ofcharacters. The java.lang.String class is used to create string object.

## Example:

```
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2.  String s=new String(ch);
```

**Java String** class provides a lot of methods to perform operations on string such as compare(), concat(), equals(),split(), length(), replace(), compareTo(), intern(), substring() etc. There are two ways to create String object:

1. By string literal→ String s="welcome";
2. By new keyword→ String s=new String("Welcome");

## Example:

```
public class StringExample {
public static void
main(String
args[ ]){String
s1="java";
char ch[ ]={'s','t','r','i','n','g','s'};String s2=new String(ch);
String s3=new String ("example");System.out.print (s1); System.out.print (s2);
System.out.print (s3);
}}
```

Exception Handling in Java

The **exception handling** in java is one of the powerful mechanism to handle the runtime errors so that normalflow of the application can be maintained.

**Exception Handling** is a mechanism to handle runtime errors such as ClassNotFound, IO etc.

## Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method. Javatry block must be followed by either catch.

## Syntax of java try-catch

```
Try {
//code that may throw exception
} catch (Exception_class_Name ref) {  }
```

## Example:

```
public class Testtrycatch2{
  public static void main(String args[]){try{
    int data=50/0;
  }catch(Arithmetic Exception e){System.out.println(e);}
  System.out.println("rest of the code...");
}}
```

**Compile by: javac Testtrycatch2.java**

**Run by: java Testtrycatch2**

```
java.lang.ArithmeticException: / by zero
rest of the code...
```

1) // program to print simple message.

```
class sample
{
        public static void main(String args[])
        {
                System.out.println("welcome to java");
       System.out.print("welcome to java");
       System.out.println("welcome to java");}}
```

2) import java.lang.Math;

```
class prog2
{
  public static void main(String args[])
    {
        double x=25;
        double y=Math.sqrt(x);
        System.out.println("square root of"+x+"is"+y);
    }}
```

3) import java.lang.Math;

```
class prog3
{
  public static void main(String args[])
    {
        double x=5,y=2;
        double z=Math.pow(x,y);
        System.out.println(x+"to the power"+y+"is"+z);
    }}
```

4) import java.lang.Math;

```
class prog4
{
  public static void main(String args[])
    {
        double x=8,y=20;
        double z=Math.max(x,y);
        System.out.println("maximum number is="+z);
   }}
```

5) import java.lang.Math;

```
class xyz
{
  public static void main(String args[])
    {
     //int x,y;
     //x=10;
     //y=20;
     int a=10,b=20,sum=a+b;
      System.out.println("welcome "+sum);
   }}
```

6) import java.lang.Math;

```
class arith
{
  public static void main(String args[])
    {
      int a,b,sum,diff,p,q;
          a=100;
```

```
            b=20;
         sum=a+b;
        diff=a-b;
        p=a*b;
        q=a/b;
        System.out.println("a="+a+"\nb="+b);
        System.out.println("sum="+sum);
        System.out.println("diff="+diff);
        System.out.println("product="+p);
        System.out.println("que="+q);
    }}
```

7) //program to illustrate if else

```
class big
{
    public static void main(String args[])
      {
       int a,b;
       a=10;
       b=20;
       if(a>b)
         {
           System.out.println(a+"is BIG");
         }
    //else
     if(a<b)
         System.out.println(b+" is B  \nB is big");
    }
}
```

8) //program to illustrate if else ladder

```
class big1
{
    public static void main(String args[])
      {
       int a=5,b=15,c=20;
       // a=10;
       // b=12;
       // c=15;
       if(a>b&&a>c)
           System.out.println(a+"is BIG");
    else if( b>a&&b>c)
         System.out.println(b+"is big");
    else if( c>a&&c>b)
         System.out.println(c+"is big");
    }}
```

9) // program to find cases.

```
class touppercase
{
  public static void main(String arg[])
  {
   String s1,s2;
  s1="hell oKeo";
  s2="WEL COME";
  System.out.println(s1.toUpperCase());
  System.out.println(s1.toLowerCase());
  System.out.println(s1.charAt(2));
```

```
  System.out.println(s2.indexOf("L"));
   }
}
class equals1
{
  public static void main(String arg[])
{
String s1,s2,s3,s4;
s1="5abc";
s2="5";
s3="5abc";
s4="A";
System .out.println(s1+"equals"+s2+".....>"+s1.equals(s2));
System .out.println(s1+"equals"+s3+".....>"+s1.equals(s3));
System .out.println(s1+"equals"+s4+".....>"+s1.equals(s4));
}}
```

10) //program to illustrate if else

```
class even
{
    public static void main(String args[])
      {
       int n;
       n=10;
       //b=20;
       if(n%2==0)
         {
           System.out.println(n+"is Even");
         }
    //else
     if(n%2==1)
          System.out.println(n+" is  odd");
         //System.out.println(n+"  is odd");
    }}
```

11) //program to illustrate if else

```
class positive
{
    public static void main(String args[])
      {
       int n;
       n=-10;
            if(n>=0)
         {
           System.out.println(n+"is Positive");
         }
          if(n<=0)
          System.out.println(n+" is  Negative");
               }}
```

12) //program to illustrate if else

```
class si
{
    public static void main(String args[])
      {
       double p=5000,t=3,r=2.5,si=0;

       si=(p*t*r)/100;
```

```
        {
          System.out.println(si+":=Simple Interest");
        }          }}
```

13) //program to illustrate area and perimeter of rectangle

```
class area
{
    public static void main(String args[])
      {
       double l=50,b=30,area=0,peri;

       area=b*l;
       peri=2*(l+b);
         {
          System.out.println(area+":=Area of Rectangle");
          System.out.println(peri+":=Perimeter of Rectangle");
        }        }}
```

14) //program to illustrate area and perimeter of square

```
class sq
{
    public static void main(String args[])
      {
       double a=30,area=0,peri;
       area=a*a;
       peri=a*4;
         {
          System.out.println(area+":=Area of Square");
          System.out.println(peri+":=Perimeter of Square");
        }        }}
```

15) //program to illustrate area and perimeter of triangle

```
class triangle
{
    public static void main(String args[])
      {
       double b=30,h=50,area=0,peri;
            area=0.5*b*h;
              {
          System.out.println(area+":=Area of Square");
            }        }}
```

16) //program to illustrate area and perimeter of Circle

```
class circle
{
    public static void main(String args[])
      {
       double r=30,area=0,cir=0;
       area=3.142*r*r;
       cir=2*3.142*r;
         {
   System.out.println(area+":=Area of Circle");
   System.out.println("Circumprence of circle:="+cir);
    } }}
```

17) //program with multiple classes

```
class Room
{
```

```
   float length;
   float breadth;
   void getdata(float a,float b)
     {
       length=a;
       breadth=b;
     }
}
class RoomArea
{
   public static void main(String args[])
    {
      float area;
      Room room1=new Room();
  room1.getdata(14,10);
      area=room1.length*room1.breadth;
      System.out.println("Area="+area);
    }}
```

18) //program to illustrate classes and objects

```
class Rectangle
{
  int length,width;
  void getdata(int x,int y)
  {     length=x;
    width=y; }
  int rectArea()
 {
  int area=length*width;
 return(area);
 }}
class RectArea
{
    public static void main(String args[])
    {
      int area1,area2;
      Rectangle rect1=new Rectangle();
      Rectangle rect2=new Rectangle();
      rect1.length=15;
     rect1.width=10;
   rect2.getdata(30,50);
    area1=rect1.length* rect1.width;
    area2=rect2.rectArea();
  System.out.println("Area1="+area1);
  System.out.println("Area2="+area2);
}
}
```

19) //program to find marks using simpleif.

```
class simpleif
{
    public static void main(String args[])
      {
       int marks=50;
       if(marks>=50)
         {
           System.out.println("PASS:= "+marks);
         }            else           {
```

```
            System.out.println("FAIL:= "+marks);
          }         }     }
20) //program to illustrate static members
class Mathoperation
{
  static double mul(double x,double y)
   {
     return(x*y);
   }
   static double divide(double x,double y)
   {      return(x/y);    }}

21) // program using MathApplication.
class MathApplication
{
  public static void main(String args[])
   {
     double a=Mathoperation.mul(40.0,5.0);
     double b=Mathoperation.divide(a,2.0);
     System.out.println("b="+b);
   }}

22) //program to illustrate Method overloading
class Room
{
   float length,breadth;
  Room(float x,float y)
  {
    length=x;
    breadth=y;
 }
  Room(float x)
 {
  length=breadth=x;
  }
  float area()
 {
 return(length*breadth);
 }}
class MethodOverloading
{
    public static void main(String args[])
    {
    Room room1=new Room(25.0f,15.0f);
    Room room2=new Room(20.0f);
    System.out.println("Area1="+room1.area());
    System.out.println("Area2="+room2.area());
    }}

23) //program to illustrate nesting of members
class Nesting
{
   int m,n;
   Nesting(int x,int y)
   {
      m=x;
      n=y;
   }
```

```
  int largest()
  {
   if(m>n)
    return(m);
   else
     return(n);
  }
 void display()
   {
   int large=largest();//calling method
   System.out.println("largest value="+large);
   }
}
class NestingTest
{
  public static void main(String args[])
   {
    Nesting nest=new Nesting(50,40);
    nest.display();
  }}
```

24) //program to illustrate array

```
class Testarray{
public static void main(String args[])
{
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;

//printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

25) //find min element in array (pass arr as argument)

```
class Testarray2{
static void min(int arr[]){
int min=arr[0];
for(int i=1;i<arr.length;i++)
 if(min<arr[i])
  min=arr[i];
  System.out.println(min);
}
  public static void main(String args[]){
  int a[]={33,3,4,5};
min(a);//passing array to method
  }}
```

26) //program to illustrate method overriding

```
class Vehicle
{
void run()
{
System.out.println("Vehicle is running");}
}
```

```
class Bike2 extends Vehicle
{
void run()
{
System.out.println("Bike is running safely");
}
public static void main(String args[])
{
Bike2 obj = new Bike2();
obj.run();  }  }
```

27) // program using util.

```
import java.util.*;
import java.io.*;
class xyz1
{
    public static void main(String args[ ])
      {
       int a,b,c;
     DataInputStream dis=new DataInputStream(System.in);
       a=dis.readLine();
       b=dis.readLine();
       c=dis.readLine();
       if(a>b&&a>c)
            System.out.println(a+"is BIG");
    else if( b>a&&b>c)
          System.out.println(b+"is big");
    else if( c>a&&c>b)
          System.out.println(c+"is big");     }}
```

28) // program using util.

```
import java.util.Scanner;
public class AverageMarks
 {
  public static void main(String[] args)
{
     Scanner s = new Scanner ( System.in);
     System.out.print("Enter your name: ");
     String name=s.next();
     System.out.print("Enter marks in three subjects: ");
     int marks1=s.nextInt();
     int marks2=s.nextInt();
     int marks3=s.nextInt();
     double average = ( marks1+marks2+marks3)/3.0;
     System.out.println("\nName: "+name);
     System.out.println("Average: "+average);
  }
}
public class Add {
   public static void main(String[] args) {
      int sum = 45;
      for (int i = 0; i < args.length; i++) {
         sum = sum + Integer.parseInt(args[i]);
      }
      System.out.println("The sum of the arguments passed is " + sum);
   }
   }
```

29) // program to find matrix.

```
import java.io.*;
class matrix
{
  public static void main(String args[])
    {
     int a[][]=("10,20,30,40");
     int b[][]=("10,20,30,40");
     int s[][]=(i[2])(j[2]);
          //a[][]={10,20,30,40};
     //b[][]={11,22,33,44};
      System.out.println("sum of elements of the matrix\n");
      for(int i=0;i<2;i++)
           for(int j=0;j<2;j++)
             int s[i][j]=a[i][j]+b[i][j];
       for(int i=0;i<2;i++)
       {
           for(int j=0;j<2;j++)
           {
             System.out.println("s[i][j]");
            }
            System.out.println("s[i][j]");
      }
}
}
```

30) // program parent to child class.

```
class grandfather
{
     int age;
   void getdata(int x)
     {
          age=x;
     }
  void putdata()
   {
       System.out.println("AGE="+age);
   }
}
class father extends grandfather
{
  int sal;
   void accept(int y)
    {
      sal=y;
    }
 void display()
   {
       System.out.println("SALARY="+sal);
   }
}
  class son extends father
{
  int roll_no;
   void input(int z)
    {
      roll_no=z;
    }
```

```
 void output()
    {
        System.out.println("ROLL NO="+roll_no);
     }
}
31) // program  multilevel.
 class multilevel
{
   public static void main(String args[])
      {
           son s=new son();
            s.getdata(30);
            s.accept(9000);
            s.input(10);
             s.putdata();
             s.display();
              s.output();
       }
}

32) // program generating student marks sheet.
import  java.io.*;
class stu
{
          public static void main(String args[])throws IOException
          {
BufferedReader stu=new BufferedReader(new InputStreamReader(System.in));
int rollno,kan,eng,hin,tmar;
double per;
String name;
System.out.println("Enter the Rollno:");
rollno=Integer.parseInt(stu.readLine());
System.out.println("Enter the name:");
name=stu.readLine();
System.out.println("Enter the Marks of Kannada:");
kan=Integer.parseInt(stu.readLine());
System.out.println("Enter the English:");
eng=Integer.parseInt(stu.readLine());
System.out.println("Enter the Hindi:");
hin=Integer.parseInt(stu.readLine());
tmar=kan+eng+hin;
per=tmar/3;
System.out.println("Total Marks:="+tmar);
System.out.println("Percentage:="+per);
{
System.out.println("Grade:=");

          if(per>=75)
                    System.out.print("A");
          else if(per>=60 && per<75)
                    System.out.print("B");
          else if(per>=50 && per<60)
                    System.out.print("C");
          else if(per>=35 && per<50)
                    System.out.print("Pass");
          else
                    System.out.print("Fail");
                    }
```

```
}
}

33) // program generating Employee pay slip.
import  java.io.*;
class emp
{
        public static void main(String args[])throws IOException
        {
BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));
int empcode;
double basic,hra,da,gross,pf,tax,esi,ded,net;
String ename,design;
System.out.println("Enter the Employee Code:");
empcode=Integer.parseInt(stdin.readLine());
System.out.println("Enter the Employee Name:");
ename=stdin.readLine();
System.out.println("Enter the Employee Design:");
design=stdin.readLine();
System.out.println("Enter the Basic Salary:");
basic=Double.parseDouble(stdin.readLine());
{
        hra=basic*32/100;
        da=basic*16/100;
        gross=basic+hra+da;
        pf=basic*12/100;
        tax=basic*18/100;
        esi=basic*1.75/100;
   ded=pf+tax+esi;
   net=gross-ded;
}
System.out.println("             Employee Code:="+empcode);
System.out.println("             Employee Name:="+ename);
System.out.println("       Employee Designation:="+design);
System.out.println("      Employee Basic Salary:="+basic);
System.out.println("Employee House Rent Allownce:="+hra);
System.out.println("  Employee Dearness Allownce:="+da);
System.out.println("      Employee Gross Salary:="+gross);
System.out.println("    Employee Providend Fund:="+pf);
System.out.println("             Employee ESI:="+esi);
System.out.println("             Employee Tax:="+tax);
System.out.println("         Employee Deduction:="+ded);
System.out.println("        Employee Net Salary:="+net);
{
// System.out.println("                  Grade:=");

        if(net>=75000)
                System.out.print("                Grade:A");
        else if(net>=60000 && net<75000)
                System.out.print("Grade:B");
        else if(net>=50000 && net<60000)
                System.out.print("Grade:C");
        else if(net>=30000 && net<50000)
                System.out.print("Grade:D");
        else
                System.out.print("               Grade:E");

        }}}
```

34) //package org.apache.pdfbox.pdmodel.PDDocument;

```
import java.io.*;
import org.apache.pdfbox.pdmodel.PDDocument;
public class emp1
{
     int emp_id=110;
     int basic=10000;
    public void print( )
    {
            System.out.println("we are in pkg1 class emp");
            System.out.println("emp id="+emp_id+"\nbasic="+basic);
            System.out.println("exit from class emp");
    }
}
```

35) // program loop using while.

```
 class loop1
{
            public static void main(String[] args) {
                        int i=10;
                        while(i>=1)
                        {
                                    System.out.println(i);
                                    i--;
                        }           }}
```

36) // program loop using while.

```
class loop1
{
            public static void main(String[] args) {
                        int i=10;
                        while(i>=1)
                        {
                                    System.out.println(i);
                                    i--;
                        }           }  }
```

37) // program loop using do while.

```
class loop2
{
            public static void main(String[] args) {
                        int i=10;
                        do
                        {
                                    System.out.println(i);
                                    i--;
                        }while(i>=1);
            }}
```

38) // program loop using for loop.

```
class loop3
{
            public static void main(String[] args) {
                        int i ;
                        for(i=1;i<=10;i++)
                        {  System.out.println(i);     }}}
```