**Course 3C: Programming with C & C++**

**Unit I**

**Introduction and Control Structures:** History of ‘C’ - Structure of C program – C character set, Tokens, Constants, Variables, Keywords, Identifiers – C data types - C operators - Standard I/O in C - Applying if and Switch Statements

**Unit II**

**Loops And Arrays:** Use of While, Do While and For Loops - Use of Break and Continue Statements - Array Notation and Representation - Manipulating Array Elements - Using Multi Dimensional Arrays

**Unit III**

**Strings and Functions:** Declaration and Initialization of String Variables - String Handling Functions -Defining Functions - Function Call - Call By Value, Call By Reference – Recursion

**Unit IV**

**Classes and Objects Introduction to OOP and its basic features** - C++ program structure - Classes and objects - Friend Functions-Constructor – Types of constructors – Destructors.

**Unit V**

**Inheritance:** Inheritance - Types of Inheritance -Types of derivation- Public – Private - Protected Hierarchical Inheritance - Multilevel Inheritance – Multiple Inheritance - Hybrid Inheritance

**References:**

(1) E. Balagurusamy "Object oriented programming with C++

(2) R.Ravichandran "Programming with C++"

(3) Mastering C by K R Venugopal and Sudeep R Prasad, McGraw Hill

(4) Expert C Programming: Deep Secrets Kindle Edition Peter van der Linden

(5) Let Us C YashavantKanetkar

(6) The C++ Programming Language Bjarne Stroustrup

(7) C++ Primer Stanley B. Lippman, Josée Lajoie, Barbara E. Moo

**UNIT - I**

**1Q) History of C-language**

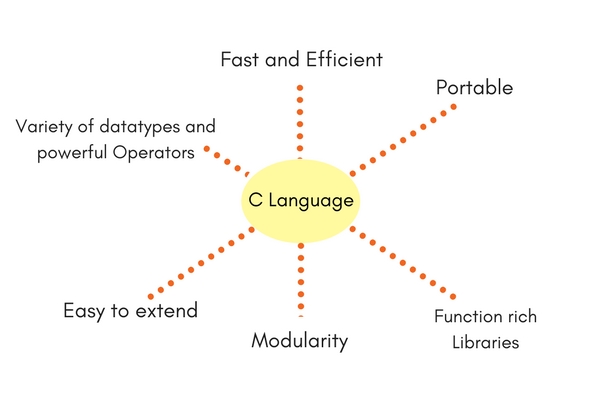
**Ans:**

* The C programming language is a **structure oriented programming language**, developed at Bell Laboratories in **1972 by Dennis Ritchie**
* C programming language features were derived from an earlier language called **“B”** (Basic Combined Programming Language – BCPL)
* C language was invented for implementing UNIX operating system
* **In 1978, Dennis Ritchie and Brian Kernighan** published the first edition “The C Programming Language” and commonly known as K&R C
* **In 1983, the American National Standards Institute (ANSI)** established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or “ANSI C”, was completed late 1988.

**2Q) Write the features of C-language**

**Ans:**

1. It is **structured oriented programming Language**.
2. It is highly **portable** this means that programs once written can be run on another machines with little or no modification.
3. It is a **robust language** with rich set of built-in functions and operators that can be used to write any complex program.
4. The C compiler combines the capabilities of an assembly language with features of a high-level language.
5. Programs Written in C are **efficient and fast**. This is due to its variety of data type and powerful operators.
6. It is many time faster than BASIC.
7. A C program is basically a collection of functions that are supported by C library. We can also create our own function and add it to C library.
8. C language is the most widely used language in operating systems and embedded system development today.
9. It is used to perform any kind of calculations.
10. It is used to control hardware components of a system.
11. It has a wide variety of derived data structures like pointers, arrays structures and unions apart from fundamental data types like integers, floating point numbers and characters.
12. It has the ability to deal efficiently with bits, bytes, word, addresses etc.,



**3Q) Write about C-Tokens?**

**Ans:**

A punctuation marks, commas, semi-colons, characters etc., is called as a C-Token. C-Tokens are of 6 types. They are –

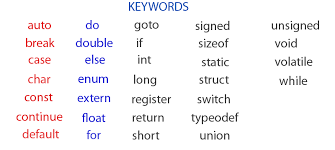
1. Identifiers
2. Keywords
3. Operators
4. Datatypes
5. Strings
6. Special Symbols
7. **IDENTIFIERS**

An identifier is a function name or variable name. To define identifiers the following rules to be followed. They are –

1. The first character of identifier name should always begin with alphabet or underscore.
2. In between identifier name there should not be any special symbol except underscore ("\_").
3. Keywords should not be defined as Identifier.
4. Duplicate identifiers cannot be defined.
5. **KEYWORDS**

The words which are already pre-defined in a c-language is said to be keywords. These are also called as **reserve words**. C-language contains **32 keywords.**

**eg:-**



1. **OPERATORS**

An operator is nothing but a symbol which, is used to operate the operands. The operators are

1. Arithmetic operators
2. Relational / Comparison operators
3. Logical Operators
4. Assignment Operators
5. Bitwise Logical Operators
6. Unary operators
7. Ternary Operators

**Arithmetic Operators**

The arithmetic operators are used to perform arithmetic calculations such as addition, subtraction, multiplication and division. The arithmetic operators are

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Examples** |
| + | Addition | A=10,B=20,C=A+B=>30 |
| - | Subtraction | A=10,B=20,C=A-B=>-10 |
| \* | Multiplication | A=10,B=20,C=A\*B=>200 |
| / | Division | A=10,B=2,C=A/B=>5 |
| % | Modulus | A=10,B=3,C=A%B=>1 |

**Relational / Comparison Operators**

The relational or comparison operators are used to compare different operands. The relational operators are

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Examples** |
| > | Greater than | A=10,b=5,a>b =>0(true) |
| < | Less than | A=10,b=5,A<b =>-1(false) |
| >= | Greater than equals to | A=10,b=5,a>=b =>0(true) |
| <= | Less than equals to | A=10,b=5,A<=b =>-1(false) |
| == | equals to | A=10,b=5,A= =b =>-1(false) |
| != | not equals to | A=10,b=5,A!=b =>0(true) |

**Logical Operators**

The logical operators are used to combine two or more expressions into one. The logical operators are

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| && | And |
| || | Or |
| ! | Not |

**Eg for AND:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Exp1** | **&&** | **Exp2** | **Result** |
| **T** | **&&** | **T** | **T** |
| **T** | **&&** | **F** | **F** |
| **F** | **&&** | **T** | **F** |
| **F** | **&&** | **F** | **F** |

**Eg for OR:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Exp1** | **||** | **Exp2** | **Result** |
| **T** | **||** | **T** | **T** |
| **T** | **||** | **F** | **F** |
| **F** | **||** | **T** | **F** |
| **F** | **||** | **F** | **F** |

**Eg for NOT:**

|  |  |
| --- | --- |
| **!Exp** | **Result** |
| **T** | **F** |
| **F** | **T** |

**Bitwise Logical Operators**

The Bitwise logical operators are used to perform bit calculations. The Bitwise logical operators are

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| >> | Right Shift |
| << | Left Shift |
| ~ | Complement |
| ^ | XOR |

**Assignment Operators**

The assignment operators are used to assign R.H.S value to L.H.S either before calculation or after calculation.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Examples** |
| = | Assigns RHS value to LHS | **a=10** |
| += | Assigns RHS value to LHS after addition | **a=10**  **a+=3=>13**  **or**  **a=a+3=>13** |
| -= | Assigns RHS value to LHS after subtraction | **a=10**  **a - =3=>7**  **or**  **a=a - 3=>7** |
| \*= | Assigns RHS value to LHS after multiplication | **a=10**  **a\*=3=>30**  **or**  **a=a\*3=>30** |
| /= | Assigns RHS value to LHS after Division | **a=10**  **a/=2=>5**  **or**  **a=a/2=>5** |
| %= | Assigns RHS value to LHS after modulus | **a=10**  **a%=3=>1**  **or**  **a=a%3=>1** |

**Unary Operator**

The Unary operators are used to perform unary calculations. The unary operators are incrementation and decrementation. The incrementation can be post incrementation or pre-incrementation and the decrementation post decrementation or pre-decrementation.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Examples** |
| ++ | Incrementation | A=5  A++ =>6 |
| -- | Decrementation | A=5  A-- =>4 |

**Ternary Operators**

It is also called as conditional operator. The conditional operators are used to execute true statement only when the condition is true otherwise it executes false statement.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| ? | Question mark |
| : | colon |

**4. DATATYPES**

Data types are used to define the variables that the same of type data it can store in memory. The data types determine the type of data to be stored in memory. The data types are used to represent the different values to be stored in the variable. They are

1. Pre-defined data types
2. User-defined data types
3. Derived data types

**Pre-defined data types:**

The data types which are already existed in C they are known as pre-defined data types. The pre-defined data types are

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **Types** | **Range** | **Memory** |
| Integer | Signed Int | -32,678 to +32,767 | 2 bytes |
| Unsigned Int | 0 to 65,535 | 2 bytes |
| Signed Long Int | -2,148,483,648 to +2,148,483,647 | 4 bytes |
| Unsigned Long Int | 0 to 4,296,967,295 | 4 bytes |
| Real | Float | 3.4 \* 10 -38 to 3.4 \* 10 +37 | 4 bytes |
| Double | 1.7 \* 10 -308  to 1.7 \* 10+ 307 | 8 bytes |
| Char | Signed char | -128 to +127 | 1 byte |
| Unsigned char | 0 to 255 | 1 byte |

**User-defined data types:**

The datatypes which creates by the user is called as user-defined data types. The user-defined data types can be created by using keywords like struct, class, enum, etc.

**Derived data types:**

The types which can be created using pre-defined data types, they are known as derived data types. The derived data types that we can create are **arrays, functions, files, pointers etc.**

**String:**

A sequence of characters which are organized under one variable or heading, it is called as a string. A string can be represented within the double quotes.

**Eg:** s1=”lotus-123”;

s2=”[anil.123@gmail.com](mailto:anil.123@gmail.com)”;

**Special Symbols:**

The symbols which have special functionality are called as special symbols.

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| [ ] | Square Brackets |
| ( ) | Parantheses |
| { } | Braces |
| “ “ | Double Quotes |
| ‘ ‘ | Single Quotes |

**4Q) Write about Constant?**

**Ans:**

Constant is also called as **Literals.** A constant is a fixed value that will not be changed ever during the execution of the program. Once the programmer defines a value to a constant, it remains same through the entire program. C supports several types of constants.

1. **Integer constants:**

An integer constant is a sequence of numerical digits. There are three types of integer constants.

* + - 1. **Decimal integer constant** consists of a set of digits 0 to 9. Decimal integers may be either positive or negative.

**Eg: 1234, 3443, -1233, 0 , 92929**

* + - 1. An **octal integer constant** consists of any combination of digits from the set 0 to 7 with a leading 0. Octal values have no sign.

**Eg: 034 , 027, 0736**

* + - 1. **Hexadecimal integer constant** consist a set of digits 0 to 9 and alphabets A to F to represent the values 10 to 15. Each hexadecimal value begins with 0x.

**Eg: 0x20, 0xF5, 0xabf6**

1. **Real Constants:**

The numbers containing fractional parts are called as **real constants**. To represent fixed floating point values, we use real constants. These numbers are represented with a decimal value containing a decimal point. A real constant may be either positive or negative

**Eg: 3.1428, 0.000455, -1.346**

A real number may also be expressed in exponential notation.

**Eg: 2.1565e2**

E2 means multiply the number 102

Then the above value becomes 215.65

1. **Character constants:**

A character constant contains single character enclosed within a pair of single quotation marks. The character may be an alphabet, digit or any symbol.

**Eg: ‘C’, ‘8’,’$’**

1. **String constants:**

A string constant is a sequence of characters enclosed between a pair of double quotation marks. A string constant may contain alphabets, digits, any symbols and white spaces.

**Eg: “hello world” “1234” “@#$&^”**

1. **Backslash Character constants:**

C supports backslash character constants that are used to format the output presented to the user. These characters are also called as **escape sequence characters**.

|  |  |
| --- | --- |
| **Code** | **Meaning** |
| \n | New line |
| \a | Beep sound |
| \t | Horizontal Tab space |
| \’ | Single quotes |
| \” | Double Quotes |
| \\ | Backslash |
| \0 | Null value |
| \r | Carriage return |

**5Q) What is Variable? Explain different types of variables?**

**Ans:**

A variable is nothing but a space in a memory where values can be constantly changed during run time of a program. To define variables the following rules to be followed. They are

* 1. Variable name should always begin with alphabet
  2. In between variable name there should not be any special symbol except underscore.
  3. Duplicate variable names cannot be defined.
  4. Keywords should not be defined a variables

**Types of Variables:**

**Local variable**

A local variable will be visible to the function in which it is declared, but not to other. It is declared inside the function. A local variable is also called automatic variable, because it is automatically created and destroyed.

**Global variable**

A global variable is visible to all the functions and it is declared outside the function. The proper place of declaration of global variable is at the beginning of the main program or before the function sub program. In other word before the declaration of all the function sub programs.

**Variable Declaration:**

Datatype var1,var2,…;

**Eg:** int a,b;

float m,n;

char c;

char s[15];

**Initialization of Variable:**

**Eg:** int a=10,b=20;

float x=1.2;

char c=’A’;

char s[15]={“Muthahar”};

**6Q) Structure of C-Program**

**Ans:** Structure of C program contains 6 sections. They are –

1. **Documentation Section**

The documentation section comprises a set of comment lines giving the name of the program, the author and other details, which the programmer would like to refer to at a later stage. The comment about a program can be written within the following symbols

**/\* Paragraph Comment \*/**

1. **Link Section**

The link section usually contains header files include statement. It provides instructions to the compiler to link functions from the C standard library.

#include<headerfile.h>

1. **Definition section**

The definition section defines all symbolic constants. So it contains preprocessor commands which always start with # as first character.

#define cons\_name value

1. **Global Declaration section**

All global variables are declared in the global declaration section. Variables declared outside all functions are called global variables and they may be accessed by any function within our program. Global variables exist the entire time your program is executing.

1. **Main function section**

This section is a part of every c program. Any c program is a set of functions which must be main() function. The empty parentheses after main() function is necessary and indicate that this main() function doesn’t include any arguments.

This section contains two parts, **local variable declaration part** and **executable part**

**The declaration part** declares all local variables used in the executable part. Every function contains one or more instructions which are enclosed within the pair of curly braces. Each statement should end with a semi-colon.

**Variables declared inside a function are called local variables**. A local variable is known to and may be accessed by only the function in which it is declared.

1. **Sub program section**

It contains all the user defined functions that are called in the main() function. User defined functions are placed immediately after the main() function.

**7Q) Input and Output Functions**

**Ans:**

1. **printf()**

It is a standard formatted output function, which is used to display an output during run time of a program.

***Syntax:*** printf(“<Message><FormatControls><EscapeSequences>”, var1,var2);

**Format Controls**

The format controls are used to display the output as per the format specified by the user. The Format controls are –

|  |  |
| --- | --- |
| **Format** | **Meaning** |
| %d (or) %i | Print an integer |
| %u | Print an Unsigned integer |
| %ld | Print a long integer |
| %f | Print float value |
| %e | Print float value in exponential form |
| %c | Print a character |
| %s | Print a string |
| %x | Print hexadecimal integer using lower case a-f |
| %X | Print hexadecimal integer using lower case A-F |
| %o | Print octal value |
| %lo | Print octal long |
| %p | Print a pointer value |

**Escape Sequences:**

These are non-printable characters but its functionality can be observed on the output screen.

|  |  |
| --- | --- |
| **Code** | **Meaning** |
| \n | New line |
| \a | Beep sound |
| \t | Horizontal Tab space |
| \’ | Single quotes |
| \” | Double Quotes |
| \\ | Backslash |
| \0 | Null value |
| \r | Carriage return |

1. **scanf():**

It is a standard formatted input function. It is used to input values during run time of a program.

***Syntax*** – Scanf(“format control”,&var1,&var2);

**Note:** While input data during runtime of a program, in front of a variable ampersand (&) should be defined. But while accepting string “&” operator should not be defined in front of a variable

1. **getchar():**

It is a character input function. It is used to input/read number of characters during run time of a program and it can store only the starting character. This function is available in a header file called stdio.h.

**Syn:** var=getchar()

1. **getche():**

It is a character input function. It is used to input/read character during run time of a program and it can store in the memory. This function is available in a header file called stdio.h.

**Syn:**var=getche()

1. **getch():**

It is a character input function. It is used to input/read character during run time of a program and it will never echoes onto the screen but it can store in the memory. This function is available in a header file called stdio.h.

**Syn:**var=getch()

1. **putchar():**

It is a character output function which is used to display character as output during run time of a program.

**Syn:** putchar(var)

**8Q) Control Structures / Control Statements**

**Ans:**

1. Conditional Statements
2. Iterative Statements
3. Jumping Statements / Unconditional Statements
4. **Conditional Statements**

The conditional statements are used to control the flow of execution of statements of a program. The conditional statements are

1. Simple if
2. if .. else
3. if .. else if
4. Nested if
5. Switch
6. Ternary Operator

**Simple if:**

This conditional statement is used to execute true statements only when the condition is true, otherwise it skips the if statement.

**Syntax:**

if (expression )

{

Statement1;

Statement2;

}

**if .. else:**

This conditional statement executes true statements only when the condition is true otherwise it executes false statements.

**Syntax:**

if (expression )

{

statement1;

statement2;

}

else

{

statement1;

statement2;

}

**if .. else if:**

It is also called as a branching statement or Ladder statement. This conditional statement executes the statements based on its respective conditional statement.

**Syntax:**

if ( expression )

{ statement1;

statement2;

}

else

if(expression2)

{ statement1;

statement2;

}

else

{ Statement1;

Statement2;

}

**Nested if:**

A if statement which can execute within an if statement itself, it is called as nested if conditional statement. It is also called as a multi-level branching statement.

**Syntax:** if(exp1)

{

if(exp1.1)

{ statement1;

statement2;

}

else

if(exp1.2)

{ statement1;

statement2;

}

}

else

if(exp2)

{

if(exp2.1)

{ statement1;

statement2;

}

else

if(exp2.2)

{ statement1;

statement2;

}

}

**Switch:**

It is also called as branching statement. This conditional statement is very similar to if .. else if conditional statement but the execution of switch statement is very faster than if..else if conditional statement.

**Syntax:**  switch(exp)

{

case constant1 : statement1;

statement2;

break;

case constant2: statement1;

statement2;

break;

case constant3: statement1;

statement2;

break;

default : statement;

}

**Ternary operator**

The ternary operators are also called as conditional operators. The ternary operators are used to control the flow of execution of statements of a program. The conditional operators execute a true statement only when the condition is true otherwise it executes a false statement.

**Syntax:** Expression? True statement : false statement;

**UNIT - II**

**1Q) Write about Loops?**

**Ans:**

Executing a statement or group of statements for a repeated number of times, it is called as a loop. C-language contains different types of loops. They are –

1. for .. loop
2. while .. loop
3. do .. while loop

**For .. loop:**

The for loop iterates the elements for the fixed number of times. It should be used if number of iterations is known. The for .. loop can be construct by using the following 3 steps.

1. **Initialization**

Here, we initialize the variable in use, it marks the start of a for..loop. An already declared variable can be used or a variable can be declared local to loop only.

1. **Testing Condition:**

It is used for testing the exit condition for a loop. It must return a boolean value. It is also an entry control loop as the as the condition is checked prior to the execution the loop statements.

1. **Changing value:**

It is used for updating the variable for next iteration.

**Syn-1:**

for(initialvalue; conditionalvalue;changingvalue)

{

block of statements;

}

statement -x;

**Syn-2:**

initialvalue;

for( ;conditionvalvalue; )

{ block of statements;

changingvalue;

}

**Nested for..loop:**

A for..loop which executes within another for..loop itself, such loop can be called as nested for..loop. In nested for.. loops, the execution of loops always depends on the conditional values of the loops. In nested for..loop the inner loop will execute first and then the outer loop will get executed.

**Syntax:** for(initialvalue;conditionalvalue;changingvalue)

{

for(initialvalue;conditionalvalue;changingvalue)

{

block of statements;

}

}

**While .. loop:**

It is also called as a conditional loop. In this loop a statement or group of statements can be executed for a repeated number of times only when the condition is true otherwise the loop will be terminated.

**Syntax:** initial value;

while(expression)

{ statement1;

statement2;

changing value;

}

**do .. while loop:**

It is also called as a condition loop. In this loop the statements of a loop will execute first and then the condition will be verified. This loop will be executed for repeated number of times only when the condition is true. In this loop the statements will executes once even though the condition is false.

**Syntax:** initial value;

do

{ statement1;

statement2;

changing value;

} while (expression);

**2Q) Write about jumping Statements?**

**Ans:** There are three different controls used to jump from one c program statement to another and make the execution of the programming procedure fast. These three jumping controls are

1. **goto statement**
2. **break statement**
3. **continue statement**

**goto statement**

The powerful jumping statement in the C language is goto statement. It is sometimes also called part of branching statement. The goto moves the controls on a specified address called label or label name. The goto is mainly of two types. One is conditional and the other is unconditional. Also jump can be either in forward direction or in backward direction. The different types of goto statement is

1. **forward goto**
2. **backward goto**

**Forward goto:**

In this the control moves forward at a specified label either according to a condition or without condition.

**Syntax for unconditional forward goto:**

Statement1;

Statement2;

**Goto label;**

Statement3;

Statement4;

**Label:**

Statement5;

Statement6;

**Syntax for conditional forward goto:**

Statement1;

Statement2;

**If(condition)**

**Goto label;**

Statement3;

Statement4;

**Label:**

Statement5;

Statement6;

**Note:**

Here first statement1 and statement2 will be executed, then specified label and execute the statements statement5 and statement6. It will skip a part of the program i.e. statement3 and statement4. In c-language label name is either a single character or combination of characters.

**Backward goto:**

The backward goto or backward jump moves the control back to the specified addresses and so creates a loop. In the case of conditional backward statement, it creates finite looping. But in the case of unconditional backward goto or jump, it creates infinite looping.

**Syntax for unconditional backward goto statement:**

Statement1;

**Label:**

Statement2;

Statement3;

**Goto label;**

Statement4;

**Syntax for conditional backward goto statement:**

Statement1;

**Label:**

Statement2;

Statement3;

**if(condition)**

Goto label;

Statement4;

**Note:**

Here first of all statements statement1, statement2 and statement3 will be executed. Then it will find a backward goto statement with a specified address and again statement2, statement3 statement will repeat either according to a condition or infinite times if there is no condition.

**Break**

* A break statement terminates the execution of the loop and the control is transferred to the statement immediately following the loop i.e, the break statement is used to terminate loops or to exit from a switch
* It can be used within a for, while, do-while or switch statement.
* The break statement is written simply as **break;**

**Syntax:** while(condition)

{

statement1;

statement2;

if(condition)

{

break;

}

statement-x;

}

**Continue statement:**

* The continue statement is used to bypass the remainder of the current pass through a loop. The loop does not terminate when a continue statement is encountered
* Instead the remaining statements are skipped and the computation proceeds directly to the next pass through the loop.
* The continue statement can be included within a while, a do-while, a for statement
* It is simply written as “continue”. The continue statement tells the compiler “skip the following statements and continue with the next iteration.
* In “while” and “do-while” loops continue causes the control to go directly to the test condition and then to continue the iteration process

**Syntax:**

while(condition)

{

Statement1;

Statement2;

if(condition)

{

Continue;

}

Statement3;

Statement4;

}

Statement-x;

**Note:** In the above structure, first of all the statements statement1 and statement2 will be executed. When a condition met within the loop and if the condition becomes true, then the continue statement be executed and it will move to start the next looping iteration by skipping the statements statement3 and statement4. When condition becomes false, then the statements statement3 and statement4 will execute.

**3Q) What is Array? Explain different types of arrays?**

**Ans:**

A collection of values of similar types which can be stored under one variable or heading it is called as an array. It is also called as homogenous type.

**TYPES OF ARRAY:**

Arrays are of different types. They are –

1. Single Dimension Array
2. Double Dimension Array

**SINGLE DIMENSION ARRAY:**

An array which contains only one row and it can have set of columns. It is called as a single dimension array. It is also called as **one dimensional array** or **single subscripted value array**. Array position always starts from **“zero”**. Each section of an array is called as an **“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 arrayName [ arraySize ];

## Initializing Arrays

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

Eg: - 1. int a[4]= { 10,20,30,40 };

a[0] = 10

a[1] = 20

a[2] = 30

a[3] = 40

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 −

2. int x[] ={10,20,30,40};

x[0] = 10

x[1] = 20

x[2] = 30

x[3] = 40

**DOUBLE DIMENSION ARRAY:**

An array which contains number of rows and number of columns, it is called as a double dimension array. It is also called as Double subscripted value array.

***Syntax:*** Datatype array\_name[rs][cs];

**Eg:- 1**. int a[3][3];

a[0][0] a[0][1] a[0][2]

a[1][0] a[1][1] a[1][2]

a[2][0] a[2][1] a[2][2]

**Initialization of values in array:**

int a[][]= {{10,20,30}, {40,50,60}, {70,80,90}}

a[0][0]=10 a[0][1]=20 a[0][2]=30

a[1][0]=40 a[1][1]=50 a[1][2]=60

a[2][0]=70 a[2][1]=80 a[2][2]=90

**4Q) Write the Advantages and Disadvantages of Arrays?**

**Ans:**

**Advantages:**

1. It is better and convenient way of storing the data of same datatype with same size.
2. It allows us to store known number of elements in it.
3. It allocates memory in contiguous memory locations for its elements. It does not allocate any extra space/ memory for its elements. Hence there is no memory overflow or shortage of memory in arrays.
4. Iterating the arrays using their index is faster compared to any other methods like linked list etc.
5. It allows to stores the elements in any dimensional array - supports multidimensional array.

**Disadvantages:**

1. Array is **Static data** Structure
2. Memory Allocated during **Compile time**.
3. It allows us to enter only fixed number of elements into it. We cannot alter the size of the array once array is declared. Hence if we need to insert more number of records than declared then it is not possible. We should know array size at the compile time itself.
4. Inserting and deleting the records from the array would be costly since we add / delete the elements from the array, we need to manage memory space too.

## 5Q) Write the Basic Operations on Arrays?

## Ans: Following are the basic operations supported by an array.

* 1. **Traverse** − print all the array elements one by one.
  2. **Insertion** − Adds an element at the given index.
  3. **Deletion** − Deletes an element at the given index.
  4. **Search** − Searches an element using the given index or by the value.
  5. **Update** − Updates an element at the given index.

## Insertion Operation

## Insert operation is to insert one or more data elements into an array. Based on the requirement, a new element can be added at the *beginning*, *end*, *or any given index of array.*

### Example

**#include<stdio.h>**

**#include<conio.h>**

main()

{ int a[] = {1,3,5,7,8}, item = 10, k = 3, n = 5, i = 0, j = n;

clrscr();

printf("The original array elements are :\n");

for(i = 0; i<n; i++)

printf("a[%d] = %d \n", i, a[i]);

n = n + 1;

while( j >= k)

{

a[j+1] = a[j];

j = j - 1;

}

a[k] = item;

printf("The array elements after insertion :\n");

for(i = 0; i<n; i++)

printf("a[%d] = %d \n", i, a[i]);

}

**Output**

**The original array elements are :**

**a[0] = 1**

**a[1] = 3**

**a[2] = 5**

**a[3] = 7**

**a[4] = 8**

**The array elements after insertion :**

**a[0] = 1**

**a[1] = 3**

**a[2] = 5**

**a[3] = 10**

**a[4] = 7**

**a[5] = 8**

1. **Deletion Operation**

Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

**Example**

**#include <stdio.h>**

main()

{

int LA[] = {1,3,5,7,8}, k = 3, n = 5, i, j;

printf("The original array elements are :\n");

for(i = 0; i<n; i++)

printf("LA[%d] = %d \n", i, LA[i]);

j = k;

while( j < n)

{

LA[j-1] = LA[j];

j = j + 1;

}

n = n -1;

printf("The array elements after deletion :\n");

for(i = 0; i<n; i++)

printf("LA[%d] = %d \n", i, LA[i]);

}

**Output**

The original array elements are :

LA[0] = 1

LA[1] = 3

LA[2] = 5

LA[3] = 7

LA[4] = 8

The array elements after deletion :

LA[0] = 1

LA[1] = 3

LA[2] = 7

LA[3] = 8

1. **Search Operation:**

You can perform a search for an array element based on its value or its index.

**Example**

#include <stdio.h>

main()

{

int LA[] = {1,3,5,7,8}, item = 5, n = 5, i = 0, j = 0;

clrscr();

printf("The original array elements are :\n");

for(i = 0; i<n; i++)

printf("LA[%d] = %d \n", i, LA[i]);

while( j < n)

{

if( LA[j] == item )

break;

j = j + 1;

}

printf("Found element %d at position %d\n", item, j+1);

}

**Output**

The original array elements are:

LA[0] = 1

LA[1] = 3

LA[2] = 5

LA[3] = 7

LA[4] = 8

Found element 5 at position 3

1. **Update Operation:**

Update operation refers to updating an existing element from the array at a given index.

**Example**

#include <stdio.h>

main()

{ int LA[] = {1,3,5,7,8},i,j,k = 3, n = 5, item = 10;

clrscr();

printf("The original array elements are :\n");

for(i = 0; i<n; i++)

printf("LA[%d] = %d \n", i, LA[i]);

LA[k-1] = item;

printf("The array elements after updation :\n");

for(i = 0; i<n; i++)

printf("LA[%d] = %d \n", i, LA[i]);

}

**Output**

The original array elements are :

LA[0] = 1

LA[1] = 3

LA[2] = 5

LA[3] = 7

LA[4] = 8

The array elements after updation :

LA[0] = 1

LA[1] = 3

LA[2] = 10

LA[3] = 7

LA[4] = 8

**UNIT – III**

**1Q) Write about Character array?**

**Ans:** A collection of characters which are organized under one variable, it is called as a character arrays. In a character array when we initialize any string by default it contains NULL value at the last of character array.

**Syn: datatype var[size];**

**Eg:-** char s[10];

**Initialization** **of string in Character array**

char s[10]={"helo"}

s[0]=’h’

s[1]=’e’

s[2]=’l’

s[3]=’o’

s[4]='\0'

**Character I/O Functions:**

The character i/o functions are -

* + getchar()
  + getche()
  + getch()
  + putchar()

1. **getchar()**

* It is a character input function.
* This function is used to input a line of text but it can able to store only the starting character into a memory.

**Syntax:** var=getchar()

1. **getche()**:

* It is a character input function.
* This function is used input only one character and it stores into a memory.

**Syntax:** var=getche()

1. **getch()**

* It is a character input function.
* This function is used to input only one character but it will never echoes on to a screen but it stores into a memory.

**Syntax:** var=getch()

1. **putchar()**

* It is a character output function.
* This function is used to display a character as output.

**Syntax:** putchar(var);

**2Q) Write about String handling functions**

**Ans:** String handling functions are available in a header file called string.h. The string handling functions are

1. **Strcmp():**

This function is used to compare two different strings and find whether they are exact or not. If they are exact it returns 0 otherwise it returns -1.

**Syn:** strcmp(string1,string2)

**Eg:** strcmp(“hello”,”Hello”); => -1 (false)

Strcmp(“hello”,”hello”) => 0 (true)

1. **Strcat():**This function is used to combine two strings into one.

**Syn:** strcat(string1,string2)

**Eg:** strcat(“hello”,”world”) => helloworld

1. **strrev():** This function is used to reverse the given string.

**Syn:** strrev(string)

**Eg:** strrev(“hello”) =>olleh

1. **Strlen():** This function is used to find the length of a given string

**Syn:** strlen(string)

**Eg:** strlen(“hello”) => 5

1. **Strupr():** This function is used to convert a given string into upper case.

**Syn:** strupr(string)

**Eg:** strupr(“hello”) =>HELLO

1. **Strlwr():** This function is used to convert a given string into lower case.

**Syn:** strlwr(“string”)

**Eg:** Strlwr(“HELLO”) =>hello

**3Q) What is function? Write its advantages?**

**Ans:**

A block of statements which performs a particular task it is called as a function. A sub-program is independent because it can be called by the main program or other sub-program.

**Advantages:**

The length of source program can be reduced by using function by using it at different places in the program according to the user’s requirement.

By using function, memory space can be properly utilized. Also less memory is required to run program if function is used.

A function can be used by many programs.

Functions increases the execution speed of the program and makes the programming simple.

By using the function, portability of the program is very easy.

Debugging becomes very easier and fast using the function sub-programming.

Function are more flexible than library functions

Testing is very easy by using functions.

Reliability is high in the function programming

**4Q) Write about the types of functions?**

**Ans:** Functions are of different types. They are –

1. Library Functions
2. User-defined Functions

**Library Functions**

The functions which are already existed in c-language is known as library functions. The Library functions are

**Eg**:- Printf( ), scanf( ), strlen( ), strcat( ), sqrt( )., etc.

**User-defined functions**

The function which creates by the user is called as user-defined functions. The user-defined functions are –

* 1. Function with no arguments no return values.
  2. Function with arguments but no return values.
  3. Function with arguments and return value
  4. Function with no arguments but return value.

1. **Function with no arguments no return value**

A function doesn’t contains any arguments, while calling a function, a called function will performs is task and it will never returns a value to a calling function.

**Syntax:** return\_type function\_name()

{

statement1;

statement2;

}

1. **Function with arguments no return value**

A function contains arguments and while calling a called function the arguments can be passed from calling function to called function, this passing of arguments from calling function to called function is called as **call by value.**

A calling function and a called function should have the same types of arguments but the argument names of calling function and called function may be same or not similar. If the names of the arguments in both calling and called function are same then such arguments can be called as **actual arguments.**

If the names of the arguments in both calling and called function are not same then such arguments can be called as **formal arguments or dummy arguments or parameterized arguments.**

The arguments passed to a called function can be make use to perform required task and the resultant value will not be returned to a calling function.

**Syntax: void main()**

**{**

statement1;

statement2;

**function();----> calling function**

**}**

**function() --> called function**

**{**

statement1;

statement2;

**}**

1. **Function with arguments and return value**

A function contains arguments and while calling a called function the arguments can be passed from calling function to called function, this passing of arguments from calling function to called function is called as **"call by value".**

A calling function and a called function should have the same types of arguments but the argument names of calling function and called function may be same or not similar. If the names of the arguments in both calling and called function are same then such arguments can be called as **actual arguments.**

If the names of the arguments in both calling and called function are not same then such arguments can be called as **formal arguments or dummy arguments or parameterized arguments.**

The arguments passed to a called function can be make use to perform required task and the resultant value can be returned to a calling function by using the statement **"return"**.

**Syntax: void main()**

**{**

**statement1;**

**statement2;**

**statement3;**

**var=function(arg1,arg2);----> calling function**

**statement4;**

**}**

**returntype function(arg1,arg2) --> called function**

**{**

**statement1;**

**statement2;**

**statement3;**

**return value;**

**}**

1. **Function with no arguments and no return value**

A function contains arguments, while calling a function a called function will executes its task and it will return a value to a calling function.

**Syn:** returntype function\_name( )

{

statement1;

statement2;

**statement3;**

return value;

}

**5Q) Write about Passing array to function?**

**Ans:**

Array plays an important role in the function handling. Arrays are declared with the function definition and so can be used in the function declaration statement.

**Passing one-dimensional array to function:**

For the single dimension array, the general syntax used for declaring function is

Function\_name(array\_name, array-size)

For example, **passing one-dimensional array values to the function** subprogram can be illustrated in a valid statement as below.

Sort(a,n);

Here **“a”** is an array defined as: int a[20]; and n be the size of array.

**/\*Eg program to pass array to function\*/**

#include<stdio.h>

#include<conio.h>

main()

{

int n,a[100],i;

clrscr();

printf("\n Enter N value");

scanf("%d",&n);

printf("\n Enter %d values in array",n);

for(i=0;i<n;i++)

scanf("%d",&a[i]);

sort(a,n);

printf("\n After sorting");

for(i=0;i<n;i++)

printf("\n a[%d]=%d",i,a[i]);

getch();

}

**sort(int a[100],int n)**

{

int i,j,t;

for(i=0;i<n;i++)

{

for(j=i+1;j<n;j++)

{

if(a[i]>a[j])

{

t=a[i];

a[i]=a[j];

a[j]=t;

}

}

}

**Passing two dimensional array to function:**

For the double dimension array, the general syntax used for the declaration statement be as:

Function\_name(array\_name, row-size,column-size)

For example, **passing one-dimensional array values to the function** subprogram can be illustrated in a valid statement as below.

Trace(a,n,m);

Here **“a”** is an array defined as: int a[10][10]; and n be the row size of array and m be the column size of array.

**Eg**

#include<stdio.h>

#include<conio.h>

main()

{

int n,a[10][10],i,j;

clrscr();

printf("\n Enter N value");

scanf("%d",&n);

printf("\n Enter %d x%d values in array",n,n);

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%d",&a[i][j]);

}

}

trace(a,n);

getch();

}

trace(int a[10][10],int n)

{

int i,j;

printf("\n Trace Matrix \n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

printf("%d ",a[j][i]);

}

printf("\n");

}

}

**6Q) Write about passing structure to function?**

**Ans:**

* A structure can be passed to any function from main function or from any sub function.
* Structure definition will be available within the function only.
* It won’t be available to other functions unless it is passed to those functions by value or by address(reference).
* Else, we have to declare structure variable as global variable. That means, structure variable should be declared outside the main function. So, this structure will be visible to all the functions in a C program
* Passing structure to function can be done in 3 ways

1. Passing structure to a function by value
2. Passing structure to a function by address(reference)
3. No need to pass a structure – Declare structure variable as global
   * 1. **PASSING STRUCTURE TO FUNCTION IN C BY VALUE:**

It means the whole structure is passed to another function with all members and their values. So, this structure can be accessed from called function. This concept is very useful while writing very big programs in C.

**Eg:**

#include <stdio.h>

#include <string.h>

struct student

{

int id;

char name[20];

float percentage;

};

main()

{

struct student record;

record.id=1;

strcpy(record.name, "Raju");

record.percentage = 86.5;

display\_studentinfo(record);

getch();

}

void display\_studentinfo (struct student record)

{ printf(" Id is: %d \n", record.id);

printf(" Name is: %s \n", record.name);

printf(" Percentage is: %f \n", record.percentage);

}

**OUTPUT:**

Id is: 1

Name is: Raju

Percentage is: 86.500000

* + 1. **PASSING STRUCTURE TO FUNCTION IN C BY ADDRESS:**

It means only the address of the structure is passed to another function. The whole structure is not passed to another function with all members and their values. So, this structure can be accessed from called function by its address.

**Eg:**

#include <stdio.h>

#include <string.h>

struct student

{

int id;

char name[20];

float percentage;

};

main()

{

struct student record;

record.id=1;

strcpy(record.name, "Raju");

record.percentage = 86.5;

display\_studentinfo (&record);

getch();

}

void display\_studentinfo (struct student \*record)

{

printf(" Id is: %d \n", record->id);

printf(" Name is: %s \n", record->name);

printf(" Percentage is: %f \n", record->percentage);

}

**OUTPUT:**

Id is: 1

Name is: Raju

Percentage is: 86.500000

* + 1. **TO DECLARE A STRUCTURE VARIABLE AS GLOBAL IN C:**

Structure variables also can be declared as global variables as we declare other variables in C. So, When a structure variable is declared as global, then it is visible to all the functions in a program. In this scenario, we don’t need to pass the structure to any function separately.

**Eg:**

#include <stdio.h>

#include <string.h>

struct student

{ int id;

char name[20];

float percentage;

};

struct student record; // Global declaration of structure

main()

{

record.id=1;

strcpy(record.name, "Raju");

record.percentage = 86.5;

structure\_demo();

getch();

}

void structure\_demo()

{

printf(" Id is: %d \n", record.id);

printf(" Name is: %s \n", record.name);

printf(" Percentage is: %f \n", record.percentage);

}

**OUTPUT:**

Id is: 1

Name is: Raju

Percentage is: 86.500000

**7Q) Write about Recursive function?**

**Ans:**

**“A function which can be called by itself for repeated number of times, it is called as a recursive function”.** Recursive functions are used to create data structures like lists, stacks, queues, etc. **Recursion is also called self-reference loop, then it is called recursion.**

**Advantages of Recursion:**

1. It is a simple, easily understandable, concise, compact and transparent to view a c program.
2. Lesser number of programming statements required with the use of recursion.
3. It is useful in solving mathematical, trigonometric, logical games and algebraic problems.
4. It is eminently useful if the solution to problem is in recursive terms.
5. It is more useful in the multiprocessing and multitasking environment.
6. It is very useful in solving the data structure problems like linked list, queries, stack, tree, quick sort and merge sort etc.
7. Recursion saves the memory. In other words it utilizes the memory well.

**Disadvantages of functions:**

1. It requires more memory to store the automatic variable to solve the problems like stack and so waste the memory space. i.e. to consume more storage space.
2. If properly recursion procedure is not checked, then it will create a problem for the processing and procedure run out of memory.
3. In some problems, it is a time consuming process and is not efficient.
4. It will create indefinite looping process, if condition not be applied at the proper place.

**/\* Eg: recursive function to find factorial of given number \*/**

#include<stdio.h>

#include<conio.h>

main()

{

int n,x;

clrscr();

printf("\n Enter N value");

scanf("%d",&n);

x=factorial(n);

printf("\n Factorial = %d",x);

getch();

}

factorial(int n)

{

int s=1;

if(n==0)

s=1;

else

s=n\*factorial(n-1);

return s;

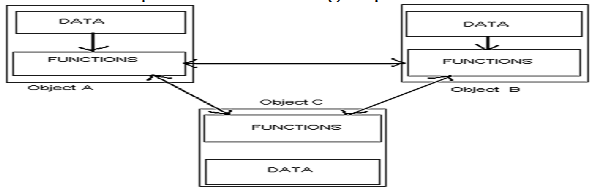
}

**UNIT - IV**

**1Q) Write about Object Oriented Programming?**

**Ans:**

* The major motivating factor in the invention of object oriented approach is to remove some of the flaws encountered in the procedural approach.
* OOP treats data as a critical element in the program development and does not allow it to flow freely around the system.
* It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions.
* OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.
* Object oriented programming as an approach that provides a way of modularizing programs by creating memory area for both data and functions that can be used as templates for creating copies of such modules on demand.



**2Q) Write the concepts of OOP**

**Ans:** The Concepts of OOP are

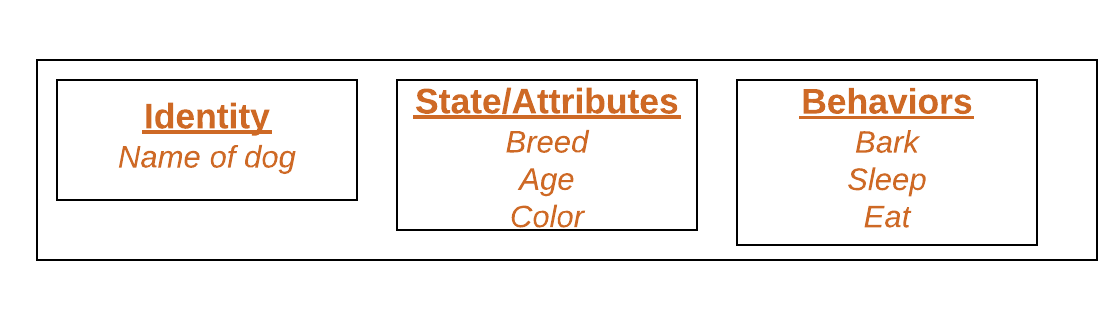
1. object
2. class
3. Data Abstraction
4. Data Encapsulation
5. Polymorphism
6. inheritance
7. Message passing / communication
8. Dynamic Binding

**Object**

* It is a basic unit of OOP and represents the real life entities.
* It is an instance of class
* A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:

1. **State**: It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behavior**: It is represented by methods of an object. It also reflects the response of an object with other objects.
3. **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

**Example of an object**: dog

[](https://media.geeksforgeeks.org/wp-content/uploads/Blank-Diagram-Page-1-5.png)

**Class**

* Collection of objects is called as a class
* A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.
* A class is the way to bind data and its associated functions together. It allows the data to be hidden if necessary from the external use. When defining a class we are creating a new abstract data type that can be treated like any other built in data type.

**Syn: Class <class\_name>**

**{**

**member1;**

**member2;**

**<returntype> method1([arglist])**

**{**

**Block of statements;**

**}**

**};**

**Data Abstraction**

* Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes and functions to operate on these attributes.
* It is a technique of creating a new data type that is suited for a specific application. **For example**, while driving a car, you do not have to be concerned with its internal working. Here you just need to concern about parts like steering wheel, Gears, accelerator, etc.

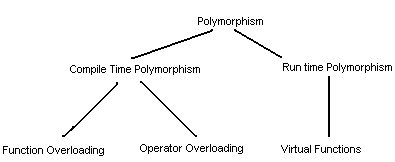
**Data Encapsulation**

The wrapping up of data and functions into a single unit is known as Data encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object’s data the program. This insulation of the data from direct access by the program is called data hiding or information hiding.

**Polymorphism**

Polymorphism is a combination of two Greek words called **poly** and **morphism**. **Poly** means **many** and **morphism** means **forms** and thus **polymorphism means many forms**. In object oriented programming, polymorphism refers to identically named methods that have different behavior depending on the type of object they refer.

Polymorphism is the process of defining, a number of objects of different classes into a group and call the methods to carry out the operation of the objects using different function calls.



**Types of Polymorphism:**

Polymorphism is classified into two types. They are

1. Compile Time Polymorphism
2. Run time Polymorphism

**Compile Time Polymorphism**

Choosing a function in normal way, during compilation time is called as **early binding** or **static binding** or **static linkage.** During compilation time, the compiler determines which function is used based on the parameters passed to the function or the function’s return type. The compiler then substitutes the correct function for each invocation. Such compiler based substitutions are called static linkage.

With early binding, one can achieve greater efficiency. Function calls are faster in this case because all the information necessary to call the function are hard coded.

Polymorphism provides two different functionalities. They are

1. **Function Overloading**

A function with identical name can be defined for any number of times whereas their arguments or return type should differ, it is called as function overloading.

1. **Operator Overloading**

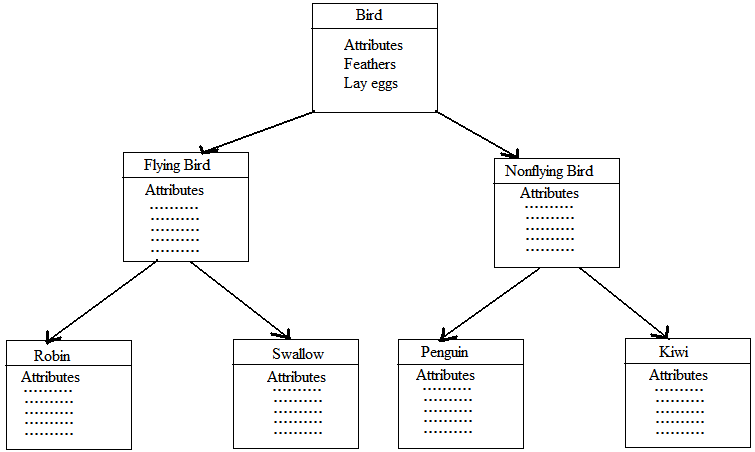
It is defined as to redefine the functionality of operators, while redefining the functionality of operators its meaning and order of precedence should not be changed.

**Run time Polymorphism**

Choosing functions during execution time is called the **late binding** or **dynamic binding** or **dynamic linkage.** Late binding requires some overhead but provides increased power and flexibility. The late binding is implemented through virtual functions. An object of a class must be declared either as a pointer to a class or a reference to a class.

**Inheritance**

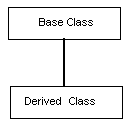
Inheritance is the process by which objects of one class acquire the properties of object of another class. It supports the concept of hierarchical classification. In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes.



Inheritances are of different types. They are -

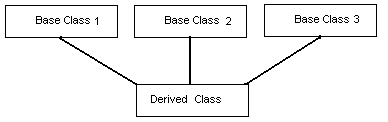
1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hybrid inheritance
5. Hierarchical inheritance
6. Multi-path Inheritance
7. **Single inheritance**

A class which can able to create another class, it is called as single inheritance. It means the features or properties of a base class can be inherited to a child class. **A parent class is also called as a base class** and a **child class is also called as a derived class**.



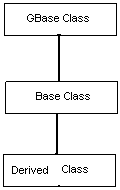
1. **Multiple inheritances**

A class which inherits the properties from different parent classes, it is called as multiple inheritance. It means the properties of different base classes can be inherited to a child class



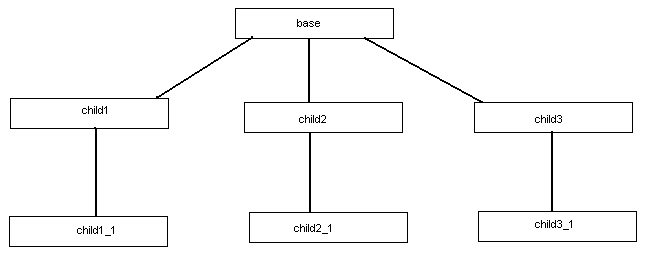
1. **Multi-level inheritance**

A class which can be created in sequence that means a class can contains the properties of grand base and parent classes it is called as multi-level inheritance. A class is derived from another derived class it is called multi-level inheritance.



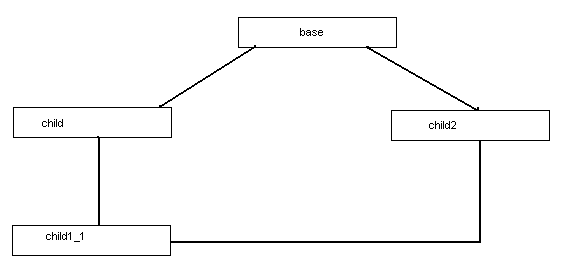
1. **Hierarchical inheritance**

The base class includes all the properties that are common to the subclasses. Subclasses can be constructed by inheriting the properties of the base class. A subclass can serve as a base class for the lower level classes and so on. This process is called hierarchical inheritance.

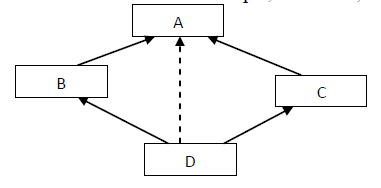


1. **Hybrid Inheritance**

A class which inherits the properties from the combination of multiple parent classes, it is called as hybrid Inheritance. It means hybrid inheritance is the combination of multi-level and multiple inheritances.



1. **Multipath inheritance**
   * Multipath inheritance in C++ is derivation of a class from other derived classes, which are derived from the same base class. In this type of inheritance, there involves other inheritance like multiple, multilevel, hierarchical etc.
   * It is famously known as diamond problem in computer programming.



* Here, class D is derived from derived classes B & C directly and from class A indirectly. (hierarchical and multiple)
* Both derived classes inherits the features of base class. Hence when we derive a new class by inheriting features form these two classes derived from the same base class, then same features from the first base is inherited to the finally derived class from two paths. This cause ambiguity in accessing first base class members.

**Message passing**

An object-oriented program consists of a set of objects that communicate with each other. The process of programming is an object-oriented language, therefore, involves the following basic steps

* Creating classes that define objects and their behavior
* Creating objects from class definitions
* Establishing communication among objects

**Dynamic Binding**

Binding refers to the tie-up of a procedure call to the addressed code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until its call at run-time. A function call associated with a polymorphic reference depends on the dynamic type of that reference.

**3Q) WRITE THE FEATURES OF OOP?**

**Ans:**

1. Emphasis is on data rather than procedure.
2. Programs are divided into what are known as objects.
3. Data structures are designed such that they characterize the objects.
4. Functions that operate on the data of an object are tied together in the data structure
5. Data is hidden and cannot be accessed by external functions.
6. Objects may communicate with each other through functions.
7. New data and functions can be easily added whenever necessary
8. Follows bottom-up approach in program design

**4Q) Structure of C++ Program**

**Ans:** A program consists of different sections. Some of these are optional and can be excluded from the program if not required. The structure of C++ program consists of following sections

1. Documentation Section
2. Include file section
3. Class declaration or definition
4. Class function Definitions
5. main() function

**Documentation Section**

**The documentation section** consists of a set of comment lines giving the name of the program, the author and other details. Comments can be written in two different ways

1. // Single line comment
2. /\* Paragraph Comment \*/

**Include File Section**

***The* include files section** provides instructions to the compiler to include code from the system library in the form of header files. Each header file has an extension of .h. The file should be included using #include directive as per the format given below

#include<filename.h>

Or

#include “filename.h”

**Class declaration or Definition**

***Declaration of a class is done in this section.*** It is also possible to declare class after the function main(), but it is better to declare it before main() function. A class contains data members, function prototypes. The class definition is always terminated by a semi-colon.

**Class function Definitions**

***This part contains definition of functions.*** The function definition can be done outside or inside the class. The functions defined inside the class are implemented as inline functions. When a function is large, it is defined outside the class. In this case, prototype of a function is declared inside the class.

**main() function**

Since every C++ program requires a main method as its starting point. It is the essential part of a C++ program. The main method creates objects of various classes and establishes communications between them.

**5Q) Write about friend functions and friend classes?**

**Ans:**

The main concept of object oriented programming is data hiding and data encapsulation. Whenever data variables are declared in a private category of a class, these members are restricted from accessing by non-member functions. The private data values can be neither read nor written by non-member functions. If any attempt is made directly to access these members, the compiler will display an error message as “**inaccessible data type**”.

The best way to access a private data member by a non-member function is to change a private data member to a public group. When the private or protected data member is changed to a public category, it violates the whole concept of data hiding and data encapsulation. To solve this problem, a friend function can be declared to have access to these data members. Friend is a special mechanism for letting non-member functions access private data. A friend function may be either declared or defined within the scope of a class definition. The keyword friend informs the compiler that it is not a member function of the class.

**Syn:**  class classname

{

private: member1;

member2;

friend returntype function(arg);

public :

method1();

method2();

friend returntype function(arg);

};

**// Program to perform friend functions**

#include<iostream.h>

#include<iomanip.h>

class sample

{

private : int a,b;

public : void accept();

friend void display(sample x);

};

void sample :: accept()

{

cout<<"Accept A and B values";

cin>>a>>b;

}

void display(sample x)

{

cout<<" A value ="<<x.a<<endl;

cout<<" B value ="<<x.b<<endl;

}

void main()

{ sample ob;

clrscr();

ob.accept();

display(ob);

getch();

}

**Friend class**

A class can have friendship with one or more classes. If the class first grants its friendship with the other class second, then the private data members of the class first are permitted to be accessed by the public members of the class second. But on the other hand, the public member functions of the class first cannot access the private members of the class second.

**Syntax** – class first

{

friend class second;

private : Member1;

Member2;

public : Method1( );

Method2( );

};

class second

{

public : Method(arg);

};

**// Granting friendship with another class**

#include<iostream.h>

class first

{ friend class second;

private: int x;

public : void accept( );

};

class second

{

public : void display(first temp);

};

void first :: accept( )

{

cout<<”Enter x value”;

cin>>x;

}

void second :: display(first temp)

{

cout<<<”X value =”<<temp.x<<endl;

}

void main( )

{

First ob1;

Second ob2;

ob1.accept( );

ob2.display(ob1);

}

**6Q) Write about Constructor?**

**Ans:**

A constructor is a special type of member function which is used to initialize values. A constructor need not be called explicitly by the user, it can be called automatically during the creation of an object. To create a constructor the following rules to be followed. They are –

1. The constructor name should be same as that of class name.
2. A constructor may or may not have arguments.
3. A constructor should not have any return types even void also.
4. A constructor should not be defined as static

**Types of Constructors:**

**Constructors are of 2 types. They are –**

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor

**Default Constructor**

The default constructor is a special member function which is invoked by the C++ compiler without any argument for initializing the objects of a class. In other words, a default constructor function initializes the data members with no arguments. It may be explicitly written in a program.

**Syn:**  class class\_name

{

Private : Member1;

Member2;

Protected : Member1;

Member2;

Public : class\_name( ); // default constructor

Method1( );

Method2( );

};

**Parameterized Constructor**

A constructor with one or more arguments can be called as parameterized constructor. It is used to initialize the members of a class. It cannot be called explicitly by the user. It can be called automatically when we create an object by passing arguments at the time of creation of object.

**Syntax**

Class class\_name

{

Private: Member1;

Member2;

Protected: Member1;

Member2;

Public : class\_name(arg1,arg2 ); // parameterized constructor

Method1( );

Method2( );

};

**Constructor Overloading**

A constructor with identical name can be defined for any number of times where as its arguments should differ. It is called as constructor over loading. The invocation of constructors always depends on the creation of an object.

**Syntax**

Class class\_name

{

Private : Member1;

Member2;

Public : Class\_name( ); //default constructor

Class\_name(arg1,arg2);//Parameterized constructor

Return\_type Method1( );

};

**Copy Constructors**

Copy constructors are always used when the compiler has to create a temporary object of a class object. The copy constructors are used in the following situations.

* The initialization of an object by another object of the same class.
* Return of objects as a function value.
* Stating the object as by value parameters of a function.

**Syntax**

Class class\_name

{

private : Member1;

Member2;

public : Class\_name( );

Class\_name(class\_name &var);

Method1( );

};

**7Q) Write about Destructors?**

**Ans:**

A destructor is a special type of member function which is used to clear the heap of a memory. A destructor need not be called by the user. It can be called automatically after the completion of an object. To create a destructor the following rules to be followed. They are –

1. A destructor name should be same as that of class name but prefix to destructor a tild operator should be specified.
2. A destructor should not have any arguments.
3. A destructor should not have any return types even void also
4. A destructor should not be defined as static.

**Syntax:** Class class\_name

{

Private : Member1;

Member2;

Public : Class\_name( ); // constructor

~ Class\_name(); // destructor

Method1( );

};

**// program to demonstrate destructor**

#include<iostream.h>

#include<conio.h>

class sample

{

private: int a,b;

public: sample()

{

a=100;

b=200;

}

~sample()

{ Cout<<”Memory is cleared”; }

void display()

{

cout<<”A value=”<<a<<endl;

cout<<”B value=”<<b<<endl;

}

};

void main()

{ sample x;

clrscr();

x.display();

getch();

}

**UNIT - V**

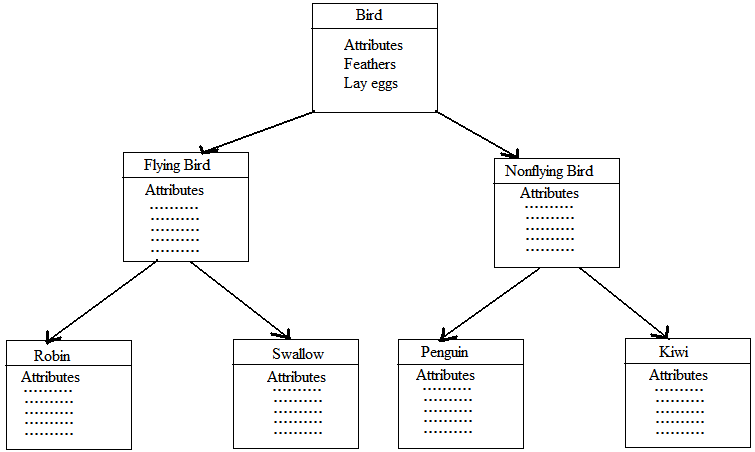
**1Q) Write about Inheritances?**

**Ans:**

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

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

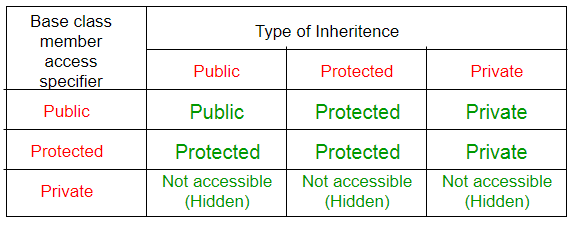
The concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes.



**Modes of Inheritance**

1. **Public mode**: If we derive a sub class from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in derived class.
2. **Protected mode**: If we derive a sub class from a Protected base class. Then both public member and protected members of the base class will become protected in derived class.
3. **Private mode**: If we derive a sub class from a Private base class. Then both public member and protected members of the base class will become Private in derived class.

The below table summarizes the above three modes and shows the access specifier of the members of base class in the sub class when derived in public, protected and private modes:



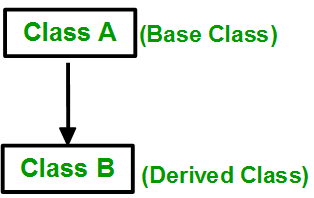
**Types of Inheritances**

**C++ supports different types of inheritances. They are**

* + - 1. Single Inheritance
      2. Multiple Inheritance
      3. Multi-level inheritance
      4. Hybrid Inheritance
      5. Hierarchical Inheritance
      6. Multi-path Inheritance.

**Single Inheritance:**

In single inheritance, a class is allowed to inherit from only one class. i.e. one sub class is inherited by one base class only.



**Syntax**:

Class Base

{

private: Member1;

Member2;

public:

Method1();

Method2();

};

Class derived: **access\_mode** Base

{

private: Member1;

Member2;

public:

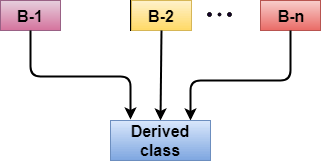
returntype Method1();

returntype Method2();

};

**Multiple inheritance:**

A class which can be derived from more than one base class, it is called as multiple inheritance



**Syntax**:

Class Base1

{

private: Member1;

Member2;

public:

returntype Method1();

returntype Method2();

};

Class Base2

{

private: Member1;

Member2;

public:

returntype Method1();

returntype Method2();

};

Class derived: **access\_mode** Base1, **access\_mode** Base2

{

private: Member1;

Member2;

public:

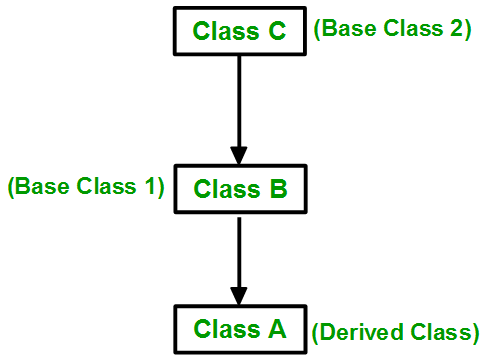
returntype Method1();

returntype Method2();

};

**Multi-level 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.



**Syntax**:

Class gBase

{

private: Member1;

Member2;

public: returntype Method1();

returntype Method2();

};

Class Base : **access\_mode** gBase

{

private: Member1;

Member2;

public: returntype Method1();

returntype Method2();

};

Class gchild: **access\_mode** Base

{

private: Member1;

Member2;

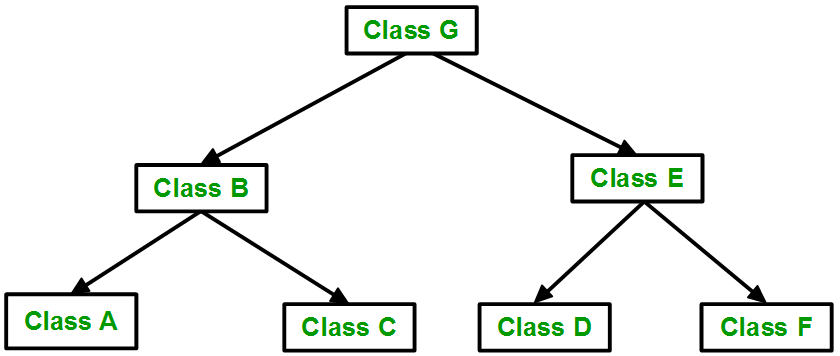
public: returntype Method1();

returntype Method2();

};

**Hierarchical Inheritance**:

In this type of inheritance, more than one sub class is inherited from a single base class. i.e. more than one derived class is created from a single base class.



**Hybrid (Virtual) Inheritance**:

* Hybrid Inheritance is implemented by combining more than one type of inheritance.
* **For example:** Combining Hierarchical inheritance and Multiple Inheritance.   
  Below image shows the combination of hierarchical and multiple inheritance:

