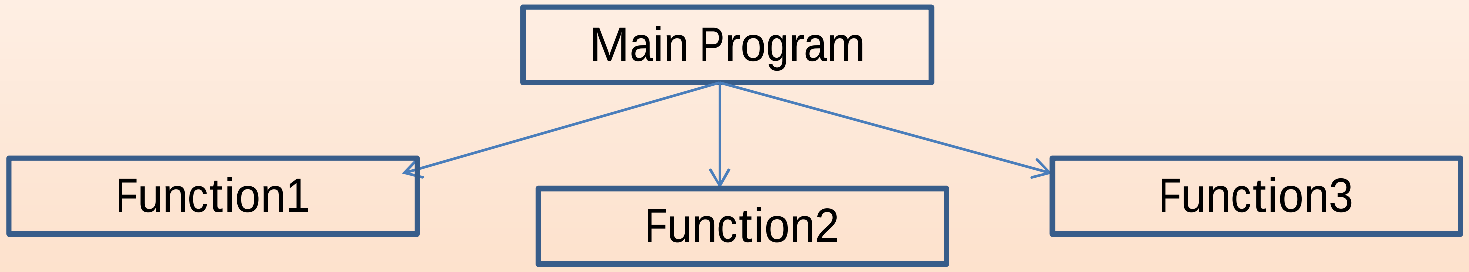
C++ and Object Oriented Programming

Unit – 1 (Topic – 1)

Principles of object oriented programming
Tokens, expressions and control statements

Procedure Oriented Programming

- In Procedure Oriented Programming(POP), the problem is viewed as a sequence of things to be done such as reading, calculating and printing (Just Like an Algorithm).
- It is accomplish with functions.
- A typical program structure for POP is shown in below fig.



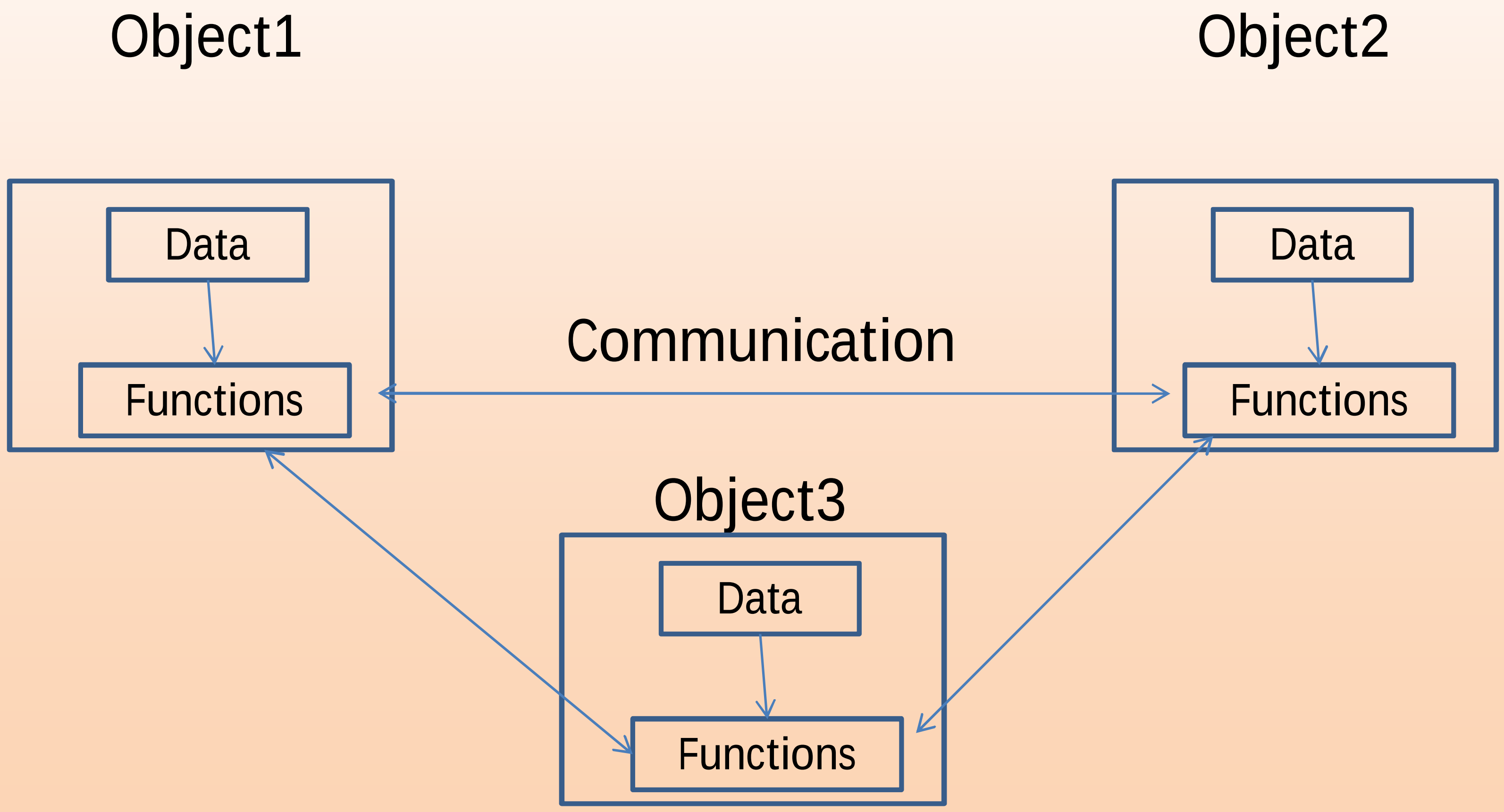
- Large Programs are divided into smaller programs known as functions.
- Most of the functions share global data.

Procedure Oriented Programming

- Data move openly around the system from function to function.
- Functions Transform data from one form to another.
- Follows Top – Down Approach in program design.
- It means it starts execution From the top means opening brace of main() and ends it at corresponding closing brace.

Object Oriented Programming

- OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around theses objects.
- The data of an object can be accessed only by the functions associated with that objects.



Object Oriented Programming

- Emphasis Is on data rather than procedure.
- Programs are divided into objects.
- Data structures are designed such that they characterize the object.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and can not be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follows Bottom – Up approach in program design.

Basic Concepts Of OOP

- Following are the basic concepts of OOP.

1. Objects
2. Classes
3. Data Abstraction
4. Encapsulation
5. Inheritance
6. Polymorphism
7. Dynamic Binding
8. Message Passing

Basic Concepts Of OOP

1. Objects:

   We can define object in following three different methods.

   a. Objects are basic run time entities in an object oriented system. It can be any thing place, person, an account on which we can create programs.

   b. Object is a collection of properties and methods.

   c. Objects are the instance of classes.

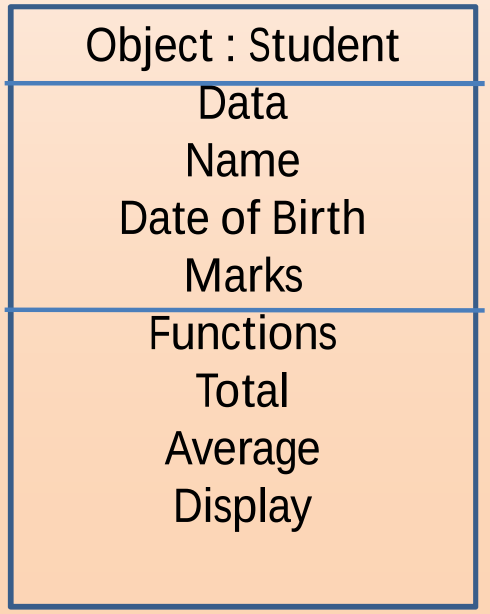
Basic Concepts Of OOP

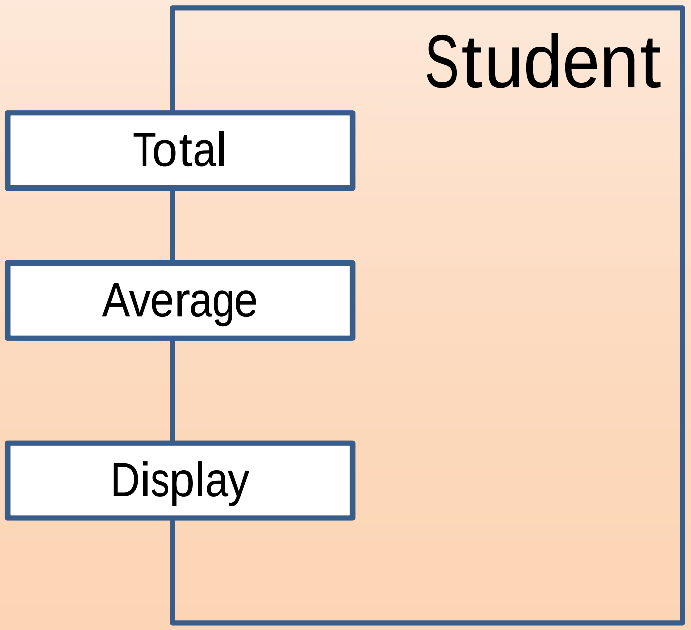
1. Objects:

   -> When a program is executed the objects interact by sending messages to one another.

   -> Each object contains data and code to manipulate the data.

   -> Objects can interact without having to know details of each others data or code.





Basic Concepts Of OOP

2. Classes:

   -> Class is the user defined type which bounds data and functions associated with them into a single unit.

   -> Once we create a class we can create any number of objects.

   -> Eg. If we create a class named student then following statement creates objects s1 and s2 of type student.

   ```
   student s1, s2;
   ```

Basic Concepts Of OOP

3. Data Abstraction:

   -> Abstraction means summary.

   -> Data abstraction means to define data members and functions associated with them without any background detail.

   -> In other words Data abstraction refers to providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.

   -> Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation.

Basic Concepts Of OOP

4. Encapsulation:

   -> Encapsulation means to bound data members and functions that manipulate the data into a single unit.

   -> Encapsulation keeps both (data and functions that manipulate data) safe from outside interference and misuse.

   -> Data encapsulation led to the important OOP concept of **data hiding**.

   -> **Data encapsulation** is a mechanism of bundling the data, and the functions that use them and **data abstraction** is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

Basic Concepts Of OOP

4. Inheritance:

   -> To acquire the properties of one class into another is called inheritance.

   -> It means Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.

   -> This also provides an opportunity to reuse the code functionality and fast implementation time.

   -> When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class.

Basic Concepts Of OOP

4. Inheritance:

   -> This existing class is called the **base** class, and the new class is referred to as the **derived** class.

   -> The idea of inheritance implements the **is a** relationship.

   -> Inheritance is also known as derivation.

   -> For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.

Basic Concepts Of OOP

5. Polymorphism:

   -> Polymorphism is a Greek word and meaning of poly is many.

   -> Polymorphism means ability to take more than one form.

   -> In other words , polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

   -> For example, Consider the operator +, When we pass numbers as operands then it will return the sum of the numbers, but if operands are strings then it will returns third string by concatenate the strings.

   -> function overloading and operator overloading are the examples of polymorphism.

Basic Concepts Of OOP

6. Polymorphism:

   -> Polymorphism is a Greek word and meaning of poly is many.

   -> Polymorphism means ability to take more than one form.

   -> In other words , polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

   -> For example, Consider the operator +, When we pass numbers as operands then it will return the sum of the numbers, but if operands are strings then it will returns third string by concatenate the strings.

   -> Function overloading and operator overloading are the examples of polymorphism.

Basic Concepts Of OOP

7. Dynamic Binding:

   -> The function call is for which function if it decides at runtime then it is called dynamic binding.

   -> In other words in dynamic binding the compiler adds code that identifies the object type at runtime then matches the call with the right function definition.

   -> It includes inheritance and polymorphism both the concepts of OOP.

   -> It is also known as run time polymorphism.

   -> This is achieved by using virtual function.

Basic Concepts Of OOP

8. Message Passing:

   -> Message passing means communication among the objects.

   -> Two or more objects passing values to each other.

Applications Of OOP

Following are some applications of OOP.

1. Client-Server Systems
2. Object-Oriented Databases
3. Real-Time System Design
4. Simulation And Modeling System
5. Hypertext And Hypermedia
6. Neural Networking And Parallel Programming
7. Office Automation Systems
8. CIM/CAD/CAM Systems
9. AI Expert Systems

Advantages Of OOP

The advantages of OOP are mentioned below:

- OOP provides a clear modular structure for programs.
- It is good for defining abstract data types.
- Implementation details are hidden from other modules and other modules has a clearly defined interface.
- It is easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
- objects, methods, instance, message passing, inheritance are some important properties provided by these particular languages
- encapsulation, polymorphism, abstraction are also counts in these fundamentals of programming language.
- It implements real life scenario.
- In OOP, programmer not only defines data types but also deals with operations applied for data structures.

Introduction Of C++

-> **C++** is a general-purpose object-oriented programming language.

-> It was created by Bjarne Stroustrup at Bell Labs circa 1980.

-> C++ is very similar to C (invented by Dennis Ritchie in the early 1970s).

-> C++ is so much compatible with C that it will probably compile over 99% of C programs without changing a line of source code.

-> Though, C++ is a lot well-structured and safer language than C as it OOPs based.

-> C++ is a high-level object-oriented programming language that helps programmers write fast, portable programs.

C++ Language Features

Some of the interesting features of C++ are:

- ***Object-oriented***: C++ is an object-oriented programming language. This means that the focus is on “objects” and manipulations around these objects. Information about how these manipulations work is abstracted out from the consumer of the object.
- ***Rich library support***: Through C++ Standard Template Library (STL) many functions are available that help in quickly writing code. For instance, there are standard libraries for various containers like sets, maps, hash tables, etc.

C++ Language Features

- ***Speed***: C++ is the preferred choice when latency is a critical metric. The compilation, as well as the execution time of a C++ program, is much faster than most other general purpose programming languages.
- ***Compiled***: A C++ code has to be first compiled into low-level code and then executed, unlike interpreted programming languages where no compilation is needed.
- ***Pointer Support***: C++ also supports pointers which are widely used in programming and are often not available in several programming languages.

Uses / Applications Of C++

Following are the applications of C++

- **Operating Systems :**

  Be it Microsoft Windows or Mac OSX or Linux - all of them are programmed in C++

- **Browsers**

  The rendering engines of various web browsers are programmed in C++ simply because if the speed that it offers.

- **Libraries**

  Many high-level libraries use C++ as the core programming language. For instance, several Machine Learning libraries use C++ in the backend because of its speed.

Uses / Applications Of C++

- **Graphics**

  All graphics applications require fast rendering and just like the case of web browsers, here also C++ helps in reducing the latency.

- **Banking Applications**

  One of the most popularly used core-banking system - Infosys Finacle uses C++ as one of the backend programming languages.

- **Cloud/Distributed Systems**

  Large organizations that develop cloud storage systems and other distributed systems also use C++ because it connects very well with the hardware and is compatible with a lot of machines.

Uses / Applications Of C++

- **Databases**

  Postgres and MySQL - two of the most widely used databases are written in C++ and C

- **Embedded Systems**

  Various embedded systems like medical machines, smart watches, etc. use C++ as the primary programming language because of the fact that C++ is closer to the hardware level as compared to other high-level programming languages.

- **Compilers**

  The compilers of various programming languages use C and C++ as the backend programming language.

Basic Structure Of A C++ Program

- The structure of C++ program is divided into four different sections:
  (1) Header File Section
  (2) Class Declaration section
  (3) Member Function definition section
  (4) Main function section

  (1) Header File Section:

  -> This section contains various header files.

  -> You can include various header files in to your program using this section.
- For example:
  # include <iostream.h >
- Header file contains declaration and definition of various built in functions as well as object. In order to use this built in functions or object we need to include particular header file in our program.

Basic Structure Of A C++ Program

(2) Class Declaration Section:

-> This section contains declaration of class.

-> You can declare class and then declare data members and member functions inside that class.

For example:

```
class Demo
{
      int a, b;
      public:
            void input();
            void output();
}
```

-> You can also inherit one class from another existing class in this section.

Basic Structure Of A C++ Program

(3) Member Function Definition Section:

-> This section is optional in the structure of C++ program.

-> Because you can define member functions inside the class or outside the class. If all the member functions are defined inside the class then there is no need of this section.

-> This section is used only when you want to define member function outside the class.

-> This section contains definition of the member functions that are declared inside the class.

- For example:

```
void Demo:: input ()
{
cout << “Enter Value of A:”;
cin >> a;
cout << “Enter Value of B:”;
cin >> b;
}
```

Basic Structure Of A C++ Program

(4) Main Function Section:

-> In this section you can create an object of the class and then using this object you can call various functions defined inside the class as per your requirement.

For example:

```
Void main ()
{
Demo d1;
d1.input ();
d1.output ();
}
```

Note : In above C++ structure the class declaration section and member function definition section both together works as a server and main () function section works as a client.

Input / Output Operators

Input Operator:

-> The statement: cin>>a;

-> The input operator, commonly known as the extraction operator (>>), is used with the standard input stream,

-> Here cin is inbuilt object of istream class, treats data as a stream of characters.

-> These characters flow from cin to the program through the input operator.

-> The input operator works on two operands, namely, the cin stream on its left and a variable on its right.

-> Thus, the input operator takes (extracts) the value through cin and stores it in the variable.

Input / Output Operators

Input Operator:

consider the following example.

```
#include<iostream.h>
int main ()
{
        int a;
        cin>>a;
        a = a+1;
        return 0;
}
```

In this example, the statement cin>> a takes an input from the user and stores it in the variable a.

Input / Output Operators

Output Operator:

-> The statement: cout<<a;

-> The output operator, commonly known as the insertion operator (<<), is used.

-> Here cout is inbuilt object of ostream class.

-> The standard output stream cout Like cin, cout also treats data as a stream of characters.

-> These characters flow from the program to cout through the output operator.

-> The output operator works on two operands, namely, the cout stream on its left and the expression to be displayed on its right.

-> The output operator directs (inserts) the value to cout.

Input / Output Operators

Output Operator:

consider the following example.

```
#include<iostream>
int main ()
{
        int a;
        cin>>a;
        a=a+1;
        cout<<a;
        return 0;
}
```

This example get the value of the variable a from the keyboard and displayed value of variable a through the instruction cout << a .

Introduction Of namespace

namespace:

-> Namespaces allow us to group named entities that otherwise would have *global scope* into narrower scopes, giving them *namespace scope*.

-> This allows organizing the elements of programs into different logical scopes referred to by names.Namespace is a feature added in C++ and not present in C.

-> A namespace is a declarative region that provides a scope to the identifiers (names of the types, function, variables etc) inside it.

-> Multiple namespace blocks with the same name are allowed.

-> All declarations within those blocks are declared in the named scope.

TOKENS OF C++

Tokens:

-> Tokens are the smallest individual unit of the program.

-> c++ has the following tokens.

- => keywords
- => identifiers
- => constants
- => string
- => operators

keywords:

-> keywords are the predefined words.

-> We can not change the meaning of them.

-> They are the reserved identifiers and can not be used as names for the program variables.

-> eg. int, float, case, break, for, if, class, void etc.

TOKENS OF C++

Identifiers:

-> A C++ identifier is a name used to identify a variable, function, class, module, or any other user-defined item.

-> An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).

-> 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++.

-> We can not declare keyword as identifier.

-> Some valid identifiers are

Void, main, ab_12, cpp123, etc….

TOKENS OF C++

Constants:

-> Constants refer to fixed values that the program may not alter and they are called **literals**.

-> Constants can be of any of the basic data types and can be divided into Integer Numerals, Floating-Point Numerals, Characters, Strings and Boolean Values.

-> Constants are treated just like regular variables except that their values cannot be modified after their definition.

Integer Constants:

- An integer constants can be a decimal, octal, or hexadecimal constant.
- A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.
- An integer constant can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

TOKENS OF C++ (Constants)

Integer Constants:

Here are some examples of integer Constant

```
212          // Legal          215u // Legal
0xFeeL       // Legal          078 // Illegal: 8 is not an octal digit
032UU        // Illegal: cannot repeat a suffix
```

Following are other examples of various types of Integer Constant

```
85           // decimal             0213 // octal
0x4b         // hexadecimal         30 // int
30u          // unsigned int        30l // long
30ul         // unsigned long
```

TOKENS OF C++ (Constants)

Floating-point Constants:

- A floating-point constant has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point constants either in decimal form or exponential form.
- While representing using decimal form, you must include the decimal point, the exponent, or both and while representing using exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.
- Here are some examples of floating-point constants

```
3.14159     // Legal        314159E-5L    // Legal
510E        // Illegal: incomplete exponent
210f        // Illegal: no decimal or exponent .
e55         // Illegal: missing integer or fraction
```

TOKENS OF C++ (Constants)

Boolean Constants:

-> There are two Boolean constants and they are part of standard C++ keywords

– A value of **true** representing true.
– A value of **false** representing false.

-> You should consider the value of true equal to nonzero and value of false equal to zero.

Character Constants:

-> Character constants are enclosed in single quotes.

-> If the constant begins with L (uppercase only), it is a wide character literal (e.g., L'x') and should be stored in **wchar_t** type of variable .

-> Otherwise, it is a narrow character literal (e.g., 'x') and can be stored in a simple variable of **char** type.

-> A character literal can be a plain character (e.g., 'x'), an escape sequence (e.g., '\t'), or a universal character (e.g., '\u02C0').

TOKENS OF C++ (Constants)

Character Constants:

-> There are certain characters in C++ when they are preceded by a backslash they will have special meaning and they are used to represent like newline (\n) or tab (\t).

-> Here, you have a list of some of such escape sequence codes.

Escape sequence	Meaning
\\	Back slash character
\"	Double Quotes character
\’	Single Quote character
\?	Question Mark Character
\a	Alert or bell
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\ooo	Octal number of one to three digits
\xhh . . .	Hexadecimal number of one or more digits

TOKENS OF C++ (Constants)

String Constants:

-> String literals are enclosed in double quotes.

-> A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters.

-> You can break a long line into multiple lines using string literals and separate them using whitespaces.

-> Here are some examples of string literals.

-> All the three forms are identical strings.

```
"hello, dear"
"hello, \ dear"
"hello, “  "d" "ear"
```

TOKENS OF C++ (Constants)

Data Types Of C++:

-> C++ data types can be divided into following categories.

1. Basic Data Types
2. User Defined Data Types
3. Derived Data Types.

-> Basic Data Types:

-> Basic Data types of C++ can be divided into main three types

=> integer : This type is useful to store whole numbers.

=> real : This type is useful to store real numbers.

=> character : This type is useful to store alphabets, digits and symbols.

TOKENS OF C++ (Constants)

Data Types Of C++:

-> All these types are further divided into sub types. Following table describes actual data type, memory occupied by the type and range of the type.

Type	**Size (In Bytes)**	**Range**
unsigned short int	2 bytes	0 to 65,535
short int	2 bytes	-32,768 to 32,767
unsigned long int	4 bytes	0 to 4,294,967,295
long int	4 bytes	-2,147,483,648 to 2,147,483,647
int (16 bit)	2 bytes	-32,768 to 32,767
int (32 bit)	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned int (16 bit)	2 bytes	0 to 65,535
unsigned int (32 bit)	2 bytes	0 to 4,294,967,295
signed char	1 byte	256 character values
unsigned char	1 byte	-128 To 127 character values
float	4 bytes	1.2e-38 to 3.4e38
double	8 bytes	2.2e-308 to 1.8e308
long double	10 Bytes	1.2e-4932 to 3.7e4932

TOKENS OF C++ (Constants)

Data Types Of C++:

2. User Defined Data Types

-> class, struct and union are user defined data types.

3. Derived Data Types.

-> array and pointer are known as derived data types.

TOKENS OF C++

Operators:

-> In an expression c = a + b;

-> a, b, c are operands. (The variables which we use in expression are known as operands)

-> +, and = are operators. (The symbols which we use in expression are known as operators)

-> Addition and Assignment are operations. (Whole process is known as operation)

-> C++ has following types of operators.

=> Arithmetic Operators

=> Relational Operators

=> Logical Operators

=> Assignment Operators

=> Increment / Decrement Operators

=> Conditional Operators

=> Bitwise Operators

TOKENS OF C++ (Operators)

Arithmetic Operators:

Operator	Meaning	Example [a=13, b=22]
+	Addition	a+b => 13+22 => 35
-	Subtraction	a-b => 13-22 => -9
*	Multiplication	a*b => 13*22 => 286
/	Division	a/b => 13 / 22 => 0

Note : If both the operands are integers then / operator return integer division.

%	Modules	a%b => 13%22 => 13

Note : % operator returns remainder.

-> Precedence Of Arithmetic Operators:

(%, /, *) (+, -)

-> Eg. 5 * 3 + 13 / 5 – 10 % 3 => 15 + 2 – 1 => 16
15 * 3 / 7 % 4 => 45 / 7 % 4 => 6 % 4 => 2

TOKENS OF C++ (Operators)

Relational Operators:

Operator	Meaning	Example [a=13, b=22]
>	Greater Than	a>b=> 13>22 => False
>=	Greater Than Or Equal To	a>=b => 13>=22 => False
<	Less Than	a<b=> 13<22 => True
<=	Less Than Or Equal To	a<=b => 13<=22 => True
==	Equal To	a==b => 13==22 => False
!=	Not Equal To	a!=b => 13 != 22 => False

-> Precedence Of Relational Operators:

Left To Right

TOKENS OF C++ (Operators)

Logical Operators:

Operator	Meaning	Example [a=13, b=22]
&&	Logical And	a >=13 && b<22 => True && False => False

[It returns true if both the conditions are true]

Operator	Meaning	Example
||	Logical Or	a >=13 || b<22 => True || False => True

[It returns true if either or condition is true]

Operator	Meaning	Example
!	Logical Not	!(a >=13) => !(True) => False

[It returns true if condition is false]

-> Precedence Of Logical Operators:

! , [&&, ||]

TOKENS OF C++ (Operators)

Assignment Operators:

-> In C++ There is only one assignment operator (i.e. =)

-> This operator assign value of Right hand Side To Left Hand Side.

-> Eg. a = b;

Above statement assign value of b to a.

-> C++ have some shorthand assignment operators.

Operator	Meaning	Example [a=22, b=13]
+=	a+=b => a=a+b	a+=b => a=35 and b=13
-=	a-=b => a=a-b	a-=b => a=9 and b=13
*=	a*=b => a=a*b	a*=b => a=286 and b=13
/=	a/=b => a=a/b	a/=b => a=1 and b=13
%=	a%=b => a=a%b	a%=b => a=9 and b=13

TOKENS OF C++ (Operators)

Increment / Decrement Operators:

-> The Increment Operator ++ is used to add 1 to operand and Decrement Operator – is used to subtract 1 from the operand.

-> Eg.
```
int a = 5, b = 5;
a++;  // a+=1; // a = a + 1;
b--;  // b-=1;  // b = b + 1;
```

-> In above example First statement assign 5 to a and b.

-> Second statement add 1 to operand a.

-> Third statement subtract 1 from b.

TOKENS OF C++ (Operators)

Prefix And Postfix:

-> Both increment and decrement operators come under prefix and postfix.

-> There is no difference between prefix and postfix when we write them individually.

-> But if we use them with assignment operator they behave differently.

prefix	postfix
int a = 5, b;	int a = 5 , b;
b = ++a; // [a=a+1][b=a]	b = a++; // [b=a] [a=a+1]
o/p:	
a = 6 and b = 6	a = 6 and b = 5

-> In prefix first it add 1 to operand and then assign the value.

-> Where as in postfix first it assign the value and then add 1 to operand.

TOKENS OF C++ (Operators)

Conditional Operator:

-> Pair of ?: is known as conditional operator.

-> It is also known as ternary operator.

-> Syntax:

```
    expr1 ? Expr2 : expr3;
```

-> Here expr1 is logical expression.

-> If it returns true then it will execute expr2 otherwise execute expr3.

-> eg.

```
    int a = 5;
    a%2==0 ? cout<<“Even” : cout<<“Odd”;

    int a = 0;
    a==0?cout<<“Zero”:a%2==0?cout<<“Even”:cout<<“Odd”;
```

TOKENS OF C++ (Operators)

Bitwise Operators:

-> Bitwise operator works on bits

-> It perform bit-by-bit operation.

-> Generally it works only with integers.

-> How bitwise operators works:

=> First it convert given decimal value into binary

=> Then find the result

=> Again convert binary result into decimal and return it.

Operator	Meaning	Example [a=22, b=13]
&	Bitwise And	a&b => 22&13 => 11010 & 01101 => 01000 (8)

Note : Binary value of 22 is 11010 and 13 is 1101

TOKENS OF C++ (Operators)

Bitwise Operators:

Operator	Meaning	Example
|	Bitwise Or	a|b => 22|13
		=> 11010 | 01101
		=> 11111 (31)
^	Bitwise XOR	a^b => 22^13
		=> 11010 ^ 01101
		=> 10111 (23)
>>	Shift Right	a>>2 => 22 >> 2
		=> 11010 >> 2
		=> 110 (6)
<<	Shift Left	a<<2 => 22 << 2
		=> 11010 << 2
		=> 1101000 (106)
~	1's Complement	~a => ~22 => ~11010
	[It Convert all 0 into 1 and all 1 into 0]	=> 1111111111100101 (65509)

TOKENS OF C++ (Operators)

More On Operator:

-> There are some special operators in c++.

=> Scope Resolution Operator

=> Member Dereferencing Operator

=> Memory Management Operator

=> Manipulators

=> Type Cast Operator

Scope resolution Operator:

-> Like c lang., c++ is also a block structured language.

-> We know that the same var. name can be used to have different meanings in different blocks.

-> The scope of the variable extends from the point of its declaration till the end of the block containing the declaration.

-> In C, the global version of a variable cannot be accessed from within the inner block.

TOKENS OF C++ (Operators)

Scope resolution Operator:

-> C++ resolves this problem by introducing a new operator **::** called the scope resolution operator.

-> This can be used to uncover a hidden variable.

Syntax:

:: variable name

-> This operator allows access to the global version of a variable.

eg.

```
int a=100;;
void main()
{
        int a=10;
        cout<<“\n a = “<<a;
        cout<<“\n::a=“<<::a;
}
```

TOKENS OF C++ (Operators)

Member Dereferencing Operators:

-> C++ permits us to define a class containing various types of data and functions as members.

-> C++ also permits us to access the class members through pointers.

-> In order to achieve this, c++ provides a set of three pointer to member operators.(also known as member dereferencing operators)

Operator	Meaning
::*	To declare a pointer to a member of a class.
.*	To access a member using object name and a pointer to that member.
->*	To access a member using pointer to the object and a pointer to that member.

TOKENS OF C++ (Operators)

Memory management Operators:

-> In c language we uses malloc() and calloc() functions to allocate memory dynamically at run time.

-> Similarly, it uses the function free() to free dynamicaly allocated memory.

-> We can also use these functions in c++.

-> C++ also defines two unary operators **new** and **delete** that perform the task of allocating and freeing the memory in a better and easier way.

new Operator:

-> An object can be created by using new.

-> A data object created inside a block with new, will remain in existence until it is explicitly destroyed by using delete.

-> The new operator can be used to create objects of any type.

TOKENS OF C++ (Operators)

Memory management Operators:

Syntax:

```
        poitnervar = new datatype;
```

-> Here pointervar is a pointer variable of type datatype.

-> The new operator allocates sufficient memory to hold a data object of type datatype and returns the address of the object.

-> The datatype is any valid data type.

-> The pointer variable holds the address of the memory space allocated.

For Example:

```
        int *p = new int;
        float *q = new float;
```

TOKENS OF C++ (Operators)

Memory management Operators:

-> In above example p is a pointer of type int and q is a pointer of type float.

-> The statement

```
    *p = 25;
    *q = 7.5;
```

-> assigns 25 to the newly created int object and 7.5 to the float object.

-> We can also initialize the memory using the new operator.

Syntax:

```
    poitnervar = new datatype ( value );
```

-> Here value specifies the initial value.

For example:

```
            int *p = new int(20);
```

TOKENS OF C++ (Operators)

Memory management Operators:

-> new can be used to create a memory space for any data type including user defined type such as arrays, structures and classes.

Syntax:

```
        poitnervar = new datatype [ size ];
```

-> Here size specifies the number of .

-> The statement

```
                int *p = new int [5];
```

-> creates a memory space for an array of 5 integers.

-> p[0] will refer to the first element, p[1] to the second element and so on.

TOKENS OF C++ (Operators)

Memory management Operators:

-> When creating multi dimensional arrays with new , all the array sizes must be specified.

For Example:

```
array_ptr = new int[3][4][5];        //legal
array_ptr = new int[m][4][5];        //legal
array_ptr = new int[3][4][];         //illegal
array_ptr = new int[][4][5];         //illegal
```

Note : The first dimension may be a variable whose value is supplied at runtime. All others must be constant.

TOKENS OF C++ (Operators)

Memory management Operators:

delete:

-> When data is no longer needed, it is destroyed to release the memory space for reuse.

Syntax:

```
	delete pointervar;
```

-> Here pointervar is the pointer that points to a data object created with new.

For Example:

```
	delete p;
	delete q;
```

-> If we want to free a dynamically allocated array, we must use the following form of delete.

```
	delete [size] pointervar;
```

-> Here the size specifies the number of elements in the array to be freed.

-> The statement `delete [] p;`

-> will delete the entire array pointed to by p.

TOKENS OF C++ (Operators)

Manipulators:

-> Manipulators are operators that are used to format the data display.

-> The most commonly used manipulators are endl and setw.

-> The endl manipulator, when used in an output statement, causes a linefeed to be inserted.[Just like ‘\n’

Type cast operators:

-> C++ permits explicit type conversion of variables or expressions using the type cast operator.

-> Traditional C casts are augmented in C++ by a function-call notation as a syntactic alternative

Syntax:

```
(typename) expression // C Notation
typename (expression) // C++ Notation
```

For Example:

```
average = sum / (float) i ;  // C Notation
average = sum / float ( i );  // C++ Notation
```

Reference Variable

Reference Variables:

-> A reference variable is an alias, that is, another name for an already existing variable.

-> Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.

Syntax:

```
        datatype & referecename = variablename;
```

For Example:

```
        float total = 100;
        float & sum = total;
```

-> total is a float type variable that has already been declared.

-> sum is the alternative name declared to represent the variable total.

-> Both the variables refer to the same data object in the memory.

-> Now the statement

```
        cout<<total;      and      cout<<sum;
```

Reference Variable

Reference Variables:

-> both print the value 100.

-> The statement total = total + 10;

-> will change the value of both total and sum to 110.

-> Same way, the assignment sum = 0;

-> will change the value of both the variables to zero.

-> A reference variable must be initialized at the time of declaration.

-> This establishes the correspondence between the reference and the data object which it names.

-> It is completely different from assignment to it.

-> C++ assigns additional meaning to the symbol &.

-> Here & is not an address operator.

-> The notation float & means reference into float.

Control structures

Conditional Control Structures:

-> C++ have following control structures.

- => simple if
- => if ... else
- => nested if else...
- => else if ladder
- => switch statement

-> Simple if

Use : When we want to execute the statements only if cond. Is true.

Syntax:

```
if(cond)
{
        statements;
}
stat-x;
```

Control structures

Conditional Control Structures:

-> Simple if

Description :

-> First it check the condition.

-> If condition is true then it will execute the statements.

-> But if condition is false then it skip the statements and directly execute the next statement(stat-x).

For Example:

```
int age;
cout<<"Enter age";
cin>>age;
if(age>=18)
{
        cout<<"You are eligible  for voting";
}
```

Control structures

Conditional Control Structures:

-> if else

Use : When we want to execute the different statements for condition is true and false

Syntax:

```
        if(cond)
        {
                true block statements;
        }
        else
        {
                false block statements;
        }
        stat-x;
```

Control structures

Conditional Control Structures:

-> if else

Description :

-> First it check the condition.

-> If condition is true then it will execute the true block statements.

-> But if condition is false then it will execute the false block statements.

-> It will execute either true block stat or false block stat and then jump to the next stat(stat-x).

For Example:

```
if(n%2==0)
{
        cout<<"No. is Even";
}
else
{
        cout<<"No. is Odd";
}
```

Control structures

Conditional Control Structures:

-> nested if else ….

Use : Nested if means one if within another if.

-> When one condition is depend upon another at that time we used nested if else statement.

Syntax:

```
if(cond1)
{
        if(cond2)
        {       stat1;
        }
        else
        {       stat2;
        }
}
else
{       stat3;
}
stat-x;
```

Control structures

Conditional Control Structures:

-> nested if else ….

Description :

-> First it check the cond1.

-> If cond1 is true then it will check cond2

-> If cond2 is true then it will execute the stat1 block.

-> If con2 is false then it will execute stat2 block.

-> But if cond1 is false then it will execute stat3 block.

-> It means it executes either stat1 or stat2 or stat3 block and after executing any block of stat it will jump to the next stat.

For Example:

```
      if(n!=0)  {
                    if(n%2==0)          {
                              cout<,"Even";      }
                    else          {
                              cout<<"Odd";      }            }
      else          {
               cout<<'Zero";       }
```

Control structures

Conditional Control Structures:

-> else if ladder

Use : When we want to check multiple conditions one by one.

Syntax:

```
if(cond1)
{           stat1;
}
else if(cond2)
{           stat2;
}
……………………………………………….
……………………………………………….
……………………………………………….
else
{           statelse;
}
stat-x;
```

Control structures

Conditional Control Structures:

-> else if ladder

Description :

-> First it check the cond1.

-> If cond1 is true then it will execute stat1.

-> But If cond1 is false then it will check cond2.

-> If con2 is true then it will execute stat2 block.

-> same way it check all the conditions one by one and if all the conditions are false then it will execute statelse.

-> It means it executes either stat1 or stat2 or ….. statelse block and after executing any block of statement it will jump to the next stat.

For Example:

```
        if(n==0)  {
                cout<<“Zero”;
        } else if(n%2==0)  {
                cout<,”Even”;
        } else      {
                cout<<“Odd”;
        }
```

Control structures

Conditional Control Structures:

-> switch statement:

Use : When we want to check multiple conditions one by one at that time instead of else if ladder we may use switch statement.

Syntax:

```
switch(expr)
{
        case val1:
                stat1;
                break;
        case val2:
                stat2;
                break;
        ......................................
        ......................................
        default:
                stat-default;
                break
}
stat-x;
```

Control structures

Conditional Control Structures:

-> switch statement:

Description :

-> First it compare value of expr with val1.

-> If it is then it will execute stat1.

-> If not then it compare value of expr with val2.

-> If it is then it will execute stat2.

-> same way it compare value of expr with all the case values and If it fails in all the cases then it will execute stat-default.

-> Limitations of switch statement.

1. Type of expr should be int or char.
2. It can not accept relational or logical operators.
3. It accept only single expr.
4. break is required at end of every case.

Control structures

Conditional Control Structures:

For Example:

```
char c;
cout<,"Enter color code";
cin>>c;
switch(c)
{
        case 'R':
                cout<<"Red";
                break;
        case 'B':
                cout<<"Blue";
                break;
        case 'G':
                cout<<"Green";
                break;
        default:
                cout<,'Invalid Color code";
}
```

Control structures

Looping Control Structures:

Loop : A **loop** is a **programming** structure that repeats a sequence of instructions until a specific condition is met.

-> C++ have following looping statements.

=> for loop

=> while loop

=> do … while loop

-> for loop:

Use : When no. of iterations are known.

-> for is an entry controlled looping statement

Syntax :

```
        for(init ; cond ; incr)
        {
                body of the loop;
        }
```

Control structures

Looping Control Structures:

-> for loop:

Description:

-> First it initialize the loop var.

-> Then it check the cond.

-> If cond is true then it execute the body of the loop.

-> Then it incr / decr the value of the loop var.

-> Again it check the cond.

-> Continue this process until cond is false.

For Example:

1.
```
for(i=1 ; i<=10 ; i++)
{
        cout<<i;
}
```

Will print 1, 2, 3, ………… 10

Control structures

Looping Control Structures:

-> for loop:

For Example:

```
2.            for(i=10 ; i>=1 ; i--)
              {
                      cout<<i;
              }
Will print    10, 9, 8, .......... 1
3.            for(i=1 ; i<=10 ; ++i)
              {
                      cout<<i;
              }
Will print    1, 2, 3 ............ 10
```

Control structures

Looping Control Structures:

-> Nested for loop:

-> Nested for means one for loop within another for loop.

Syntax:

```
            for(init1 ; cond1 ; incr1)
            {
                    for(init2 ; cond2 ; incr2)
Outer Loop          {                                    Inner  Loop

                            body of inner loop
                    }
            }
            stat-x;
```

Control structures

Looping Control Structures:

-> Nested for loop:

Description:

-> First it initialize the outer loop var (init1).

-> Then it check the outer loop cond (cond1) .

-> If outer loop cond (cond1) is true then it enters in the outer loop.

-> Now it initialize the inner loop var (init2)

-> Then it check the inner loop cond (cond2).

-> If inner loop cond (cond2) is true then it execute body of inner loop.

-> Now it incr / decr the value of inner loop var.

-> Again check the inner loop cond (cond2).

-> Continue the execution of inner loop until inner loop cond is false.

-> Now it incr / decr the value of outer loop var (incr1)

-> Then it check the outer loop cond (cond1).

-> Continue the execution of outer loop until outer loop cond (cond2) is false.

Control structures

Looping Control Structures:

-> Nested for loop:

For Example:

```
for(i=1; i<=5 ; i++)
{
        for(j=1 ; j<=i; j++)
        {
                cout<<"\t"<<j;
        }
        cout<<endl;
}
```

Output:

```
        1
        1       2
        1       2       3
        1       2       3       4
        1       2       3       4       5
```

Control structures

Looping Control Structures:

-> while loop:

Use : When no. of iterations are unknown but first value of loop variable is known.

-> while is an entry controlled looping statement

Syntax :

```
while( cond )
{
        body of the loop;
}
stat-x;
```

Description:

-> First it check the condition.

-> If condition is true then it execute the body of the loop.

-> Again it check the condition.

-> Continue this process until condition is false.

Control structures

Looping Control Structures:

-> while loop:

For Example:

```
int n, s=0, r;
cout<<“Enter n:”;
cin>>n;
while(n>0)
{
        r = n % 10;
        s = s + r;
        n = n / 10;
}
cout<<”\n Sum =“<<s;
```

-> Above example display the sum of individual digit of the number.

Control structures

Looping Control Structures:

-> do ….. while loop:

Use : When no. of iterations and first value of loop variable both are unknown.

-> do … while is an exit controlled looping statement

Syntax :

```
do
{
        body of the loop;
} while( cond );
stat-x;
```

Description:

-> First without checking any type of condition it execute the body of the loop.

-> Now it check the condition.

-> If condition is true then it re-execute the body of the loop.

-> Again it check the condition.

-> Continue this process until condition is false.

Control structures

Looping Control Structures:

-> do …. while loop:

For Example:

```
int n, s=0;
do
{
        cout<<“Enter n (0 to terminate):”;
        cin>>n;
        if(n!=0)
                s = s + n;
}while(n!=0);
cout<<”\n Sum =“<<s;
```

-> Above example accept the numbers one by one until the number is 0 and then display sum of all the numbers.

Control structures

Jumping statements:

-> C++ have following jumping statements.

=> goto

=> break

=> continue

-> goto statement

Use : This statement is useful to skip as well as to repeat some statements.

-> There are two types of jump is with goto statement.

=> Forward Jump

=> Backward Jump

-> Forward Jump:

Use : This type of jump is useful to skip some statements.

Syntax:

```
          goto Label;
                statements;
          Label:
```

Control structures

Jumping statements:

Description:

-> In this type of jump **goto Label** is before the **Label**.

-> It skip the statements which we write between **goto Label** and **Label**.

For Example:

```
int n;
cout<<"Enter  n";
cin>>n;
if(n<0)
        goto L1;
cout<<"Square root of the number is "<<sqrt(n);
L1:
```

-> Above example display the square root of the inputted number if it is non-negative.

Control structures

Jumping statements:

-> Backward Jump

Use : This type of jump is useful to repeat some statements.

Syntax:

```
Label:
        statements;
goto Label:
```

Description:

-> In this type of jump **goto Label** is after the **Label**.

-> It repeat the statements which we write between **Label** and **goto Label.**

For Example:

```
int n=1;
L1:
cout<<"\t"<<n++;
if(n<=10)
        goto L1;
```

-> Above example display 1, 2, 3, ... 10.

Control structures

Jumping statements:

break:

Use: break is useful for early exit from the loop.

Syntax:

```
break;
```

Description:

-> break statement skip the statements which we write after it and exit form the loop.

For Example:

```
for(i=1;i<=5;i++)
{
        if(i%3==0)
                break;
        cout<<i;
}
```

-> Above example display 1 and 2. It break the loop when value of i is 3.

Control structures

Jumping statements:

continue:

Use: continue is useful for continue the loop with next value of the loop var.

Syntax:

```
continue;
```

Description:

-> continue statement skip the statements which we write after it and continue the loop with next value of the loop variable.

For Example:

```
for(i=1;i<=5;i++)
{
        if(i%3==0)
                continue;
        cout<<i;
}
```

-> Above example display 1, 2, 4 and 5. It skip the print statement when value of i is 3 and continue the loop with next value of i i.e. 4..