UNIT-1

Introduction to Object-Oriented Programming

History & overview of C++:

- C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.
- C++ is regarded as a middle-level language, as it comprises a combination of both highlevel and low-level language features.
- C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.
- C++ is a superset of C, and that virtually any legal C program is a legal C++ program.
- Note: A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time.

Basic concepts of OOP:

The various OOPS concepts in C++ are:

- **Class:** The class is a user-defined data type which defines its properties and its functions. For example, Human being is a class. The body parts of a human being are its properties, and the actions performed by the body parts are known as functions. The class does not occupy any memory space. Therefore, we can say that the class is the only logical representation of the data.

The syntax of declaring the class:

```
class student
{
//data members;
//Member functions
}
```

- **Object:** An object is a run-time entity. An object is the instance of the class. An object can represent a person, place or any other item. An object can operate on both data members and member functions. The class does not occupy any memory space. When an object is created using a new keyword, then space is allocated for the variable in a heap, and the starting address is stored in the stack memory. When an object is created without a new keyword, then space is not allocated in the heap memory, and the object contains the null value in the stack.

```
class Student
{
//data members;
//Member functions
}
```

The syntax for declaring the object:

```
Student s = new Student();
```

- Inheritance: Inheritance provides reusability. Reusability means that one can use the functionalities of the existing class. It eliminates the redundancy of code. Inheritance is a technique of deriving a new class from the old class. The old class is known as the base class, and the new class is known as derived class.

Syntax

```
class derived_class :: visibility-mode base_class;
```

- **Encapsulation:** Encapsulation is a technique of wrapping the data members and member functions in a single unit. It binds the data within a class, and no outside method can access the data. If the data member is private, then the member function can only access the data.
- **Abstraction:** Abstraction is a technique of showing only essential details without representing the implementation details. If the members are defined with a public keyword, then the members are accessible outside also. If the members are defined with a private keyword, then the members are not accessible by the outside methods.
- **Data binding:** Data binding is a process of binding the application UI and business logic. Any change made in the business logic will reflect directly to the application UI.
- **Polymorphism:** Polymorphism means multiple forms. Polymorphism means having more than one function with the same name but with different functionalities. Polymorphism is of two types:

1. Static polymorphism is also known as early binding.
2. Dynamic polymorphism is also known as late binding.

Benefits of OOP:

- We can build the programs from standard working modules that communicate with one another, rather than having to start writing the code from scratch which leads to saving of development time and higher productivity,
- OOP language allows to break the program into the bit-sized problems that can be solved easily (one object at a time).
- The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost.
- OOP systems can be easily upgraded from small to large systems.
- It is possible that multiple instances of objects co-exist without any interference,
- It is very easy to partition the work in a project based on objects.
- It is possible to map the objects in problem domain to those in the program.
- The principle of data hiding helps the programmer to build secure programs which cannot be invaded by the code in other parts of the program.
- By using inheritance, we can eliminate redundant code and extend the use of existing classes.
- Message passing techniques is used for communication between objects which makes the interface descriptions with external systems much simpler.
- The data-centred design approach enables us to capture more details of model in an implementable form.

Difference between POP and OOP:

Sr. No.	Key	OOP	POP
1	Definition	OOP stands for Object Oriented Programming.	POP stands for Procedural Oriented Programming.
2	Approach	OOP follows bottom up approach.	POP follows top down approach.
3	Division	A program is divided to objects and their interactions.	A program is divided into funtions and they interacts.
4	Inheritance supported	Inheritance is supported.	Inheritance is not supported.
5	Access control	Access control is supported via access modifiers.	No access modifiers are supported.
6	Data Hiding	Encapsulation is used to hide data.	No data hiding present. Data is globally accessible.
7	Example	C++, Java	C, Pascal

Application of OOP:

- Real-Time System design: Real-time system inherits complexities and makes it difficult to build them. OOP techniques make it easier to handle those complexities.
- Hypertext and Hypermedia: Hypertext is similar to regular text as it can be stored, searched, and edited easily. Hypermedia on the other hand is a superset of hypertext. OOP also helps in laying the framework for hypertext and hypermedia.
- AI Expert System: These are computer application that is developed to solve complex problems which are far beyond the human brain. OOP helps to develop such an AI expert System
- Office automation System: These include formal as well as informal electronic systems that primarily concerned with information sharing and communication to and from people inside and outside the organization. OOP also help in making office automation principle.
- Neural networking and parallel programming: It addresses the problem of prediction and approximation of complex-time varying systems. OOP simplifies the entire process by simplifying the approximation and prediction ability of the network.
- Stimulation and modelling system: It is difficult to model complex systems due to varying specifications of variables. Stimulating complex systems require modelling

and understanding interaction explicitly. OOP provides an appropriate approach for simplifying these complex models.

- Object-oriented database: The databases try to maintain a direct correspondence between the real world and database object in order to let the object retain it identity and integrity.
- Client-server system: Object-oriented client-server system provides the IT infrastructure creating object-oriented server internet(OCSI) applications.
- CIM/CAD/CAM systems: OOP can also be used in manufacturing and designing applications as it allows people to reduce the efforts involved. For instance, it can be used while designing blueprints and flowcharts. So it makes it possible to produce these flowcharts and blueprint accurately.

Control structures: Decision making statements and loops

- if Statement :-
  The syntax of the if statement is:
  if (condition) {
  //body of if statement
  }

```
// Program to print positive number entered by the user
// If the user enters a negative number, it is skipped

#include <iostream>
using namespace std;
int main() {
  int number;
  cout << "Enter an integer: ";
  cin >> number;
  // checks if the number is positive
  if (number > 0) {
    cout << "You entered a positive integer: " << number << endl;
  }
  cout << "This statement is always executed.";
  return 0;
}
```

- if...else statement :-
  The if statement can have an optional else clause. Its syntax is:

```
if (condition) {
  // block of code if condition is true
}
else {
  // block of code if condition is false
}
```

```
// Program to check whether an integer is positive or negative
// This program considers 0 as a positive number

#include <iostream>
using namespace std;

int main() {

  int number;

  cout << "Enter an integer: ";
  cin >> number;

  if (number >= 0) {
    cout << "You entered a positive integer: " << number << endl;
  }
  else {
    cout << "You entered a negative integer: " << number << endl;
  }

  cout << "This line is always printed.";

  return 0;
}
```

- if...else...else if statement :-
  The if...else statement is used to execute a block of code among two alternatives. However, if we need to make a choice between more than two alternatives, we use the if...else if...else statement.

  The syntax of the if...else if...else statement is:

  ```
  if (condition1) {
    // code block 1
  }
  else if (condition2){
    // code block 2
  }
  else {
    // code block 3
  }
  ```

  Note: There can be more than one else if statement but only one if and else statements.

```
// Program to check whether an integer is positive, negative or zero

#include <iostream>
using namespace std;

int main() {

  int number;

  cout << "Enter an integer: ";
  cin >> number;

  if (number > 0) {
    cout << "You entered a positive integer: " << number << endl;
  }
  else if (number < 0) {
    cout << "You entered a negative integer: " << number << endl;
  }
  else {
    cout << "You entered 0." << endl;
  }

  cout << "This line is always printed.";

  return 0;
}
```

- **Nested if...else**
  Sometimes, we need to use an if statement inside another if statement. This is known as nested if statement. Think of it as multiple layers of if statements. There is a first, outer if statement, and inside it is another, inner if statement.
  Its syntax is:

```
// outer if statement
if (condition1) {
  // statements
  // inner if statement
  if (condition2) {
    // statements
  }
}
```

```
// C++ program to find if an integer is positive, negative or zero using nested if statements

#include <iostream>
using namespace std;
int main() {
int num;
   cout << "Enter an integer: ";
  cin >> num;

 // outer if condition
 if (num != 0) {

   // inner if condition
   if (num > 0) {
     cout << "The number is positive." << endl;
   }
   // inner else condition
   else {
     cout << "The number is negative." << endl;
   }
 }
 // outer else condition
 else {
   cout << "The number is 0 and it is neither positive nor negative." << endl;
 }
 cout << "This line is always printed." << endl;
 return 0;
}
```

Notes:

- We can add else and else if statements to the inner if statement as required.
- The inner if statement can also be inserted inside the outer else or else if statements (if they exist).
- We can nest multiple layers of if statements.

- **C++ for loop:-**
  The syntax of for-loop is:

```
for (initialization; condition; update)
{
   // body of-loop
}
```

Example 1: Printing Numbers From 1 to 5

```
#include <iostream>

using namespace std;

int main() {
      for (int i = 1; i <= 5; ++i) {
      cout << i << " ";
   }
   return 0;
}
```

- **C++ while Loop:-**
  The syntax of the while loop is:

```
while (condition) {
   // body of the loop
}
```

```
// C++ Program to print numbers from 1 to 5

#include <iostream>
using namespace std;

int main() {
   int i = 1;

   // while loop from 1 to 5
   while (i <= 5) {
      cout << i << " ";
      ++i;
   }
   return 0;
}
```

- **C++ do...while Loop**
  The do...while loop is a variant of the while loop with one important difference: the body of do...while loop is executed once before the condition is checked.

  Its syntax is:

```
do {
  // body of loop;
}
while (condition);
```

```
// C++ Program to print numbers from 1 to 5

#include <iostream>
using namespace std;

int main() {
   int i = 1;

   // do...while loop from 1 to 5
   do {
      cout << i << " ";
      ++i;
   }
   while (i <= 5);

   return 0;
}
```

- **C++ Array Declaration**

  dataType arrayName[arraySize];

  For example:- int x[6];

How to insert and print array elements?

```
int mark[5] = {19, 10, 8, 17, 9}

// change 4th element to 9
mark[3] = 9;

// take input from the user
// store the value at third position
cin >> mark[2];


// take input from the user
// insert at ith position
cin >> mark[i-1];

// print first element of the array
cout << mark[0];

// print ith element of the array
cout >> mark[i-1];
```

Displaying Array Elements

```
#include <iostream>
using namespace std;
int main() {
  int numbers[5] = {7, 5, 6, 12, 35};
  cout << "The numbers are: ";

  //  Printing array elements
  // using range based for loop
  for (const int &n : numbers) {
    cout << n << "  ";
  }
  cout << "\nThe numbers are: ";

  //  Printing array elements
  // using traditional for loop
  for (int i = 0; i < 5; ++i) {
    cout << numbers[i] << "  ";
  }
  return 0;
}
```

Display Sum and Average of Array Elements Using for Loop

```
#include <iostream>
using namespace std;
int main() {
  // initialize an array without specifying size
  double numbers[] = {7, 5, 6, 12, 35, 27};

  double sum = 0;
  double count = 0;
  double average;

  cout << "The numbers are: ";

  //  print array elements
  // use of range-based for loop
  for (const double &n : numbers) {
    cout << n << "  ";

    //  calculate the sum
    sum += n;
    // count the no. of array elements
    ++count;
  }

  // print the sum
  cout << "\nTheir Sum = " << sum << endl;

  // find the average
  average = sum / count;
  cout << "Their Average = " << average << endl;
  return 0;
}
```

Output

```
The numbers are: 7  5  6  12  35  27
Their Sum = 92
Their Average = 15.3333
```

- **C++ Multidimensional Arrays**

  In C++, we can create an array of an array, known as a multidimensional array. For example:

  int x[3][4];
  Here, x is a two-dimensional array. It can hold a maximum of 12 elements.

  We can think of this array as a table with 3 rows and each row has 4 columns as shown below.

	Col 1	Col 2	Col 3	Col 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

1. Initialization of two-dimensional array-
   int test[2][3] = { {2, 4, 5}, {9, 0, 19}};

	Col 1	Col 2	Col 3
Row 1	2	4	5
Row 2	9	0	19

2. Initialization of three-dimensional array-

```
int test[2][3][4] = {
                { {3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2} },
                { {13, 4, 56, 3}, {5, 9, 3, 5}, {5, 1, 4, 9} }
            };
```

Two Dimensional Array

```
// C++ Program to display all elements
// of an initialised two dimensional array

#include <iostream>
using namespace std;
int main() {
   int test[3][2] = {{2, -5},
                     {4, 0},
                     {9, 1}};

   // use of nested for loop
   // access rows of the array
   for (int i = 0; i < 3; ++i) {

      // access columns of the array
      for (int j = 0; j < 2; ++j) {
         cout << "test[" << i << "][" << j << "] = " << test[i][j] << endl;
      }
   }
   return 0;
}
```

Output

```
test[0][0] = 2
test[0][1] = -5
test[1][0] = 4
test[1][1] = 0
test[2][0] = 9
test[2][1] = 1
```

3D Array:

The basic concept of printing elements of a 3d array is similar to that of a 2d array. However, since we are manipulating 3 dimensions, we use a nested for loop with 3 total loops instead of just 2:

- the outer loop from i == 0 to i == 1 accesses the first dimension of the array
- the middle loop from j == 0 to j == 2 accesses the second dimension of the array
- the innermost loop from k == 0 to k == 1 accesses the third dimension of the array

Three Dimensional Array

```
// C++ Program to Store value entered by user in
// three dimensional array and display it.

#include <iostream>
using namespace std;
int main() {
   // This array can store upto 12 elements (2x3x2)
   int test[2][3][2] = {
                       {
                          {1, 2},
                          {3, 4},
                          {5, 6}
                       },
                       {
                          {7, 8},
                          {9, 10},
                          {11, 12}
                       }
                    };

   // Displaying the values with proper index.
   for (int i = 0; i < 2; ++i) {
      for (int j = 0; j < 3; ++j) {
         for (int k = 0; k < 2; ++k) {
            cout << "test[" << i << "][" << j << "][" << k << "] = " << test[i][j][k] << endl;
         }
      }
   }
   return 0;
}
```

Output

```
test[0][0][0] = 1
test[0][0][1] = 2
test[0][1][0] = 3
test[0][1][1] = 4
test[0][2][0] = 5
test[0][2][1] = 6
test[1][0][0] = 7
test[1][0][1] = 8
test[1][1][0] = 9
test[1][1][1] = 10
test[1][2][0] = 11
test[1][2][1] = 12
```

- **Passing Array to a Function in C++**

  In C++, we can pass arrays as an argument to a function. And, also we can return arrays from a function.

  The syntax for passing an array to a function is:

```
returnType functionName(dataType arrayName[arraySize]) {
   // code
}
```

  Example-

```
int total(int marks[5]) {
   // code
}
```

  Here, we have passed an int type array named marks to the function total(). The size of the array is 5.

Passing One-dimensional Array to a Function

```
// C++ Program to display marks of 5 students

#include <iostream>
using namespace std;

// declare function to display marks
// take a 1d array as parameter
void display(int m[5]) {
   cout << "Displaying marks: " << endl;

   // display array elements
   for (int i = 0; i < 5; ++i) {
      cout << "Student " << i + 1 << ": " << m[i] << endl;
   }
}

int main() {

   // declare and initialize an array
   int marks[5] = {88, 76, 90, 61, 69};

   // call display function
   // pass array as argument
   display(marks);

   return 0;
}
```

Passing Multidimensional Array to a Function

```
// C++ Program to display the elements of two
// dimensional array by passing it to a function

#include <iostream>
using namespace std;

// define a function
// pass a 2d array as a parameter
void display(int n[][2]) {
   cout << "Displaying Values: " << endl;
   for (int i = 0; i < 3; ++i) {
      for (int j = 0; j < 2; ++j) {
         cout << "num[" << i << "][" << j << "]: " << n[i][j] << endl;
      }
   }
}
int main() {

   // initialize 2d array
   int num[3][2] = {
      {3, 4},
      {9, 5},
      {7, 1}
   };

   // call the function
   // pass a 2d array as an argument
   display(num);
   return 0;
}
```

Output

```
Displaying Values:
num[0][0]: 3
num[0][1]: 4
num[1][0]: 9
num[1][1]: 5
num[2][0]: 7
num[2][1]: 1
```

- **C++ Strings**

C-strings are arrays of type char terminated with null character, that is, \0 (ASCII value of null character is 0).

String is a collection of characters. There are two types of strings commonly used in C++ programming language:

- Strings that are objects of string class (The Standard C++ Library string class)
- C-strings (C-style Strings)

C++ string using string data type

```
#include <iostream>
using namespace std;
int main()
{
   // Declaring a string object
   string str;
   cout << "Enter a string: ";
   getline(cin, str);

   cout << "You entered: " << str << endl;
   return 0;
}
```

In this program, a string str is declared. Then the string is asked from the user.
-Instead of using cin>> or cin.get() function, you can get the entered line of text using getline().
-getline() function takes the input stream as the first parameter which is cin and str as the location of the line to be stored.

C++ program to display a string entered by user.

```
#include <iostream>
using namespace std;
int main()
{
   char str[100];

   cout << "Enter a string: ";
   cin >> str;
   cout << "You entered: " << str << endl;

   cout << "\nEnter another string: ";
   cin >> str;
   cout << "You entered: "<<str<<endl;

   return 0;
```

C++ program to read and display an entire line entered by user.

```
#include <iostream>
using namespace std;
int main()
{
   char str[100];
   cout << "Enter a string: ";
   cin.get(str, 100);
   cout << "You entered: " << str << endl;
   return 0;
}
```

Passing String to a Function

-Strings are passed to a function in a similar way arrays are passed to a function.
-In this program, two strings are asked to enter.
-These are stored in str and str1 respectively, where str is a char array and str1 is a string object.
-Then, we have two functions display() that outputs the string onto the string. The only difference between the two functions is the parameter.
-The first display() function takes char array as a parameter, while the second takes string as a parameter. This process is known as function overloading.

Passing String to a Function

```
#include <iostream>
using namespace std;
void display(char *);
void display(string);
int main()
{
   string str1;
   char str[100];
   cout << "Enter a string: ";
   getline(cin, str1);
   cout << "Enter another string: ";
   cin.get(str, 100, '\n');
   display(str1);
   display(str);
   return 0;
}

void display(char s[])
{
   cout << "Entered char array is: " << s << endl;
}

void display(string s)
{
   cout << "Entered string is: " << s << endl;
}
```

Output

```
Enter a string:  Programming is fun.
Enter another string:  Really?
Entered string is: Programming is fun.
Entered char array is: Really?
```

C++ Structures

Structure is a collection of variables of different data types under a single name. It is similar to a class in that, both holds a collection of data of different data types.

How to declare a structure in C++ programming?

The struct keyword defines a structure type followed by an identifier (name of the structure).

Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person
{
   char name[50];
   int age;
   float salary;
};
```

Here a structure person is defined which has three members: name, age and salary.

How to define a structure variable?

-Once you declare a structure person as above. You can define a structure variable as:

```
Person bill;
```

-Here, a structure variable bill is defined which is of type structure Person.

When structure variable is defined, only then the required memory is allocated by the compiler.

-Considering you have either 32-bit or 64-bit system, the memory of float is 4 bytes, memory of int is 4 bytes and memory of char is 1 byte.

-Hence, 58 bytes of memory is allocated for structure variable bill.

How to access members of a structure?

The members of structure variable is accessed using a **dot (.)** operator.

Suppose, you want to access age of structure variable bill and assign it 50 to it. You can perform this task by using following code below:

```
bill.age = 50;
```

-Here in this program, a structure Person is declared which has three members name, age and salary.

-Inside main() function, a structure variable p1 is defined. Then, the user is asked to enter information and data entered by user is displayed.

C++ Program to assign data to members of a structure variable and display it.

```
#include <iostream>
using namespace std;

struct Person
{
   char name[50];
   int age;
   float salary;
};

int main()
{
   Person p1;

   cout << "Enter Full name: ";
   cin.get(p1.name, 50);
   cout << "Enter age: ";
   cin >> p1.age;
   cout << "Enter salary: ";
   cin >> p1.salary;

   cout << "\nDisplaying Information." << endl;
   cout << "Name: " << p1.name << endl;
   cout <<"Age: " << p1.age << endl;
   cout << "Salary: " << p1.salary;

   return 0;
}
```

Output

```
Enter Full name: Pradnya Thakre
Enter age: 25
Enter salary: 20000

Displaying Information.
Name: Pradnya Thakre
Age: 25
Salary: 20000
```

Passing structure to function in C++

A structure variable can be passed to a function in similar way as normal argument.

C++ Structure and Function

```
#include <iostream>
using namespace std;
struct Person {
   char name[50];
   int age;
   float salary;
};
void displayData(Person);   // Function declaration
int main() {
   Person p;
   cout << "Enter Full name: ";
   cin.get(p.name, 50);
   cout << "Enter age: ";
   cin >> p.age;
   cout << "Enter salary: ";
   cin >> p.salary;

   // Function call with structure variable as an argument
   displayData(p);
   return 0;
}
void displayData(Person p) {
   cout << "\nDisplaying Information." << endl;
   cout << "Name: " << p.name << endl;
   cout <<"Age: " << p.age << endl;
   cout << "Salary: " << p.salary;
}
```

Output

```
Enter Full name: Bill Jobs
Enter age: 55
Enter salary: 34233.4
Displaying Information.
Name: Bill Jobs
Age: 55
Salary: 34233.4
```

In this program, user is asked to enter the name, age and salary of a Person inside main() function.

Then, the structure variable p is to passed to a function using.

displayData(p);

The return type of displayData() is void and a single argument of type structure Person is passed.

Then the members of structure p is displayed from this function.

Returning structure from function in C++

```
#include <iostream>
using namespace std;
struct Person {
   char name[50];
   int age;
   float salary;
};

Person getData(Person);
void displayData(Person);
int main() {

   Person p, temp;
   temp = getData(p);
   p = temp;
   displayData(p);
   return 0;
}
Person getData(Person p) {

   cout << "Enter Full name: ";
   cin.get(p.name, 50);

   cout << "Enter age: ";
   cin >> p.age;

   cout << "Enter salary: ";
   cin >> p.salary;
   return p;
}

void displayData(Person p) {
   cout << "\nDisplaying Information." << endl;
   cout << "Name: " << p.name << endl;
   cout <<"Age: " << p.age << endl;
   cout << "Salary: " << p.salary;
}
```