PART -A

1. **Define : objects and classes.**

Object :

An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

- When we define a class we define just a blueprint or we just map data members and member functions.
- The declaration of class develops a template but data members cannot be manipulated unless the object of its type is created.

Class:

The building block of C++ that leads to Object Oriented programming is a Class. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

For Example: Consider the Class of Cars.

2. **Define inline function.**

Defining member function is to replace the function declaration by the actual function definition inside the class. It is treated as inline function.

The inline functions are a C++ enhancement feature to increase the execution time of a program. Functions can be instructed to compiler to make them inline so that compiler can replace those function definition wherever those are being called. What is meant by Constructors and list various types of constructors.

3. **What is meant by Constructors and list various types of constructors**.

1) Constructor is used for Initializing the values to the data members of the Class.

2) Constructor is that whose name is same as name of class.

3) Constructor gets Automatically called when an object of class is created.

4) Constructors never have a Return Type even void.

5) Constructor are of Default , Parameterized and Copy Constructors.

Types :

- Default constructor
- Parameterized constructor
- Copy constructor
- Destructor

4. **What do you mean by type conversions?**

A type cast is basically a conversion from one type to another. There are two types of type conversion:

(i) Implicit Type Conversion:

- Done by the compiler on its own, without any external trigger from the user.
- All the data types of the variables are upgraded to the data type of the variable with largest data type.

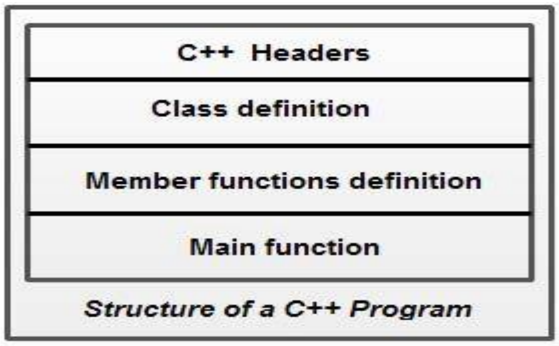
(ii) Explicit Type Conversion:

- This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.
- By explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.

  Syntax: type (expression)

5. Give the structure of C++ program.

Programs are a sequence of instructions or statements. These statements form the structure of a C++ program. C++ program structure is divided into various sections, namely, headers, class definition, member functions definitions and main function.



Structure of a C++ Program

6. What is scope resolution operator(::)?

The :: (scope resolution) operator is used to get hidden names due to variable scopes so that you can still use them. The scope resolution operator can be used as both unary and binary. For example, if you have a global variable of name my_var and a local variable of name my_var, to access global my_var, you'll need to use the scope resolution operator.

7. What is a copy constructor?

A copy constructor is a member function which initializes an object using another object of the same class. A copy constructor has the following general function prototype:

```
ClassName (const ClassName &old_obj);
```

8. What is meant by Destructor? Give example.

A destructor is a special member function that works just opposite to constructor, unlike constructors that are used for initializing an object, destructors destroy (or delete) the object.

Syntax

```
~class_name()
{
  //Some code
}
```

9. List the applications of object oriented programming.

Main application areas of OOP are:

- User interface design such as windows, menu.
- Real Time Systems
- Simulation and Modeling
- Object oriented databases
- AI and Expert System
- Neural Networks and parallel programming
- Decision support and office automation systems etc.

10. Write the few words about object oriented languages.

The basic thing which are the essential feature of an object oriented programming are Inheritance, Polymorphism and Encapsulation. Any programming language that supports these feature completely are complete Object-oriented programming language.

- Inheritance is used to provide the concept of code-reusability.
- Polymorphism makes a language able to perform different task at different instance.
- Encapsulation makes data abstraction (security or privacy to data) possible. In object-oriented programming language, Encapsulation is achieved with the help of a class.

11. What is polymorphism.

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Ex:

A person at a same time can have different characteristic. Like a man at a same time is a father, a husband, a employee.

In C++ polymorphism is mainly divided into two types:

- Compile time Polymorphism
- Runtime Polymorphism

12. What is friend function.

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

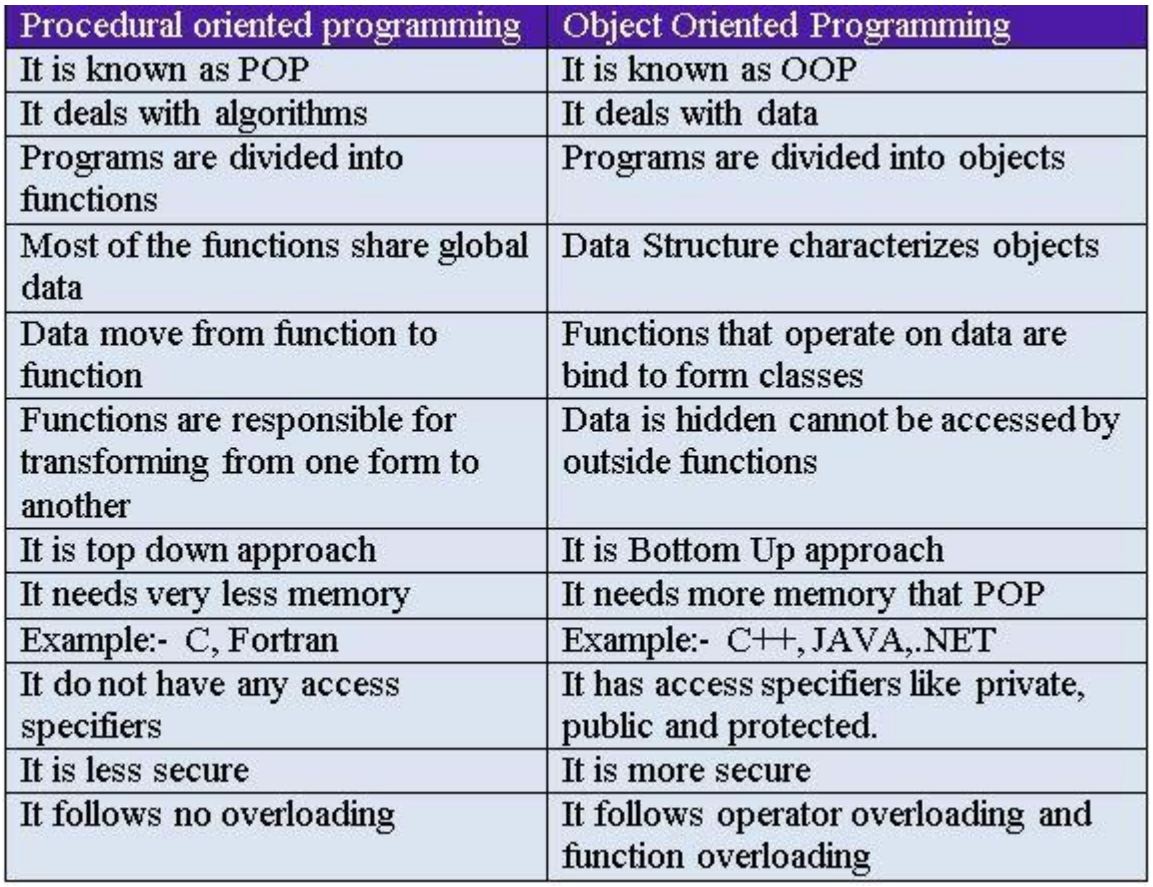
A friend function can be:

a) A method of another class

b) A global function

PART - B

13. a) Compare procedural and Object-oriented Programming.



Procedural oriented programming	Object Oriented Programming
It is known as POP	It is known as OOP
It deals with algorithms	It deals with data
Programs are divided into functions	Programs are divided into objects
Most of the functions share global data	Data Structure characterizes objects
Data move from function to function	Functions that operate on data are bind to form classes
Functions are responsible for transforming from one form to another	Data is hidden cannot be accessed by outside functions
It is top down approach	It is Bottom Up approach
It needs very less memory	It needs more memory that POP
Example:- C, Fortran	Example:- C++, JAVA,.NET
It do not have any access specifiers	It has access specifiers like private, public and protected.
It is less secure	It is more secure
It follows no overloading	It follows operator overloading and function overloading

13.b) Explain C++ tokens briefly.

A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

- Keywords
- Identifiers
- Constants
- Strings
- Special Symbols
- Operators

Keyword:

Keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. C++ language supports 32 keywords which are given below:

auto break case char continue const default do double else enum extern float for goto if int long register return short signed sizeof static struct switch typedef union unsigned void volatile while

Identifiers:

Identifiers are used as the general terminology for naming of variables, functions and arrays. These are user defined names consisting of arbitrarily long sequence of letters and digits with either a letter or the underscore(_) as a first character. Identifier names must differ in spelling and case from any keywords.

There are certain rules that should be followed while naming c identifiers:

- They must begin with a letter or underscore(_).
- They must consist of only letters, digits, or underscore. No other special character is allowed.
- It should not be a keyword.
- It must not contain white space.

Some examples of C++ identifiers:

_A9	Valid
Temp.var	Invalid as it contains special character other than the underscore
void	Invalid as it is a keyword

Constants:

Constants are also like normal variables. But, only difference is, their values cannot be modified by the program once they are defined. Constants refer to fixed values. They are also called as literals.

Constants may belong to any of the data type.

Syntax:

const data_type variable_name; (or) const data_type *variable_name;

Types of Constants:

- Integer constants – Example: 0, 1, 1218, 12482
- Real or Floating point constants – Example: 0.0, 1203.03, 30486.184
- Octal & Hexadecimal constants – Example: octal: $(013\ )_8 = (11)_{10}$, Hexadecimal: $(013)_{16} = (19)_{10}$
- Character constants -Example: ‘a’, ‘A’, ‘z’
- String constants -Example: “Ramesh”

Strings:

Strings are nothing but an array of characters ended with a null character (‘\0’).This null character indicates the end of the string. Strings are always enclosed in double quotes. Whereas, a character is enclosed in single quotes in C and C++.

<u>Declarations for String:</u>

char string[20] = {‘r’, ’a’, ‘m’, ‘e’, ‘s’, ‘h’};
char string[20] = “ramesh”;
char string [] = “ramesh”;

Special Symbols:

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose.

[] () {}, ; * = #

Operators:

Operators are symbols that triggers an action when applied to C++ variables and other objects. The data items on which operators act upon are called operands.

Depending on the number of operands that an operator can act upon, operators can be classified as follows:

Unary Operators:

Those operators that require only single operand to act upon are known as unary operators.For Example increment and decrement operators

Binary Operators:

Those operators that require two operands to act upon are called binary operators. Binary operators are classified into :

- Arithmetic operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Conditional Operators
- Bitwise Operators

Ternary Operators:

These operators requires three operands to act upon. For Example Conditional operator(?:).

14. a) What is Operator overloading? Explain how binary operator can be overloaded with example.

Operator Overloading :

In C++, we can make operators to work for user defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading.
For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +.
Other example classes where arithmetic operators may be overloaded are Complex Number, Fractional Number, Big Integer, etc.

Binary operator in overloading :

Binary operator is an operator that takes two operand(variable). Binary operator overloading is similar to unary operator overloading except that a binary operator overloading requires an additional parameter.

- Binary Operators
- Arithmetic operators (+, -, *, /, %)
- Arithmetic assignment operators (+=, -=, *=, /=, %=)
- Relational operators (>, <, >=, <=, !=, ==)

Ex :

```
#include<iostream.h>
#include<conio.h>
class Rectangle
{
        int L,B;
        public:
        Rectangle()          //Default Constructor
        {
                L = 0;
                B = 0;
        }
        Rectangle(int x,int y)          //Parameterize Constructor
        {
                L = x;
                B = y;
        }
         Rectangle operator+(Rectangle Rec)
        {
                Rectangle R;
                        R.L = L + Rec.L;
                        R.B = B + Rec.B;
                return R;
        }
        void Display()
        {
                cout<<"\n\tLength : "<<L;
                cout<<"\n\tBreadth : "<<B;
        }
};
void main()
{
        Rectangle R1(2,5),R2(3,4),R3;
        cout<<"\n\tRectangle 1 : ";
        R1.Display();
        cout<<"\n\n\tRectangle 2 : ";
        R2.Display();
        R3 = R1 + R2;      Statement 1
        cout<<"\n\n\tRectangle 3 : ";
        R3.Display();
}
```

Output :

```
Rectangle 1 :
L : 2
B : 5

Rectangle 2 :
L : 3
B : 4

Rectangle 3 :
L : 5
B : 9
```

14. b) Describe the type conversions with examples.

A type cast is basically a conversion from one type to another. There are two types of type conversion:

1. **Implicit Type Conversion**

   It is also known as 'automatic type conversion'. Done by the compiler on its own, without any external trigger from the user. Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data. All the data types of the variables are upgraded to the data type of the variable with largest data type.

It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

Ex :

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

        int main()
        {
        int x = 10; // integer x
        char y = 'a'; // character c
        x = x + y;
        float z = x + 1.0;

        cout << "x = " << x << endl
                << "y = " << y << endl
                << "z = " << z << endl;
        return 0;
}
```

Output :

```
x = 107
y = a
z = 108
```

15. Explicit Type Conversion:

This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.

In C++, it can be done by two ways:

- Converting by assignment: This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.

Syntax: (type) expression

Ex :

```
#include <iostream>
using namespace std;
int main()
{
	double x = 1.2;
	int sum = (int)x + 1;
	cout << "Sum = " << sum;
	return 0;
}
```

Output : Sum = 2

- Conversion using Cast operator: A Cast operator is an unary operator which forces one data type to be converted into another data type.

C++ supports four types of casting:

- Static Cast
- Dynamic Cast
- Const Cast
- Reinterpret Cast

Ex :

```
	#include <iostream>
	using namespace std;
	int main()
	{
		float f = 3.5;
		int b = static_cast<int>(f);
		cout << b;
	}
```

16. a) Explain the basic data types in C++.

All variables use data-type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data it can store.

Data types in C++ is mainly divided into two types:

1. Primitive Data Types:

These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char , float, bool etc. Primitive data types available in C++ are:

- Integer
- Character
- Boolean
- Floating Point
- Double Floating Point
- Valueless or Void

2. Abstract or user defined data type:

These data types are defined by user itself. Like, defining a class in C++ or a structure.

This one describes primitive data types available in C++

Integer:

Keyword used for integer data types is int. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.

Character:

Character data type

Character data type is used for storing characters. Keyword used for character data type is char. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.

Boolean:

Boolean data type is used for storing boolean or logical values. A boolean variable can store either true or false. Keyword used for boolean data type is bool.

Floating Point:

Floating Point data type is used for storing single precision floating point values or decimal values. Keyword used for floating point data type is float. Float variables typically requires 4 byte of memory space.

Double Floating Point:

Double Floating Point data type is used for storing double precision floating point values or decimal values. Keyword used for double floating point data type is double. Double variables typically requires 8 byte of memory space.

void:

Void means without any value. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.

DATA TYPE	SIZE (IN BYTES)	RANGE
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
Int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
Float	4	
Double	8	
long double	12	

16. b) Explain inline function with example.

C++ inline function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.

Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality.

To inline a function, place the keyword inline before the function name and define the function before any calls are made to the function. The compiler can ignore the inline qualifier in case defined function is more than a line.

A function definition in a class definition is an inline function definition, even without the use of the inline specifier.

Syntax :

```
inline return-type function-name(parameters)
{
    // function code
}
```

Inline functions provide following advantages:

1) Function call overhead doesn't occur.

2) It also saves the overhead of push/pop variables on the stack when function is called.

3) It also saves overhead of a return call from a function.

4) When you inline a function, you may enable compiler to perform context specific optimization on the body of function. Such optimizations are not possible for normal function calls. Other optimizations can be obtained by considering the flows of calling context and the called context.

5) Inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function call preamble and return.

Inline function disadvantages:

1) The added variables from the inlined function consumes additional registers, After in-lining function if variables number which are going to use register increases than they may create overhead on register variable resource utilization. This means that when inline function body is substituted at the point of function call, total number of variables used by the function also gets inserted. So the number of register going to be used for the variables will also get increased. So if after function inlining variable numbers increase drastically then it would surely cause an overhead on register utilization.

2) If you use too many inline functions then the size of the binary executable file will be large, because of the duplication of same code.

3) Too much inlining can also reduce your instruction cache hit rate, thus reducing the speed of instruction fetch from that of cache memory to that of primary memory.

4) Inline function may increase compile time overhead if someone changes the code inside the inline function then all the calling location has to be recompiled because compiler would require to replace all the code once again to reflect the changes, otherwise it will continue with old functionality.

5) Inline functions may not be useful for many embedded systems. Because in embedded systems code size is more important than speed.

6) Inline functions might cause thrashing because inlining might increase size of the binary executable file. Thrashing in memory causes performance of computer to degrade.

The following program demonstrates the use of use of inline function.

```
#include <iostream>
using namespace std;
class operation
{
        int a,b,add,sub,mul;
        float div;
public:
        void get();
        void sum();
        void difference();
        void product();
        void division();
};
inline void operation :: get()
{
        cout << "Enter first value:";
        cin >> a;
        cout << "Enter second value:";
        cin >> b;
}

inline void operation :: sum()
{
        add = a+b;
        cout << "Addition of two numbers: " << a+b << "\n";
}

inline void operation :: difference()
{
        sub = a-b;
        cout << "Difference of two numbers: " << a-b << "\n";
}
int main()
{
        cout << "Program using inline function\n";
        operation s;
        s.get();
        s.sum();
        s.difference();
        return 0;
}
```

17. a) Explain the constructor with an example.

A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object create. It is special member function of the class.

A constructor is different from normal functions in following ways:

- Constructor has same name as the class itself

- Constructors don’t have return type
- A constructor is automatically called when an object is created.
- If we do not specify a constructor, C++ compiler generates a default constructor for us.

Types of Constructor:

1. **Default Constructors:**

   Default constructor is the constructor which doesn’t take any argument. It has no parameters.

2. **Parameterized Constructors:**

   It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor’s body, use the parameters to initialize the object.

3. **Copy Constructor:**

   A copy constructor is a member function which initializes an object using another object of the same class. Detailed article on Copy Constructor.

Syntax:

```
class class-name
{
   Access Specifier :
   Member - Variables
   Member - Functions
public:
   class-name() {
      // Constructor code
   }

   //... other Variables & Functions
}
```

Ex:

```
include<conio.h>
using namespace std;
class Example {
   int a, b;
public:
   Example()
{
      a = 10;
      b = 20;
```

```
        cout << "I am Constructor\n";
    }
    void Display() {
        cout << "Values :" << a << "\t" << b;
    }
};
int main()
{
    Example Object;
    Object.Display();
    getch();
    return 0;
}
```

Ouptut:

```
    I am Constructor
    Values :10      20
```

17. **b) Explain Classes and Objects with example.**

Class:

The building block of C++ that leads to Object Oriented programming is a Class. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

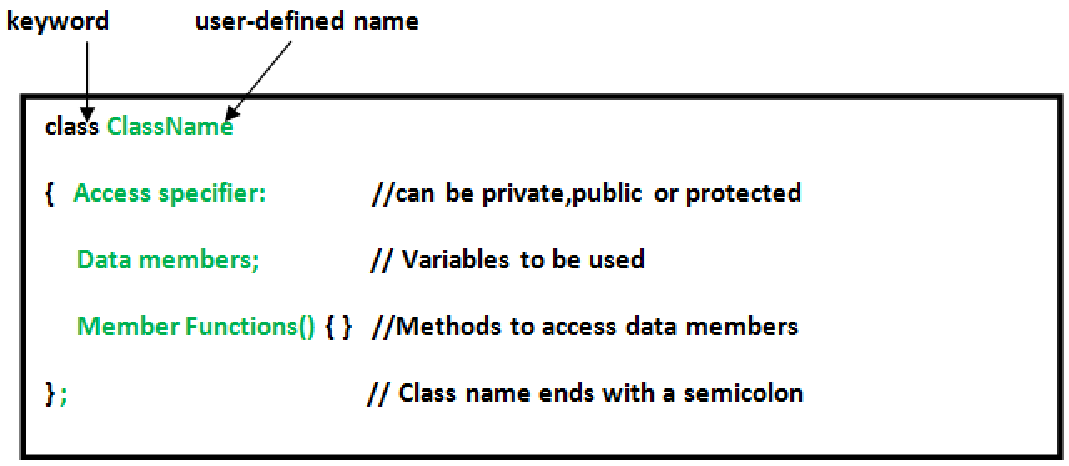
A Class is a user defined data-type which has data members and member functions. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.

Object :

An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Defining Class and Declaring Objects

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.



```
class ClassName

{  Access specifier:         //can be private,public or protected

   Data members;             // Variables to be used

   Member Functions() { }    //Methods to access data members

};                           // Class name ends with a semicolon
```

Declaring Objects:

When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:

ClassName ObjectName;

Accessing data members and member functions: The data members and member functions of class can be accessed using the dot('.') operator with the object.

Accessing Data Members

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

This access control is given by Access modifiers in C++. There are three access modifiers: public, private and protected.

Ex:

```
#include <iostream.h>
using namespace std;
class test
{
        public:
        string myname;
        void printname()
        {
        cout << "My name is: " << myname;
        }
};
int main() {

        test obj1;
        obj1.myname = "ram";
        obj1.printname();
        return 0;
}
```

Output:

```
My name is : ram
```

18. a) Describe the different styles of function prototypes.

Prototype of a function is the function without its body. Prototyping of a function makes the program easier to understand and makes the program short and more convenient.

Function prototype are usually written at the beginning of a program, ahead of any programmer-defined functions including main(). We can add as many parameter to the functions as much programmer needs

Making function prototyping and using includes 3 steps :-
1) Function Prototype;
2) Function Definition;
3) Function Call.

Syntax :

```
<ReturnType> <FunctionName> (type 1 arg. 1, type 2 arg. 2,...type n arg n);
```

Define function :

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

C++ function definition consists of a function header and a function body. Here are all the parts of a function −

Return Type − A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
Function Name − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
Parameters − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
Function Body − The function body contains a collection of statements that define what the function does.

Calling function

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

Ex:

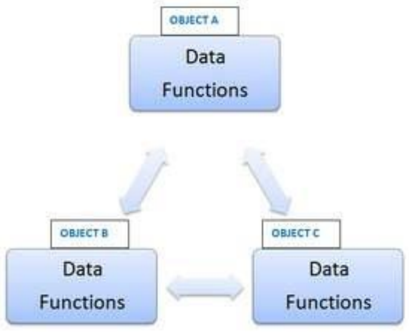
```
#include <iostream>
using namespace std;
int max(int num1, int num2);
int main ()
{
  int a = 100;
  int b = 200;
  int ret;
  ret = max(a, b);
  cout << "Max value is : " << ret << endl;
  return 0;
}

int max(int num1, int num2)
{
  int result;
  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

18. b) List some of the striking features of OOP.

Object oriented programming(OOP), treats data as a critical element in the program and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it to flow freely around the system. It allows decomposition of a problem into a number of entities called Objects. The data of an object can be accessed only by the functions associated with the object.



***OOP Characteristics*:**

- Objects
- Classes
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

***Objects*:**

Objects are the basic run-time entities in an object-oriented programming. They may represents a person, a place, a bank account, a table of data or any item that the program has to handle. They may also represents user-defined data such as vector, time and lists. When the program is executed, the object interact by sending message to one another.

***Classes*:**

Objects contain data, and code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the help of class. In fact, objects are variable of the type class. Once a class has been defined, we can create a number of objects belonging to that class. A class is a collection of objects of similar type.
Ex. Fruit mango;

***Data Abstraction*:**

Abstractions refer to the act of representing essential features without including background details or explanation. They are commonly known as Abstraction Data Type(ADT).

***Encapsulation*:**

The wrapping up of data and functions into single unit is known as *encapsulation*. Data encapsulation is a 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 and the program.

***Inheritance*:**

Inheritance is the process by which objects of one class acquire the properties of object of another class. The class whose members are inherited is called the Base class and the class that inherits those members is called Derived class. It supports class of hierarchical classification.

***Polymorphism*:**

Polymorphism is another OOP concept. Polymorphism means the ability to take more than one form. An operation may exhibit different behaviors at different instances.

19. a) Explain about multiple constructor with example in C++.

Definition

In C++, Constructor is automatically called when an object(an instance of the lass) create. It is the special member function of the class. Which constructor has arguments is called Parameterized Constructor.

One Constructor overload another constructor is called Constructor Overloading .
It has the same name of the class.

- It must be a public member.
- No Return Values.
- Default constructors are called when constructors are not defined for the classes.

Syntax:

```
class class_name
{
   Access Specifier :
   Member_Variables
   Member_Functions
public:
   class_name()
{
// Constructor code
   }
   class_name(variables)
 {
// Constructor code
   }
   ... other Variables & Functions
}
```

Ex:

```
#include<iostream>
#include<conio.h>
using namespace std;
class Example
{
   int a, b;
public:
   Example()
{
      a = 50;
      b = 100;
      cout << "\nIm Constructor";
   }
   Example(int x, int y)
{
      a = x;
      b = y;
      cout << "\nIm Constructor";
   }
   void Display()
 {
      cout << "\nValues :" << a << "\t" << b;
   }
};
int main()
{
   Example Object(10, 20);
   Example Object2;
   Object.Display();
   Object2.Display();
   getch();
   return 0;
}
```

19. b) Explain pointers to members with example.

<u>Pointer to Data Members of Class</u>

We can use pointer to point to class's data members (Member variables).

Syntax :

```
datatype class_name :: *pointer_name;
```

Syntax for Assignment :

```
pointer_name = &class_name :: datamember_name;
```

Both declaration and assignment can be done in a single statement too.

```
datatype class_name::*pointer_name = &class_name::datamember_name ;
```

Using Pointers with Objects

For accessing normal data members we use the dot . operator with object and -> with pointer to object. But when we have a pointer to data member, we have to dereference that pointer to get what its pointing to, hence it becomes,

```
Object.*pointerToMember
```

and with pointer to object, it can be accessed by writing,

```
ObjectPointer->*pointerToMember
```

Example:

```
class Data
{
   public:
   int a;
   void print()
   {
      cout << "a is "<< a;
   }
};
int main()
{
   Data d, *dp;
   dp = &d;     // pointer to object
   int Data::*ptr=&Data::a;   // pointer to data member 'a'
   d.*ptr=10;
   d.print();
   dp->*ptr=20;
   dp->print();
}
```

Output :

```
a is 10
a is 20
```

PART – C

20. Describe the various control structures in C++ with suitable examples.

Control Statements, Looping and Iteration

(i) Conditional structure: if and else
(ii) For Loop
(iii) While Loop
(iv) Do While
(v) Switch Statement and Break

(i) Conditional structure: if and else

- The if statement executes based test expression inside the braces.
- If statement expression is to true, If body statements are executed and Else body statements are skipped.
- If statement expression is to false If body statements are skipped and Else body statements are executed.
- Simply, Block will execute based on If the condition is true or not.

if and else Syntax

```
if (expression) // Body will execute if expression is true or non-zero
{
        //If Body statements
}else
{
        //Else Body statements
}
```

Ex:

```
        if (i == 3)
 {
                doSomething();
        }
        else
{
                doSomethingElse();
        }
```

Ex:
```
        #include<iostream>
        #include<conio.h>
        using namespace std;
        int main()
{
        int a;
        cout<<"Enter the Number :";
        cin>>a;
        if(a > 10)
        {
          cout<<a <<" Is Greater than 10";
        }
        else
        {
          cout<<a<<" Is Less than/Equal to 10";
        }
        getch();
        return 0;
}
```

The for statement

The C++ for statement has the following form:

Syntax:

```
for  (expression1;Condition;expression2)
        statement;
for  (expression1;Condition;expression2) {
        block of statements
}
```

expression1 initialises; expression2 is the terminate test; expression3 is the modifier;

Ex:

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
   int x=3;
        for (x=3;x>0;x--)
        {
                cout<<"x="<<x<<endl;
        }
    getch();
    return 0;
}
```

The while statement

The while statement is similar to those used in other languages although more can be done with the expression statement.

Syntax:

```
while (expression)
        statement;
while (expression)
        block of statements
}
```

Ex:

```
#include<iostream>
#include<conio.h>
using namespace std;
//Main Function
int main()
{
    int x=3;
            while (x>0)
            {
                        cout<<"x="<<x<<endl;
                        x--;
            }
     getch();
     return 0;
}
```

The do-while statement

Syntax:

```
do
{
   statement;
}
while (expression);
```

Ex:

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
    int x=3;
             do {
               cout<<"x="<<x<<endl;
               x--;
             }while (x>0);
     getch();
     return 0;
}
```

Switch Case Statement

- We can use switch statements alternative for an if..else ladder.
- The switch statement is often faster than nested if...else Ladder.
- Switch statement syntax is well structured and easy to understand.

Syntax :

```
switch ( <expression> or <variable> )
 {
case value1:
  break;

case value2:
  break;
...
default:
  Code to execute for not match case
  break;
}
```

21. What is Class in C++? Explain with syntax and example program.

Class:

The building block of C++ that leads to Object Oriented programming is a Class. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

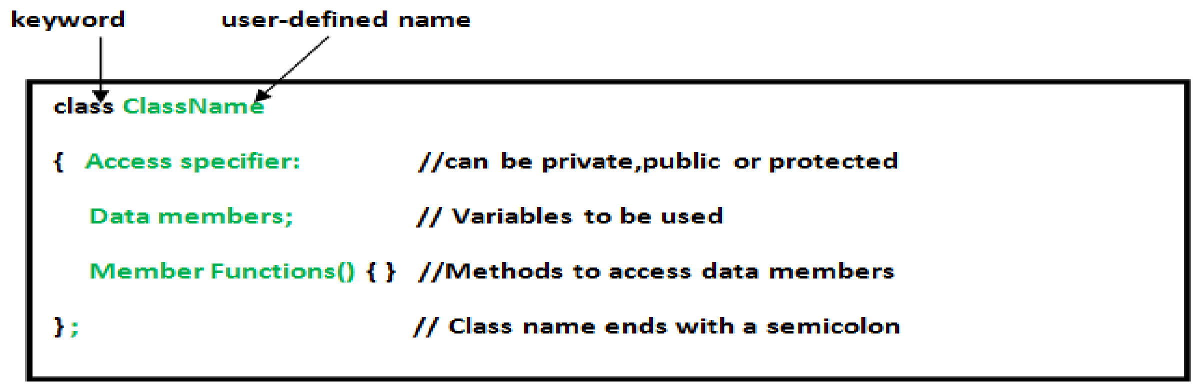
For Example:

Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

A Class is a user defined data-type which has data members and member functions. Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.

In the above example of class Car, the data member will be speed limit, mileage etc and member functions can be apply brakes, increase speed etc.

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.



```
class ClassName
{  Access specifier:          //can be private,public or protected
   Data members;              // Variables to be used
   Member Functions() { }     //Methods to access data members
};                            // Class name ends with a semicolon
```

Accessing Data Members

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

This access control is given by Access modifiers in C++. There are three access modifiers :

- public,
- private and
- protected.

```
#include <iostream.h>
using namespace std;
class test
{
        public:
        string myname;
        void printname()
        {
        cout << "My name is: " << myname;
        }
};
int main() {

        test obj1;
        obj1.myname = "ram";
        obj1.printname();
        return 0;
}
```

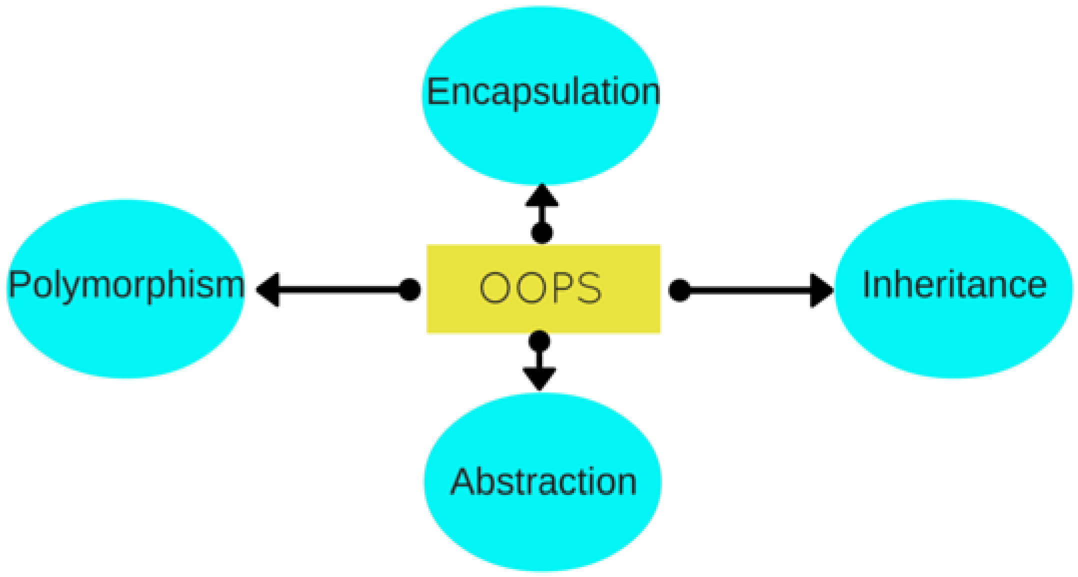
Output:

```
My name is : ram
```

22. Explain OOPs concept with its benefits.

Object Oriented Programming

Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.



There are a few principle concepts that form the foundation of object-oriented programming −

Object

This is the basic unit of object oriented programming. That is both data and function that operate on data are bundled as a unit called as object.

Class

When define a class, you define a blueprint for an object. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

Abstraction

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.

For example, a database system hides certain details of how data is stored and created and maintained. Similar way, C++ classes provides different methods to the outside world without giving internal detail about those methods and data.

Encapsulation

Encapsulation is placing the data and the functions that work on that data in the same place. While working with procedural languages, it is not always clear which functions work on which variables but object-oriented programming provides you framework to place the data and the relevant functions together in the same object.

Inheritance

One of the most useful aspects of object-oriented programming is code reusability. As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class.

Polymorphism

The ability to use an operator or function in different ways in other words giving different meaning or functions to the operators or functions is called polymorphism. Poly refers to many. That is a single function or an operator functioning in many ways different upon the usage is called polymorphism.

Overloading

The concept of overloading is also a branch of polymorphism. When the exiting operator or function is made to operate on new data type, it is said to be overloaded.

Benefits of OOPs:

1. **Data Re-usability**:- "Write a once and use multiple time" we can achieve this by using class. Once We Write a class we can bus it number of time by creating the object for class.
2. **Data Redundancy**:- Inheritance is the good feature for data redundancy if you need a same functionality in multiple class you can write a common class for the same functionality and inherit that class to sub class.
3. **Easy Maintenance**:- It is easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
4. **Data hiding**:- Implementation details are hidden from other modules and other modules has a clearly defined interface.
5. **Security**:- Using data hiding and abstraction we are providing necessary data only it mean we are maintaining security.

23. What is friend function? Explain the friend function with example and also list its characteristics.

Definition :

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend function can be:

a) A method of another class

b) A global function

Characteristics :

- A friend function is not in the scope of the class, in which it has been declared as friend.

- It cannot be called using the object of that class.

- It can be invoked like a normal function without any object.

- Unlike member functions, it cannot use the member names directly.

- It can be declared in public or private part without affecting its meaning.

- Usually, it has objects as arguments.

Ex:

//program to find the greatest number among two numbers using the friend function

```
#include <iostream.h>
#include <conio.h>
class Num
{
        int x,y;
   public:
      void num()
      {
          cout<<"Enter the first number = "<<endl;
         cin>>x;
         cout<<"Enter the second number = "<<endl;
         cin>>y;
         cout<<"The entered numbers are "<< x <<" and "<<y<<endl;
      }
      friend void max(Num m);
};
void max(Num m)
      {
          if(m.x>m.y)
         {
             cout<<"The greatest number among "<<m.x<<" and "<<m.y<<" is "<<m.x<<endl;
         }
         else
         {
             cout<<"The greatest number among "<<m.x<<" and "<<m.y<<" is "<<m.y<<endl;
         }
      }
void main()
{
    Num a;
   a.num();
   max(a);
   getch();
}
```

Output :

```
            Enter the first number =3
            Enter the second number =5
            The greatest number among 3 and 5 is 5
```

24. Explain the concept of function in C++ with syntax and example.

Function :

In C++, a function is a group of statements that is given a name, and which can be called from some point of the program. The most common syntax to define a function is:

type name (parameter1, parameter2, ...) { statements }

Where:

- type is the type of the value returned by the function.
- name is the identifier by which the function can be called.
- parameters (as many as needed): Each parameter consists of a type followed by an identifier, with each parameter being separated from the next by a comma.

Ex:

```
#include <iostream>
using namespace std;
int max(int num1, int num2);
int main ()
{
  int a = 100;
  int b = 200;
  int ret;
  ret = max(a, b);
  cout << "Max value is : " << ret << endl;
  return 0;
}
int max(int num1, int num2)
{
  int result;
  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

<u>Functions with no type. The use of void</u>

Syntax:

type name (argument1, argument2 ...) { statements }

Requires the declaration to begin with a type.

Ex:

```
#include <iostream>
using namespace std;
void printmessage ()
{
 cout << "I'm a function!";
}
int main ()
{
 printmessage ();
}
```

The return value of main

The return type of main is int, but most examples in this and earlier chapters did not actually return any value from main.

If the execution of main ends normally without encountering a return statement the compiler assumes the function ends with an implicit return statement:

```
return 0;
```

Arguments passed by value and by reference

Arguments have always been passed by value. This means that, when calling a function, what is passed to the function are the values of these arguments on the moment of the call, which are copied into the variables represented by the function parameters. For example

```
include <iostream>
using namespace std;
void duplicate (int& a, int& b, int& c)
{
  a*=2;
  b*=2;
  c*=2;
}
int main ()
{
  int x=1, y=3, z=7;
  duplicate (x, y, z);
  cout << "x=" << x << ", y=" << y << ", z=" << z;
  return 0;
}
```

const references

Calling a function with parameters taken by value causes copies of the values to be made. This is a relatively inexpensive operation for fundamental types such as int, but if the parameter is of a large compound type, it may result on certain overhead. For example, consider the following function:

```
string concatenate (string a, string b)
{
  return a+b;
}
```

This function takes two strings as parameters (by value), and returns the result of concatenating them. By passing the arguments by value, the function forces a and b to be copies of the arguments passed to the function when it is called. And if these are long strings, it may mean copying large quantities of data just for the function call.

But this copy can be avoided altogether if both parameters are made references:

```
string concatenate (string& a, string& b)
{
  return a+b;
}
```

Inline function

Defining member function is to replace the function declaration by the actual function definition inside the class. It is treated as inline function.

The inline functions are a C++ enhancement feature to increase the execution time of a program. Functions can be instructed to compiler to make them inline so that compiler can replace those function definition wherever those are being called. What is meant by Constructors and list various types of constructors.

For example, the concatenate function above may be declared inline as:

```
inline string concatenate (const string& a, const string& b)
{
  return a+b;
}
```

25. Describe the various types of constructors in C++.

A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class.

A constructor is different from normal functions in following ways:

- Constructor has same name as the class itself
- Constructors don't have return type
- A constructor is automatically called when an object is created.
- If we do not specify a constructor, C++ compiler generates a default constructor for us.

Types of Constructors:

- ➢ Default constructor
- ➢ Parameterized constructor
- ➢ Copy constructor
- ➢ Destructor

Default Constructors:

Default constructor is the constructor which doesn't take any argument. It has no parameters.

```
Ex: #include <iostream>
    using namespace std;
    class construct
 {
public:
    int a, b;
    construct()
    {
            a = 10;
            b = 20;
    }
};
int main()
{
    construct c;
    cout << "a: " << c.a << endl
            << "b: " << c.b;
    return 1;
}
```

Parameterized Constructors:

It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

Ex:

```
#include <iostream>
using namespace std;
class Point
 {
private:
     int x, y;
public:
     Point(int x1, int y1)
     {
               x = x1;
               y = y1;
     }
     int getX()
     {
               return x;
     }
     int getY()
     {
               return y;
     }
};
int main()
{
     Point p1(10, 15);
     cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
     return 0;
}
```

Copy Constructor:

A copy constructor is a member function which initializes an object using another object of the same class. A copy constructor has the following general function prototype:

ClassName (const ClassName &old_obj);

Ex:

```
#include<iostream>
using namespace std;
class Point
{
private:
     int x, y;
public:
     Point(int x1, int y1)
 {
 x = x1;
 y = y1;
}
Point(const Point &p2)
{
x = p2.x;
 y = p2.y;
}
     int getX()
      {
 return x;
 }
     int getY()
      {
 return y;
 }
};
int main()
{
Point p1(10, 15); // Normal constructor is called here
Point p2 = p1; // Copy constructor is called here
cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();
return 0;
}
```

Destructor:

Destructor is a member function which destructs or deletes an object.

A destructor function is called automatically when the object goes out of scope:

(1) the function ends

(2) the program ends

(3) a block containing local variables ends

(4) a delete operator is called

```
Ex:#include<iostream>
     #include<conio.h>
     using namespace std;
     class BaseClass
{
public:
   BaseClass()
 {
     cout << "Constructor of the BaseClass : Object Created"<<endl;
   }
   ~BaseClass()
{
     cout << "Destructor of the BaseClass : Object Destroyed"<<endl;
   }
};
int main ()
{
     BaseClass des;
     return 0;
}
```