27/04/22

Date
Page 01

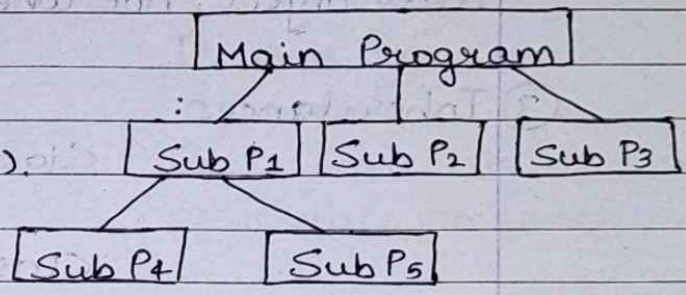
OOP

Object oriented programming

- OOP was coined by Alan Kay in 1996 or 1997.
- C++ has total 5 updates, latest one in 2020 (C++20).
- Six characteristics:
  1. Class
  2. Object
  3. Inheritance
  4. Data Abstraction
  5. Data Encapsulation
  6. Polymorphism.
  7. Overridding and overloading

POP

Procedure oriented programming

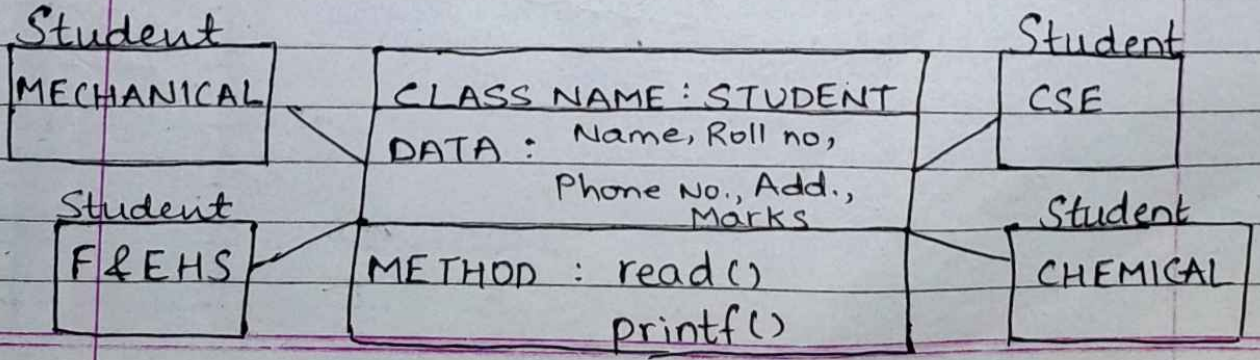


Disadvantages :-

1. Lower data security, as all sub programs use global data.
2. Cannot extend program easily.
3. POP follows top-down approach and focuses on procedure rather than data.

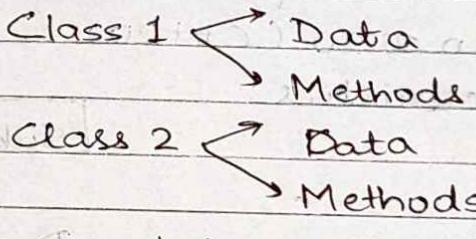
(1) Class : It is a blueprint or plan that defines data and methods used by object.

(2) Object : It is an instance of class that combines data and methods.



- We can create any number of classes and objects.
- Memory allocation happens when we create an ~~class~~ object, not when we create a class.

③ Inheritance :



If we want to use data in Class 2 that is present in Class 1 but not in Class 2, we use inheritance.

Here,

Class 1 : parent class or base class

Class 2 : child class or derived class.

④ Data Abstraction : Hides the complexity of the code and shows the simplest terms to the user. (Summary).

Q. Differentiate between OOP and POP.

	OOP	POP
1.	It is a bottom-up approach.	It is a top-down approach.
2.	It is concentrated on data rather than on functions.	It is concentrated on procedure or functions.
3.	Data is used by associated class and functions within.	In POP, data is used globally.
4.	It is more secure because external functions cant use another data.	It is less secure as data is global.
5.	Ease of modification.	Modification is difficult.
6.	Eg) C++, Java, Python, PHP etc.	Eg) C, BASIC, COBOL, FORTRAN etc.

04/'22

C	C++
#include<stdio.h>	#include <iostream.h> iostream ⇒ input output stream.
C	C++
printf("Hello");	cout << "Hello"; cout ⇒ console out. cin ⇒ console input
C	C++

int a = 10;
printf("a=%d", a);
	int a = 10;
	cout << "a = " << a;
	<< : Insertion opeeator
	>> : Extraction operator

C	C++
int a = 10, b = 20; printf("a = %d, b = %d", a, b);	int a = 10, b = 20; cout << "a = " << a << ", b = " << b;
C	C++
\n	<< end
C	C++

int a;
printf("Enter a:");
scanf("%d", &a);
printf("Value of a is
%d", a);
	int a;
	cout << "Enter a:";
	cin >> a;
	cout << "Value of a is"
	<< a;

29/04/22.

- When we use multiple i/o operations in a single line, it is called cascading.
  eg) cin >>a>>b;

- To find range,
  Range = 2^n ; where n = no. of bits.
  For int : No. of bytes = 2
  ∴ No. of bits = 16
  Range - 2^{16} = 65536 (1 to 65536)
  which is written as 0 to 65535
  Now, 65536/2 = 32768 (0 to 32767)
  ∴ Range : −32767 to +32767

- Two ways to define global variable:

1) # define var-name value;
   var-name in UPPER CASE
   Can be done before main() function.

2) const var-name = value;
   eg: const a = 10;
   where, a : Symbolic constant
   10 : literal
   Can be used inside main function.

* CONDITIONAL STATEMENTS :

① IF STATEMENT :

```
if (condition)
{
   Statements;
}
```

② IF... ELSE STATEMENT :

```
if (condition)
{
   Statement(s);
}
else
{
   Statement(s);
}
```

Q. Write a program to identify whether a person can vote or not.

→

```
#include <iostream.h>
int main()
{
   int a;
   cout << " Enter your age:";
   cin >> a;
   if (a >= 18)
   {
      cout << " Eligible for voting.";
   }
```

```
    else
    {
        cout << "Ineligible for voting.";
    }
    return 0;
}
```

③ IF... ELSE... IF LADDER

```
if (condition 1)
{
    Statement-block 1;
}
else if (condition 2)
{
    Statement-block 2;
}
else
{
    Statement(s);
}
```

Q. Write a program to allot following groups, ba
on age :

Group A : 0 - 15 years
Group B : 16 - 35 years
Group C : 36 - 60 years
Group D : 61 years and older.

→

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

```
int a;
cout << "Enter your age:";
cin >> a;
if (a >= 0 && a <= 15)
{
   cout << " Group A";
}
else if (a >= 16 && a <= 35)
{
   cout << "Group B";
}
else if (a >= 36 && a <= 60)
{
   cout << "Group C";
}
else
{
   cout << "Group D";
}
return 0;
}
```

④ NESTED IF... ELSE

```
if (condition 1)
{
   Statement(s);
   if (condition 2)
   {
      Statement(s);
   }
```

```
    else
    {
      statement(s);
    }
}
else
{
   Statement(s);
}
```

Q. Write a program to check greater among three numbers.

→
```
#include <iostream.h>
int main()
{
   int a,b,c;
   cout << "Enter the three numbers :";
   cin >> a >>b >>c;
   if (a>b)
   {
      if (a>c)
      {
        cout <<" a is greatest";
      }
      else
      {
        cout <<" c is greatest";
      }
   }
   else
```

```
    {
        if (b>c)
        {
            cout << " b is greatest";
        }
        else
        {
            cout << " c is greatest";
        }
    }
    return 0;
}
```

⑤ FOR LOOP

```
for (initialisation ; condition ; inc/dec)
{
    Statements ;
}
```

⑥ WHILE LOOP

```
initialisation
while (condition)
{
    Statement(s); inc/dec;
}
```

⑦ DO.. WHILE LOOP

```
initialization
do
{
```

```
    Statement(s);
    inc/dec;
}
while (condition);
```

- For loop and while loop are entry-controlled loops and do...while loop is exit-controlled loop.

Q. Write for loop to print odd numbers between 1 and 10.

→
```
#include<iostream.h>
int main()
{
   int i;
   for (i=1; i<=10; i=i+2)
   {
      cout<<"\n"<<i;
   }
   return 0;
}
```

04/05/'22 UNIT 2 : CLASS AND OBJECT

Date Page 12

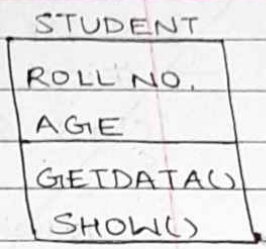
- CLASS : uses encapsulation
  - data
  - function
- Syntax :

```
keyword ← class class_name
          {
Access      private :
specifiers/     data
Access      public :
modifiers       function
          };
```

- The data defined in class is called data member, while function defined in class is called member function.
- ACCESS SPECIFIER
  - Private ↓ Can be used within the class
  - Public ↓ Anywhere in program
  - Protected ↓ Within class and Inherited class.

⇒ Declaring / defining class:
(Before main function)

```
class student
{
    private:
            int roll, age;
    public:
            getdata();
            show();
};  OR  } s1, s2;
```



- When we do not specify access specifier, it is by default set to private.
- Functions can be defined inside class or outside class.

⇒ Declare object (Inside main() function):

```
class-name object-name;
student s1;
student s2;
```

} → `student s1, s2;`

- Class-name and object-name can't be the same.
  `student student;` ✗
  C++ is case sensetive, so,
  `student Student;` ✓

- Memory allocation will happen differently for dat of objects but commonly for functions.

Q. Write a program for given example :

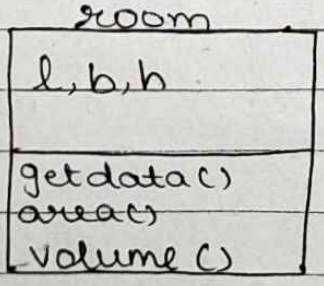
→
```
#include <iostream.h>
class student
{
  private:
        int roll, age;
  public:
    void getdata()
    {
      cout << " Enter roll no., age : ";
      cin >> roll >> age;
    }
    void show()
    {
      cout << " Roll no = " << roll;
      cout << " Age = " << age;
    }
};
int main()
{
  student s1, s2;
  s1.getdata();
  s2.getdata();
  s1.show();
  s2.show();
  return 0;
}
```

- This is example of defining function inside class.

05

⊙ SCOPE RESOLUTION OPERATOR (::)

- Syntax - :: var_name.
- This is used when the name of global variable is same as a local variable.
- ```
#include <iostream.h>
int a = 123;
int main()
{
    int a = 456;
    cout << "Local a = " << a;
    cout << " Global a = " << ::a;
    return 0;
}
```
- This is also used when defining functions outside class.




Write program defining functions inside class and outside class.
```

5/05/22.

⇒
```
#include<iostream.h>
class room
{
  private:
          int l,b,h;
  public:
         void getdata()
         {
          cout<<"Enter l,b and h:";
           cin>>l>>b>>h;
         }
         void area()
         {
           cout<<" Area="<<l*b;
         }
         void volume()
         {
           cout<<"Volume="<<l*b*h;
          }
} R1;
int main()
{
    R1.getdata();
    R1.area();
    R1.volume();
    return 0;
}
```

⇒
```
#include <iostream.h>
class room
{
    float l,b,h;
    public:
            void getdata();
            void area();
             void volume();
}R1;
void room :: getdata()
{
   cout << " Enter l,b,h : ";
   cin >> l >> b >> h;
}
void room :: area()
{
    cout<< "Area = " << l * b;
}
void room :: volume()
{
    cout << " Volume = " << l*b*h;
}
int main()
{
    R1.getdata();
    R1.area();
    R1.volume();
     return 0;
}
```

09/05/22. FUNCTION PROTOTYPING

① Function without return type and without argument

⇒

```
#include<iostream.h>
void sum();
int main()
{
    sum();
    return 0;
}
void sum()
{
    int a,b,c=0;
    cout<<"Enter values of a and b:";
    cin>>a>>b;
    c=a+b;
    cout<<"The sum is:"<<c;
}
```

Q. Create a program to add, subtract, multiply two numbers.

→

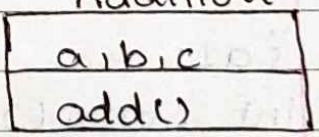
```
#include<iostream.h>
void sum();
void sub();
void mul();
int main()
{
    sum();
    sub();
    mul();
    return 0;
```

```
}
void sum()
{
    int a,b,c=0;
    cout<<"Enter values of a and b:";
    cin>>a>>b;
    c=a+b;
    cout<<" sum="<<c;
}
void sub()
{
    int a,b,c=0;
    cout<<"Enter values of a and b:";
    cin>>a>>b;
    c=a-b;
    cout<<" Difference="<<c;
}
void mul()
{
    int a,b,c=0;
    cout<<"Enter values of a and b:";
    cin<<>>a>>b;
    c=a*b;
    cout<<" Product="<<c;
}
```

Q. Program to add, subtract—X.

Q. Write a program to add two numbers using the concept of class and object.

→
```
#include <iostream.h>
Class addition
{
    int a,b,c=0;
    public:
        void add()
        {
          cout<<"Enter a and b:";
          cin>>a>>b;
          c=a+b;
          cout<<"Addition="<<c;
        }
};A1;
int main()
{
   A1.add();
   return 0;
}
```



13/05/'22.

Revision of Function Prototyping

16/05/'22. Function without return type and without arguments.

→

```
#include<iostream.h>
class add
{
    private:
        int a,b;
    public:
        void sum()
        {
            cout<<"Enter a and b:";
            cin>>a>>b;
            cout<<"The sum is:"<<a+b;
        }
}a1;
int main()
{
    a1.add();
    return 0;
}
```

* Function without return type and with argument.

→

```
#include<iostream.h>
class add
{
    public:
            void add (int, int)
            {
                cout << " Sum = " << a+b;
            }
};a1;
int main()
{
    int a, b;
    cout << " Enter values of a and b";
    cin >> a >> b;
    a1.add (a,b);
    return 0;
}
```

⊙ The compiler, due to function prototyping, comes to know,

1. Name of function
2. Return type of function
3. Number of parameters
4. Type of parameters
5. Sequence of parameters

* Function with return type and with argument:
→

```
#include<iostream.h>
class add
{
    public :
        int sum(int , int)
        {
            return a+b;
        }
}a1;
int main()
{
    int a,b,c;
    cout<<" Enter values of a and b:";
    cin>>a>>b;
    c=a1.add(a,b);
    cout<<"Sum ="<<c;
    return 0;
}
```

* Function with return type and without argument.

→

```
#include<iostream.h>
class add
{
    int a,b,c;
    public:
            int sum()
            {
                ~~c=a+b;~~
                return a+b;
            }
}a1;
int main()
{
    ~~int~~ cout<<"Enter a and b:";
            cin>>a>>b;
            c=a1.sum();
            cout<<"Sum="<<c;
            return 0;
}
```

* Write a program to display age where the access specifier of variable is public.

→
```
#include<iostream.h>
Class A
{
   public:
        int age;
        void printage()
        {
            cout << "Age = " << age;
        }
}a1;
int main()
{
    ~~int~~ age = 18;
    a1.printage();
    return 0;
}
```

⊙ NOTE :- When a variable is declared as private, it cannot be viewed in main() function.

* Manipulators :- ① endl
② setw
③ setfill
④ dec
⑤ oct
⑥ hex

• Used with header file #include <iomanip.h>

① endl → \n
→ used to create a new line.

- cout << "Hello \n";
  cout << "World";
- cout << "Hello";
  cout << "\n World";
- cout << "Hello \n World";
- cout << "Hello" << endl;
  cout << "World";
- cout << "Hello";
  cout << endl << "World";
- cout << "Hello" << endl << "World

Output :- Hello
World.

② Setw → set width

cout << "__Hello";
cout << "\t Hello";
cout << setw(7) << "Hello";

Output :- ___Hello

cout << setw(3) << "Hello";
will give the same answer :- does not change data.

③ setfill (Specifies symbol, character or digit to write in space left by setw).

cout << setw(6) << setfill('$') << "Neha";
Output :- $$Neha.

④ int n=11;
cout << "hex=" << hex << n;

20/05/'22.

Date
Page 28

Task :- Application of reference variable.

- Syntax to create reference variable -
  datatype &ref_variable = actual_variable

int a=5; a is actual variable and y
int &y = a; is reference variable.
Any change in value of a will also be made to value of y.

⇒
```
#include <iostream.h>
int main()
{int a = 5;
 int &y = a;
 cout << "a = " << a;
 cout << "y = " << y;
 return 0;
}
```

Output : a=5 y=5

* DEFAULT ARGUMENT :

Normal function :-

```
void sum (int a, int b) //Declaration
sum(a,b); // Calling.
```

Function with default argument :-

```
void sum( int a=5, int b=10); //Declaration
sum( ); // Calling.
```

When no arguments are passed while calling

these default values are used. If different arguments are passed then they are considered

```
void sum(int a, int b)
{
        Func. definition,
}
```

Don't have to specify values while defining the function.

⇒ If we are giving default value to a variable, then we also have to give a default value to the variable to its right.

```
#include<iostream.h>
#include<iomanip.h>
void sum(int a=10, int b=20);
int main()
{
   int a=1, b=3;
   sum();
   sum(a,b);
   sum(a);
   sum(b);
   return 0;
}
void sum(int a, int b)
{
   cout<<"a+b="<<a+b<<endl;
}
```

Output :-

a+b = 30
a+b = 4
a+b = 21
a+b = 23

⇒ sum(, b) will give an error because arg of right variable is not being passed.

THEORY QUESTIONS

(MID - SEMESTER)

Q1. Write down the difference between C and C++.

C	C++
• C is a procedure / function oriented language.	• C++ language is driven by a procedure or object.
• Data is not protected in C	• In C++ data is secured
• C uses a top-down approach. The program is prepared step by step.	• C++ uses bottom-up approach. Base elements are prepared first.
• In C we cannot give the same name to two functions.	• In C++ we can give the same name to two function due to function overloading feature
• C uses printf() and scanf() functions to write and read respectively	• C++ uses cout and cin objects for output and input operations. Further, cout uses insertion operator (<<) and cin uses (>>) extraction operator.
• C uses stdio.h file for input and output functions.	• C++ uses iostream.h for cout and cin functions.
• Constructors and destructors are absent in C.	• They are present in C++.
• No inline functions	• Inline functions are supported

Q2. Differentiate between OOP and POP.
→ Page 03

Q3. Enlist the features of OOP and explain any three.
→ The features of OOP are:
1) Object
2) Class
3) Inheritance
4) Data Abstraction
5) Data Encapsulation
6) Polymorphism
7) Overriding and Overloading.

(i) CLASS - A class in C++ is a user-defined type or data structure declared using keyword class that contains data and functions.
- A class can also be defined as a blueprint or plan that defines data and methods used by object.
- The data defined in class is called data member and the function defined in class is called member function.
- The access of data and functions can be controlled using the access modifiers private, public and protected. The default access modifier is private.
- Syntax for declaring and defining class:

```
class class_name
{
    private:
```

data members
public:
member functions
};

- eg)

STUDENT
DATA : Rollno. Marks
FUNCTION : getdata() showdata()

```
class STUDENT
{
  private:
     int Rollno;
     float marks;
  public:
     void getdata();
     void showdata();
};
```

(ii) OBJECT - An object is an instance of class that combines data and functions.

- Memory allocation does not take place when a class is defined but when an object is declared.
- Memory allocation will happen differently for data but common for functions.
- An object can be declared in the following two way
  (i) At the end of class
  (ii) Inside main function.
- An object and class cannot have the same name

eg)

STUDENT
DATA: roll no.
FUNCTION:
getdata()
showdata()

⇒ At end of class

```
class STUDENT
{
  pub private:
        int rollno;
    public :
        void getdata();
        void showdata();
} R1, R2;
```

⇒ Inside main function

```
int main()
{
    STUDENT R1, R2;
    Statement(s);
}
```

(iii) INHERITANCE : Inheritance is used when a class wants access to the data and functions used in another class.

CLASS 1 → Data
Function

CLASS 2 → Data
Function.

If we want to use data and functions of CLASS 1 in class 2, which are not present in CLASS 2, we

use inheritance.

- Here, CLASS 1 is called parent class or base class and CLASS 2 is called child class or derived class.

Q4. Define basic structure of C++ programming and also explain how to declare class and object.

→ The basic structure of C++ programming is -

```
Include statements
Global Data or Functions
Class Declaration
main()
{
  Object creation;
  Statements;
}
```

⊙ A class is a blueprint or plan for an object.

- It can be declared using keyword 'class' followed by the name of the class. The body of the class is defined inside curly brackets and terminated by a semicolon at the end.

eg)
```
class class_name
{
  private:
      data;
  public:
      functions;
};
```

⊙ An object is an instance of class that combines data and functions.

- Memory allocation occurs when object is declared.
- Objects can be declared in two ways -

1) At end of class

```
class class_name
{
    private:
        data;
    public:
        function;
} obj 1, obj 2, ... obj n;
```

2) Inside main function

```
int main()
{
    class_name obj 1, obj 2, .., obj n;
    statement(s);
    return 0;
}
```

Q5. List down the applications of C++.

→

(i) OPERATING SYSTEMS - Microsoft Windows, Mac OSX and Linux - all are programmed in C++ as it is a stronly typed and fast programming language.

(ii) BROWSERS - The rendering engines of various web browsers are programmed in C++ as they require faster execution provided by C++.

(iii) LIBRARIES - Many high-level libraries use C++ as the core programming language in the backend because of its speed.
eg) Tensorflow which is a Machine Learning library

(iv) GRAPHICS - All graphics applications require fast rendering and just like the case of web browsers, C++ helps in reducing latency.

(v) GAMING - PS4s and xBOX

(vi) BANKING SYSTEMS (APPLICATIONS) - They process millions of transactions on a daily basis and require high concurrency and low latency support. One of the most popularly used core-banking system - Infosys Finacle uses C++ as one of the backend programming languages.

(vii) DATABASES - MySQL and Postgres : two of the most widely used databases are written in C and C++.

Q6. List down any five control statements supported by C++ and explain any two.

→ 1) if... else
2) for loop
3) while loop
4) do...while loop
5) Switch case
6) break
7) goto
8) if statement

(i) IF... ELSE STATEMENT - An if... else statement is used to execute a block of code among two alternatives.

- Syntax :

```
if (condition 1)
{
    block of Statements 1;
}
else
{
    statement block 2;
}
```

- If the condition is true, then statement-block 1 is executed and if the condition is false then statement-block 2 is executed.

eg) To find greater amongst two numbers :

```
#include <stdio.h> <iostream.h>
int main()
{
  int a, b;
  cout << "Enter a and b:";
  cin >> a >> b;
  if (a > b)
  {
    cout << "a is greater";
  }
  else
  {
    cout << "b is greater";
  }
  return 0;
}
```

⊙ Output :

Enter a and b : 4
3
a is greater.

(ii) FOR LOOP - The for loop is used to iterate a part of the program several times, when the number of iterations is fixed.

- Syntax :

```
for(initialisation; condition; increment/decreme
{
    statement(s);
}
```

- The body of the loop is executed while the conditi is true.
- For loop is an entry-controlled loop.

eg) Print numbers from 1 to 10.

```
#include <stdio.h> <iostream.h>
int main()
{
  int i;
  for(i=01; i<=10; i++)
  {
    cout<<i<<" ";
  }
  return 0;
}
```

⊙ Output :

1 2 3 4 5 6 7 8 9 10

Q7. What do you mean by access modifier? How many access modifier are supported by C++? Also explain the importance of private access specifier.

→ Access modifiers define how the members (data and functions) of a class can be accessed.

- They are keywords in object-oriented languages that set the accessibility of members.

⊙ C++ supports three access modifiers -

(i) public : The members of the class can be accessed and modified anywhere outside class.

(ii) private : The members of the class can only be accessed and used inside the class.

(iii) protected : The members of the class can be accessed inside class and inherited class.

⊙ IMPORTANCE OF PRIVATE ACCESS SPECIFIER :

- The default access modifier in C++ is private.
- private access specifier allows a class to hide its member variables and member functions from other functions, classes and objects.
- The private access specifier provides data security.
- It helps in data hiding.

Q8. Explain how to define function outside class with use of example.

OR

Q9. Write down the importance of scope resolution operator in c++ with use of example.

→ The scope resolution operator, denoted by '::' is used for various reasons such as -

1) When the name of a local variable is same as that of a global variable, it is used to call the global variable.
2) Defining a function outside of class.

eg)
```
#include<iostream.h>
class student
{
   private:
        int rollno;
        float marks;
   public:
        void getdata();
        void showdata();
} s1;
void student ::getdata()
{
  cout<<"Enter your roll number and marks:";
  cin>>rollno>>marks;
}
void student ::showdata()
{
```

```
    cout << "Roll Number = " << rollno;
    cout << "Marks = " << marks;
}
int main()
{
    s1.getdata();
    s1.showdata();
    return 0;
}
```

⊙ OUTPUT :

```
Enter your rollnaumber and marks : 4136
48
Roll Number = 4136
Marks = 48
```

Q10. How does function prototyping help the compiler ?

→ The function prototypes are used to tell the compiler about -

(i) Return type of function
(ii) Number of arguments of function
(iii) Required data types of function parameter.

- This information helps the compiler cross-check the function signatures before calling it.
- If function prototypes are not mentioned then the program is compiled with some warnings and sometimes generates some

strange output.

Q11. List down all the manipulators supported by C++ and explain endl, setw and setfill.

→ The manipulators supported by C++ are:

1) endl
2) setw
3) setfill
4) hex
5) dec
6) oct

NOTE: These can be used with header file -
#include<iomanip.h>

- They change formatting, not data.

(i) endl

- This manipulator is used to create a new line.
- It does the same work as '\n'.

eg) cout << "My roll number is" << endl << "4136";

Output:

My roll number is
4136.

eg) cout << "Roll Number = 4136" << endl;
cout << "Marks = 48";

Output:

Roll Number = 4136
Marks = 48

(ii) setw

- It stands for set width.
- It specifies the width of the output.
- Syntax: setw(width)

eg) cout << setw(7) << "Hello";

Output :

__Hello

eg) cout << setw(3) << "Hello";

Output :

Hello

Here, "Hello" is displayed even though its width is 5 as the data remains unchanged.

(iii) Setfill

- This manipulator specifies the character, symbol or digit to be displayed instead of space created by setw.

eg) cout << setw(7) << setfill('$') << "Hello";

Output :

$$Hello

Q12. Explain the applications of reference variable.

→

(i) MODIFY THE PASSED PARAMETERS IN A FUNCTION : If a function recieves a reference to a variable, it can modify the value of the variable.

(ii) AVOIDING A COPY OF LARGE STRUCTURES : If a large object is passed without reference, a new copy of it is created which causes wastage of CPU time and memory. We use reference to avoid this.

(iii) IN LOOPS TO MODIFY OBJECTS : We can use references in loops to modify elements and avoid the copy of elements everytime the body iterates.

(iv) ALIAS : A reference variable provides an alias (alternative name) for a previously defined variable.

Q13. Explain the concept of default argument in detail.

→ A function can be declared with default arguments.

- When no arguments are passed during function calling the default values are used.
- If different arguments are passed during function calling, then they are used.
- When default arguments are used, we dont have to specify the values while defining the function.
- If we are giving a default value to a variable then we also have to give a default value to the variable on its right.

eg)

```
#include <iostream.h>
#include <iomanip.h>
void sum(int a=10, int b=20);
int main()
{
   int a=1, b=3;
   sum();
   sum(a,b);
```

```
    sum(a);
    sum(b);
    return 0;
}
void sum(int a, int b)
{
    cout << "a+b=" << a+b << endl;
}
```

⊙ Output :

```
a+b = 30
a+b = 4
a+b = 21
a+b = 23
```

NOTE : sum(,b) will give an error because here default value of a is used but b is not used and b is to the right of a. So to pass default argument of a, default argument of b should be passed.

Q14. What is typecasting and explain implicit type-casting in detail with example.

→ Type casting refers to the conversion of one data type into another inside the program.

- Type casting is of two types -

(i) IMPLICIT TYPE CONVERSION

- Also known as automatic type conversion

- Done by the compiler on its own without any instructions by the user.
- Generally takes place in an expression where more than one data type is present.
- All data types of the variable are upgraded to the data type of the variable with largest data type.

bool → char → short int → int → unsigned int → long → unsigned → long long → float → double → long double.

eg)
```
#include<iostream.h>
int main()
{
    int x = 10;
    char y = a;
    x = x + y;
    cout << " x = " << x << endl;
    cout << " y = " << y;
    return 0;
}
```

⊙ Output :

x = 107

y = a

(ii) EXPLICIT TYPE CONVERSION

- It is user-defined
- Can be done by two ways -

1) Converting by assignment
2) Conversion using Cast operator.

Q15. List down the cases in which we cant use inline function.

→

(i) If a function contains a loop.
(ii) If a function contains static variables
(iii) If a function is recursive.
(iv) If a function return type is other than void and the return statement doesn't exist in function body.
(v) If a function contains switch or goto statement.

08/06/'22.

* FUNCTION CALLING METHODS :
  1) Call by value
  2) Call by reference
  3) Call by address

⊙ Call By Value :

```
#include<iostream.h>
void copy (int a);
int main()
{
   int x=12;
   cout<<"Before calling function: \n"<<x;
   copy(x);
   cout<<"After calling function: \n"<<x;
   return 0;
}
void copy(int a)
{
   cout<<"\n The value of a: "<<a;
   a++;
   cout<<"\n The value of a:"<<a;
}
```

⊙ Output :

```
Before calling function : 12
The value of a: 12
The value of a : 13
After calling function : 12
```

⊙ Call By Reference:

```
#include<iostream.h>
void copy (int &a);
int main()
{
    int x=12;
    cout<<" Before calling function :"<<x;
    copy(x);
    cout<<"After calling function :"<<x;
    return 0;
}
void copy(int &a)
{
    cout<<" \n Value of a :"<<a;
    a++;
    cout<<"\n Value of a :"<<a;
}
```

⊙ Output :

```
Before calling function : 12
Value of a : 12
Value of a : 13
After calling function : 13
```

⊙ Call By Address :

```
#include<iostream.h>
void copy (int *a);
int main()
{
   int x=12;
   cout<<"Before calling function :"<<x;
   copy(&x);
   cout<<"\n After calling function :"<<x;
   return 0;
}
void copy (int *a)
{
   cout<<"\nValue of a :"<<a;
    a++;
    cout<<"\n Value of a :"<<a;
}
```

⊙ Output :

```
Before calling function : 12
Value of a : 12
Value of a : 13
After calling function : 13
```

* Difference Between Call By Address and Call By Reference :

Call By Address is a way of calling a function in which the address of the actual arguments is copied to the formal parameters.

- Memory is allocated for both actual arguments and formal arguments.

Call By Reference is a way of calling a function by passing arguments to a function by copying a reference to the argument into the formal parameter.

- Memory is allocated only for actual arguments and formal arguments share that memory.

(How??) => One is just address Value stays same.

Actual parameters : function calling

Formal parameters : function definition

=> In call by address both actual and formal parameters indirectly share the same variable.

- pointers are used as formal arguments

```
void swap(int *a);
main()
{
  swap(&x);
}
void swap(int *a)
{
   // function def.
}
```

```
void swap(int&a);
main()
{
swap(x);
}
void swap(int &a)
{
  // function def
}
```

* Swap two values using call by value :

```
#include<iostream.h>
#include<iomanip.h>
void swap(int &a, int b);
int main()
{
    int x=2, y=3;
    cout<<"Values before swapping :"<<endl;
    cout<<" x = "<<x<<" y= "<<y<<endl;
    swap(x,y);
    cout<<"Values after swapping :"<<endl;
    cout<<"x = "<<x<<": y= "<<y<<endl;
    return 0;
}
void swap(int a, int b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

⊙ Output :

```
Values before swapping:
x = 2   y = 3
Values after swapping:
x = 3   y = 2
```

* Swap two values using call by reference:

```
#include<iostream.h>
#include<iomanip.h>
void swap(int &a, int &b);
Int main()
{
    int x = 2, y = 3;
    cout<<"Values before swapping:"<<endl;
    cout<<" x = "<<x<<" y = "<<y<<endl;
    swap(x,y);
    cout<<"Values after swapping:"<<endl;
    cout<<" x = "<<x<<" y = "<<y;
    return 0;
}
void swap(int &a,int &b)
{
        int temp;
        temp = a;
        a = b;
        b = temp;
}
```

⊙ Output:

```
Values before swapping:
x = 2  y = 3
Values after swapping:
x = 3  y = 2
```

* Swap two values using call by address

```
#include <iostream.h>
#include <iomanip.h>
void swap(int *a, int *b);
int main()
{
    int x=2, y=3;
    cout<<"Values before swapping:"<<endl;
    cout<<"x = "<<x<<" y= "<<y<<endl;
    swap(&x,&y);
    cout<<"Values after swapping:"<<endl;
    cout<<"x = "<<x<<" y = "<<y<<endl;
    return 0;
}
void swap(int * a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

⊙ Output:

```
Values before swapping:
x=2 y=3
Values after swapping:
x=3 y=2
```

* ARRAY WITHIN THE CLASS

```
#include<iostream.h>
class SOT
{
   int roll[5];
   public:
          void getdata()
          {
              for(int i=0;i<5;i++)
              {
                 cin>>roll[i];
              }
          }
          void display()
          {
              for(int i=0;i<5;i++)
              {
                 cout<<"Roll Number = "<<roll[i]<<
                       "\n";
              }
          }
};CSE;
int main()
{
   CSE.getdata();
   CSE.display();
   return 0;
}
```

⊙ Output :

110
111
112
113
114
Roll Number = 110
Roll Number = 111
Roll Number = 112
Roll Number = 113
Roll Number = 114

* ARRAY OF OBJECT

```
#include<iostream.h>
class SOT
{
    private:
            int roll;
    public:
           void getdata()
           {
              cout<<"Roll Number: ";
              cin >> roll;
           }
           void display()
           {
              cout<<"Roll Number = "<<roll<<"\n";
           }
}CSE[5];
int main()
{
   for(int i=0; i<5; i++)
   {
      CSE[i].getdata();
      CSE[i].display();
   }
    return 0;
}
```

Date
Page 5

⊙ Output:

Roll Number : 110
Roll Number = 110
Roll Number : 111
Roll Number = 111
Roll Number : 112
Roll Number = 112
Roll Number : 113
Roll Number = 113
Roll Number : 114
Roll Number = 114

08/06/'22.

* FUNCTION OVERLOADING

- This happens when two or more functions in a single program have the same name and same return type.
- These functions can be differentiated by
  1) number of arguments
  2) argument's name
  3) data type of arguments.

eg) A program can have the following functions:

```
void Copy ();
void Copy (int a);
void Copy (float a);
void Copy (int a, int b);
```

=>
```
#include<iostream.h>
void sum();
void sum(int, int);
Int main()
{
    int x = 12, y = 13;
    Sum();
    Sum(x,y);
    return 0;
}
void sum()
{
    int a = 2, b = 3;
    Cout << " Sum = " << a+b;
```

```
}
void sum(int a, int b)
{
    cout<<"\nSum = "<<a+b;
}
```

⊙ Output:

Sum = 5

Sum = 25