PRACTICAL 1 : % by CLASS & object

```
#include<iostream.h>
#include<conio.h>
class student
{
 int roll_no;
 char name[20];
 char class_st[8];
 int marks[5];
 float percentage;
 float calculate();
 public:
 void readmarks();
 void displaymarks();
};
float student::calculate()
{
 percentage=0;
 for(int i=0;i<5;i++)
 percentage+=marks[i];
 percentage=(percentage/5);
 return percentage;
}
void student::readmarks()
{
 cout<<"Enter the roll no.:";
 cin>>roll_no;
 cout<<"Enter the name:";
 cin>>name;
 cout<<"Enter the class studing in:";
 cin>>class_st;
 cout<<"Enter the marks:"<<endl;
 for(int j=0;j<5;j++){
 cout<<"\tEnter mark "<<j+1<<":";
 cin>>marks[j];
 }
}
void student::displaymarks()
{
 cout<<"Roll no:"<<roll_no<<endl;
 cout<<"Name:"<<name<<endl;
 cout<<"Class:"<<class_st<<endl;
 cout<<"Percentage:"<<calculate()<<endl;
}
int main()
{
 student s1;
 s1.readmarks();
 s1.displaymarks();
 return 0;
}
```

PRACTICAL 2: area of cube

```
#include<iostream>
using namespace std;
int main()
{
 float area, side;
 cout<<"This program is for Surface Area of Cube\n";
 cout<<"Enter the length of the side of the cube\n";
 cin>>side;
 area = 6*side*side;
 cout<<"The Area of the cube with side "<<side<<" is = "<<area<<" sq
units";
 return 0;
}
```

PRACTICAL 3 : employee gross salary

```
#include<iostream>
using namespace std;
int main (){
int salary,gross,hra,da;
cout<<"enter the basic salary of the employee."<<endl;
cin>>salary;
if(salary<= 10000){
da=salary*20/100;
hra=salary*80/100;
gross=salary+da+hra;
cout<<"the gross salary of the employee is"<<endl<<gross;
}
if(salary<= 20000){
da=salary*25/100;
hra=salary*90/100;
gross=salary+da+hra;
cout<<"the gross salary of employee is"<<endl<<gross;
}
else if(salary>20000){
da=salary*30/100;
hra=salary*95/100;
gross=salary+da+hra;
cout<<"the gross salary of employee is"<<endl<<gross;
}
else{
cout<<"you have no salary"<<endl;
}
}
```

PRACTICAL 4 : Student detail in Array

```
#include <iostream>
using namespace std;
#define MAX 10
class student{
 private:
 char name[30];
 int rollNo, total;
 float perc;
 public:
 void getDetails(void);
 void putDetails(void);
};
void student::getDetails(void){
 cout << "Enter name: " ;
 cin >> name;
 cout << "Enter roll number: ";
 cin >> rollNo;
 cout << "Enter total marks outof 500: ";
 cin >> total;

 perc=(float)total/500*100;
}
void student::putDetails(void){
 cout << "Student details:\n";
 cout << "Name:"<< name << ",Roll Number:" << rollNo << ",Total:" <<
total
<< ",Percentage:" << perc;
}
int main(){
 student std[MAX];
 int n,loop;
 cout << "Enter total number of students: ";



cin >> n;
 for(loop=0;loop< n; loop++){
 cout << "Enter details of student " << loop+1 << ":\n";
 std[loop].getDetails();
 }
 cout << endl;
 for(loop=0;loop< n; loop++){
 cout << "Details of student " << (loop+1) << ":\n";
 std[loop].putDetails();
 }
 return 0;
}
```

PRACTICAL 5 : area of circle by constructor

```
#include <iostream>
#define PI 3.141
using namespace std;
class AreaCircle
{
 private:
 float area;
 public:
 AreaCircle(float radius)
 {
 area = PI * radius * radius;
 }
 void display()
 {
 cout << "Area of circle:\t" << area << endl;
 }
};
int main()
{
 cout << "Enter value of radius:\t";
 int radius;
 cin >> radius;
 AreaCircle area(radius);
 area.display();
 return 0;
}
```

PRACTICAL 6 : multiple inheritance

```
#include<iostream>
using namespace std;
class A
{
 int a = 4;
 int b = 5;
 public:
 int mul()
 {
 int c = a*b;
 return c;
 }
};
class B : private A
{
 public:
 void display()
 {
 int result = mul();
 std::cout <<"Multiplication of a and b is : "<<result<< std::endl;
 }
};
int main()
{
 B b;
 b.display();
 return 0;
}
```

PRACTICAL 7 : multiple inheritance

```
#include <iostream>
using namespace std;
class student_detail
{
 protected:
 int rno, sum = 0, i, marks[5];
 public:
 void detail()
{
 cout << " Enter the Roll No: " << endl;
 cin >> rno;
 cout << " Enter the marks of five subjects " << endl;
 // use for loop
 for (i = 0; i < 5; i++)
 {
 cin >> marks[i];
 }
 for ( i = 0; i < 5; i++)
 {
 // store the sum of five subject
 sum = sum + marks[i];
 }  }
};
class sports_mark
{
 protected:
 int s_mark;
 public: void get_mark()
 {
 cout << "\n Enter the sports mark: ";
 cin >> s_mark;
 }
};
class result: public student_detail, public sports_mark
{
 int tot, avg;
 public:
 void disp ()
 {
 tot = sum + s_mark;
 avg = tot / 6; // total marks of six subject / 6
 cout << " \n \n \t Roll No: " << rno << " \n \t Total: " << tot << endl;
 cout << " \n \t Average Marks: " << avg;
 }
};

int main (){
 result obj;
 obj.detail();
 obj.get_mark();
 obj.disp();
}
```

PRACTICAL 8 : HYBRID INHERITANCE

```
#include<iostream>
using namespace std;
class Person
{
 int id;
 char name[200];
 public:
 void accept_person_details()
 {
 cout<<"\n Enter Id : ";
 cin>>id;
 cout<<"\n Enter Name : ";
 cin>>name;
 }
 void display_person_details()
 {
 cout<<"\n Id : "<<id;
 cout<<"\n Name : "<<name;
 }
};
class Teaching : public Person
{
 char subject_name[100];
 char teacher_name[200];
 public:
 void accept_teacher_details()
 {
 accept_person_details();
 cout<<"\n Enter Subject Name : ";
 cin>>subject_name;
 cout<<"\n Enter Teacher Name : ";
 cin>>teacher_name;
 }
 void display_teacher_details()
 {
 display_person_details();
 cout<<"\n Subject Name : "<<subject_name;
 cout<<"\n Teacher Name : "<<teacher_name;
 }
};
class NonTeaching : public Person
{
 char dept_name[200];

 public:
 void accept_nonteaching_details()
 {
 cout<<"\n Enter Department Name : ";
 cin>>dept_name;
 }
 void display_nonteaching_details()
 {
 cout<<"\n Department Name : "<<dept_name;
 }
};
class Instructor : public NonTeaching, public Teaching
{
 public:
 void accept_instructor_details()
```

```
 {
 accept_teacher_details();
 accept_nonteaching_details();
 }
 void display_instructor_details()
 {
 display_teacher_details();
 display_nonteaching_details();
 }
};
int main(){
 Instructor *inst;
 int cnt, i;
 cout<<"\n Enter No. of Instructor Details You Want? : ";
 cin>>cnt;
 inst=new Instructor[cnt];
 for(i=0; i<cnt; i++)
 {
 inst[i].accept_instructor_details();
 }
 for(i=0; i<cnt; i++)
 {
 inst[i].display_instructor_details();
 }
 return 0;
}
```

PRACTICAL 9: Dynamic Binding

```
#include < iostream >
using namespace std;
int Square(int x) //Square is =x*x;
{
 return x * x;
}
int Cube(int x) // Cube is =x*x*x;
{
 return x * x * x;
}
main() {
 int x ;
 cout<< "Enter the value to sqaure or cube:" ;
 cin>>x;
 int choice;
 do {
 cout << "Enter 0 for Square value, 1 for Cube value :\n";
 cin >> choice;
 }
 while (choice < 0 || choice > 1);
 int( * ptr)(int);
 switch (choice) {
 case 0:
 ptr = Square;
 break;
 case 1:
 ptr = Cube;
 break;
 }
 cout << "The result is :" << ptr(x);
 return 0;
}
```

PRACTICAL 10 : increament decreament of value using POINTER

```
#include <iostream>
#include<conio.h>
using namespace std;
int main() {
 int a;
 int *pt;
 cout << "Pointer Example C++ Program : Increment and Decrement
Integer\n";
 cout<<"Enter value :";
 cin>>a;
 pt = &a;
 (*pt)++; //Post Increment
 cout << "\n[a ]:Increment Value of A = " << a;
 ++(*pt); //Pre Increment
 cout << "\n[a ]:Increment Value of A = " << a;
 (*pt)--; //Post Decrement
 cout << "\n[a ]:Decrement Value of A = " << a;
 --(*pt); //Pre Decrement
 cout << "\n[a ]:Decrement Value of A = " << a;
 getch();
 return 0;
}
```

PRACTICAL 11 : DYNAMIC MEMORY MANAGEMENT

```
#include <iostream>
using namespace std;
int main ()
{
 int* m = NULL;
 m = new(nothrow) int;
 if (!m)
 cout<< "allocation of memory failed\n";
 else
 {
 *m=29;
 cout<< "Value of m: " << *m <<endl;
 }
 float *f = new float(75.25);
 cout<< "Value of f: " << *f <<endl;
 // Request block of memory of size
 int size = 5;
 int *arr = new(nothrow) int[size];
 if (!arr)
 cout<< "allocation of memory failed\n";
 else
 {
 for (int i = 0; i< size; i++)
 arr[i] = i+1;
 cout<< "Value store in block of memory: ";
 for (int i = 0; i< size; i++)
 cout<<arr[i] << " ";
 }
 delete m;
 delete f;
 delete[] arr;
 return 0;
}
```

PRACTICAL 12 : VIRTUAL FUNCTION

```
#include <iostream>
using namespace std;
int main ()
{
 int* m = NULL;
 m = new(nothrow) int;
 if (!m)
 cout<< "allocation of memory failed\n";
 else
 {
 *m=29;
 cout<< "Value of m: " << *m <<endl;
 }
 float *f = new float(75.25);
 cout<< "Value of f: " << *f <<endl;
 // Request block of memory of size
 int size = 5;
 int *arr = new(nothrow) int[size];
 if (!arr)
 cout<< "allocation of memory failed\n";
 else
 {
 for (int i = 0; i< size; i++)
 arr[i] = i+1;
 cout<< "Value store in block of memory: ";
 for (int i = 0; i< size; i++)
 cout<<arr[i] << " ";
 }
 delete m;
 delete f;
 delete[] arr;
 return 0;
}
```

PRACTICAL 13 : PURE VIRTUAL FUNCTION

```
// C++ program to calculate the area of a square and a circle


#include <iostream>
using namespace std;
// Abstract class
class Shape {
 protected:
 float dimension;
 public:
 void getDimension() {
 cin >> dimension;
 }
 // pure virtual Function
 virtual float calculateArea() = 0;
};
// Derived class
class Square : public Shape {
 public:
 float calculateArea() {
 return dimension * dimension;
 }
};
// Derived class
class Circle : public Shape {
 public:
 float calculateArea() {
 return 3.14 * dimension * dimension;
 }
};
int main() {
 Square square;
 Circle circle;
 cout << "Enter the length of the square: ";
 square.getDimension();
 cout << "Area of square: " << square.calculateArea() << endl;
 cout << "\nEnter radius of the circle: ";
 circle.getDimension();
 cout << "Area of circle: " << circle.calculateArea() << endl;
 return 0;
}
```

Output:

14 : FILE HANDLING & STREAM MANIPULATION

```
#include<iostream>
#include<fstream>
using namespace std;
main()
{
int rno,fee;
char name[50];
cout<<"Enter the Roll Number:";
cin>>rno;
cout<<"\nEnter the Name:";
cin>>name;
cout<<"\nEnter the Fee:";
cin>>fee;
ofstream fout("d:/student.doc");
fout<<rno<<"\t"<<name<<"\t"<<fee; //write data to the file student
fout.close();
ifstream fin("d:/student.doc");
fin>>rno>>name>>fee; //read data from the file student
fin.close();
cout<<endl<<rno<<"\t"<<name<<"\t"<<fee;
return 0;
}
```

Output:

PRACTICAL 15

C++ program to demonstrate the use of class templates

```
#include <iostream>
using namespace std;
// Class template
template <class T>
class Number {
 private:
 // Variable of type T
 T num;
 public:
 Number(T n) : num(n) {} // constructor
 T getNum() {
 return num;
 }
};
int main() {
 // create object with int type
 Number<int> numberInt(7);
 // create object with double type
 Number<double> numberDouble(7.7);
 cout << "int Number = " << numberInt.getNum() << endl;
 cout << "double Number = " << numberDouble.getNum() << endl;
 return 0;
}
```

Output: