Lab Sheet 6
Understanding the Concept of Type Conversion and Inheritance

Type Conversion

To convert data from basic type to user defined type and vice versa we cannot rely on built in conversion routines because compiler doesn't know anything about user defined types. We have to provide the conversion routines to be used for type casting. Following methods are used for conversion.

a) To convert from basic type to user defined type, conversion is done by using the constructor function with one argument of basic type as follows

```
MyClass(BasicType var)
{
      //Conversion code
    }
```

b) To convert from user defined type to basic type, conversion is done by overloading the cast operator of basic type as a member function as follows

```
class MyClass
{
      ...
      Public:
         operator BasicType()
         {
           //Conversion code
         }
```

```
};
```

The conversions between objects of different classes can be carried out by using similar methods for conversions between basic types and user-defined types. For more details please refer to any text books.

Inheritance

Inheritance is the concept by which the properties of one entity are made available to another. It allows new classes to be built from older and less specialized classes instead of being rewritten from scratch. The class that inherits properties and functions is called the *subclass* or the *derived class* and the class from which they are inherited is called the *super class* or the *base class* . The derived class inherits all the properties of the base class and can add properties and refinements of its own. The base class remains unchanged.

Example

```
class
person //base
class {

protected:

  int age;

  char name[20];

public:

  void readAge(void)
  {cout<<"Enter Age: ";
   cin>>age;

  }

  void readName(void)
  {
   cout<<"\nEnter
   Name:


   cin>>name;

  }

  void
  printPerInformation(v
  oid)
  {cout<<"Name
   - "<<name;

   cout<<"\nAge -
   "<<age;

  }

};
```

```
//derived class inherits
base class
class
student:public
person {

private:

    int Sno;

    int

percentage; public:

    void readSno(void)
     {  cout<<"Enter Sno.: ";
                  cin>>Sno;
    }
```

```
void
readpercentage(void)
{cout<<"Enter
 percentage: ";
 cin>>percentage;
}

void
printStuInformation(v
oid)
{cout<<"\nName -
 "<<name;
 cout<<"\nAge -
 "<<age;
 cout<<"\nS.no
 -
 "<<Sno<<endl;
 cout<<"Percentage
-
"<<percentage<<endl
;
 cout<<"conclusion"<
 < endl;
 if(percentage>=80)
  cout<<"\
nThe student
is
Outstanding"<
<endl
 else
 if(percentage>=70)
 cout<<"The student is
 Medium"<<endl;
else
```

```
        cout<<"The
        student is
        Poor"<<endl;
    }
};
int main(void)
{
    clrscr();
    student st;
    st.readName();
    st.readAge();
    st.readSno();
    st.readpercentage()
    ;
    st.printStuInforma
    tion( );

    return 0;
}
```

In Above example, person is the base class whereas student is the derived class which inherits all the features of the base class. Multiple classes may be derived from same base class and a derived class can also inherit characteristics of two or more classes.

This pointer

A special pointer called this pointer can be employed in C++ programs. When the object needs to point to itself then this pointer is used. Remember this pointer points (or represents the address of the) object of the class but not the class. Example

```
class student:public person
{
private:
    ......
public:
    ......
    void printAddress(void)
    {
        cout<<"I am from within the object and
        my address is"<<this;
    }
};
```

Exercises

1. Write a program that can convert the Distance (meter, centimeter) to meters measurement in float and vice versa. Make a class distance with two data members, meter and centimeter. You can add function members as per your requirement.

```
#include <iostream>
using namespace
std; class convert
{
    float
m,cm; public:
    convert() {}
    convert(float dm,float dcm)
    {
        m=dm;
        cm=dcm;
    }
    operator float()
    {
        float dis;
        dis=m+cm/100;
        return dis;
    }
    operator int()
    {
        int dis;
        dis=m*100+cm;
        return dis;
    }
};
```

```
int main()
{
    convert distance(12,17);
    float c1;
    int c2;
    c1=distance;
    cout<<"Distance in meter=
    "<<c1<<endl; c2=distance;
    cout<<"Distance in centi-meter=
    "<<c2; return 0;
}

#include<iostream>//or
using namespace std;
class distanc
{
    float meter ,centimeter;
public:
    distanc(float m,float cm)
    {
        meter=m;
        centimeter=cm;
    }
    distanc(float m)
    {
        meter=m;
        centimeter=0;
    }
    operator float()
    {
        return (meter+(centimeter/100));
    }
    void display()
    {
        int m;
        float c;
        m=static_cast<int>(meter);
        c=(meter-m)*100;
        cout<<m<<" meter , "<<c<<" centimeter"<<endl;
    }
};
int main()
{
```

```
     distanc d1(2.1,56.5),d2(.563);
      float disp;
      d2.display();
      disp=d1;
      cout<<"meter = "<<d1;
  }
```

2.Write two classes to store distances in meter-centimeter and feet-inch system respectively. Write conversions functions so that the program can convert objects of both types.

```
#include<iostream>
using namespace std;
class feet
{
    float ft,in;
public:
    feet():ft(0),in(0){}
    feet(float f, float i):ft(f),in(i){}
    float return_ft(){return ft;} float
    return_in(){return in;} void
    showdata()
    {
        cout<<"\n\nIn fps system:
"<<ft<<"ft "<<in<<"in."<<endl;
    }
};
class meter
{
    float m,cm,mt;
```

```
public:
    meter():m(0),cm(0),mt(0){}
    meter(float a, float b):m(a),cm(b){}
    operator float()
    {
        return (m+cm/100)*3.281;
    }
    meter(feet obj)
    {
           mt=(obj.return_ft()+obj.return_in()/12.0)/3.281;
    }
    void showdata()
    {
        cout<<"In metric system: "<<m<<"m
"<<cm<<"cm."<<endl;
    }
    void in_meter()
    {
        cout<<"In meter: "<<mt<<"m";
    }
};
int main()
{
    meter mobj1(10.0,50.0),mobj2;
    feet fobj(4.0,2.0); float d;

    d=mobj1;
    mobj2=fobj;
    mobj1.showdata();
    cout<<"In Feet: "<<d;
    fobj.showdata();
    mobj2.in_meter();
    return 0;
}
```

```
#include<iostream>//or
using namespace std;
class ft_in
{
    float feet,inch;
public:
    ft_in(float ft, float in)
    {
        feet=ft;
        inch=in;
    }
    ft_in()
    {
        feet=0;
        inch=0;
    }
    float get_feet()
    {
        return feet;
    }
    float get_inch()
    {
        return inch;
    }
    void display()
    {
        cout<<feet<<" feet & "<<inch<<" inch"<<endl;
    }
};
class m_cm
{
    float meter,centimeter;
public:
    m_cm(float m, float cm)
    {
        meter=m;
        centimeter=cm;
    }
    m_cm()
    {
        meter=0;
        centimeter=0;
    }
     operator ft_in()
    {
```

```
        float f,in,t,m;
        m=meter+(centimeter/100);
        t=m*3.28084;
        f=static_cast<int>(t);
        in=(t-f)*12;
        return ft_in(f,in);
    }
    m_cm(ft_in fin)
    {
        float m,cm,t,f;
        f=fin.get_feet()+(fin.get_inch()/12);
        t=0.3048*f;
        meter=static_cast<int>(t);
        centimeter=(t-meter)*100;
    }
    void display()
    {
        cout<<meter<<" meter & "<<centimeter<<"
centimeter"<<endl;
    }
};
int main()
{
    m_cm a1(3,50),a2;
    ft_in b1(4,9),b2;
    b2=a1;
    a2=b1;
    b2.display();
    a2.display();
}
```

3. Create a class called Musicians to contain three methods string ( ), wind ( ) and perc ( ). Each of these methods should initialize a string array to contain the following instruments

- veena, guitar, sitar, sarod and mandolin under string ( )
- flute, clarinet saxophone, nadhaswaram and piccolo under wind ( )
- tabla, mridangam, bangos, drums and tambour under perc ( )

It should also display the contents of the arrays that are initialized. Create a derived class called TypeIns

to contain a method called get ( ) and show ( ). The get ( ) method must display a menu as follows
Type of instruments to be displayed a. String instruments
b. Wind instruments
c. Percussion instruments
The show ( ) method should display the relevant detail according to our choice. The base class variables must be accessible only to its derived classes.

```
#include <iostream>
using namespace std;
class musicians
{
 protected:
  string str[5];
  string wind[5];
  string per[5];
 public:
  musicians()
  {
    str[0] ="vern";
    str[1]="guitar";
    str[2]="sitar";
    str[3]="sarod";
    str[4]="mandolin";
    wind[0]="flute";
    wind[1]="mridangam";
    wind[2]="bangos";
   wind[3]="drums";
   wind[4]="tambour";
   per[0]="tabla";
   per[1]="mridangam";
   per[2]="bangos";
   per[3]="drums";
   per[4]="tambour";
 }
 void displaystr()
 {
     for(int i=0;i<5;i++)
```

```
        {
            cout<<str[i]<<endl;
        }
    }
    void displaywind()
    {
        for(int i=0;i<5;i++)
        {
            cout<<wind[i]<<endl;
        }
    }
    void displayper()
    {
        for(int i=0;i<5;i++)
        {
            cout<<per[i]<<endl;
        }
    }
};
class typeins:public musicians
{
public:
    void get()
    {
        char choice;
        cout<<"Enter type of string to display"<<endl;
        cout<<"a. String Instruments"<<endl;
        cout<<"b. Wind Instruments"<<endl;
        cout<<"c. Perc Instruments"<<endl;
        cout<<"Enter your choic: ";
        cin>>choice;
        if (choice=='a')
        {
           displaystr();
        }
        if (choice=='b')
        {
           displaywind();
```

```
        }
        if (choice=='c')
        {
           displayper();
        }
    }
};
int main()
{
    typeins obj;
    char c;
    obj.get();
    return 0;
}

#include<iostream>//or
#include<vector>
using namespace std;
class Musicians
{
protected:
    vector<string> str,win,per;
public:
    void String ()
    {
        str={"veena","guitar","sitar","sarod","mandolin"};
        cout<<"\nstring instruments:"<<endl;
        for (int i=0;i<5;i++)
        {
            cout<<str[i]<<endl;
        }

cout<<"\n*************************************************\
n";
    }
    void Wind()
    {

win={"flute","clarinet","saxophone","nadhaswaram","piccolo
"};
        cout<<"\nwind instruments:"<<endl;
        for (int i=0;i<5;i++)
        {
```

```
            cout<<win[i]<<endl;
        }

cout<<"\n**********************************************\
n";
    }
    void Perc ()
    {

per={"tabla","mridangam","bangos","drums","tambour"};
        cout<<"\npercussion instruments:"<<endl;
        for (int i=0;i<5;i++)
        {
            cout<<per[i]<<endl;
        }

cout<<"\n**********************************************\
n";
    }
};
class TypeIns:public Musicians
{
    char c;
public:
    void get()
    {
        cout<<"Type of instruments to be
displayed:"<<endl;
        cout<<"a. String instruments"<<endl;
        cout<<"b. Wind instruments"<<endl;
        cout<<"c. Percussion instruments"<<endl;
        cout<<"Enter your selection(a/b/c):";
        cin>>c;
    }
    void show()
    {
        switch(c)
        {
        case 'a':
            String();
            break;
        case 'b':
            Wind();
            break;
        case 'c':
```

```
            Perc();
            break;
        }
    }
};
int main()
{
    Musicians m1;
    TypeIns t1;
    m1.String();
    m1.Wind();
    m1.Perc();
    t1.get();
    t1.show();
}
```

4. Write three derived classes inheriting functionality of base class person (should have a member function that asks to enter name and age) and with added unique features of student, and employee, and functionality to assign, change and delete records of student and employee. And make one member function for printing address of the objects of classes (base and derived) using this pointer. Create two objects of base class and derived classes each and print the addresses of individual objects. Using calculator, calculate the address space occupied by each object and verify this with address spaces printed by the program.

```
#include <iostream>
#include <cstring>
using namespace std;
class person
{
 private:
  string name;
  int age;
 public:
  void setname(string n)
  {
    name = n;
  }
  void setage(int a)
  {
    age = a;
  }
  string getname()
  {
    return name;
  }
};
class student: public person
{
 private:
  int rank;
  int clas;
 public:
 int getrank()
  {
    return rank;
  }
  int getclass()
  {
    return clas;
  }
  void setrank(int r)
  {
```

```
    rank = r;
  }
  void setclass(int c)
  {
    clas = c;
  }
};
class employee: public person
{
 private:
  string dep;
  int salary;
 public:
  void setdep(string d)
  {
    dep = d;
  }
  void setsalary(int s)
  {
    salary = s;
  }
  string getdep()
  {
    return dep;
  }
  int getsalary()
  {
    return salary;
  }
};
int main()
{
  person p1, p2;
  employee e1, e2;
  student s1, s2;
  cout <<&p1<<endl<<&p2<<endl;
  cout<< sizeof(p1)<<endl;
  cout<<&p2-&p1<<endl<<endl;
```

```
  cout<<&e1<<endl<<&e2<<endl;
  cout<<sizeof(e1)<<endl;
  cout<<&e2-&e1<<endl<<endl;
  cout<<&s1<<endl<<&s2<<endl;
  cout<<sizeof(s1)<<endl;
  cout<<&s2-&s1<<endl;
  return 0;
}

#include<iostream>//or
using namespace std;
class Person
{
    string name;
    int age;
public:
    void display()
    {
        cout<<"The address of the object in person class
is: "<<this<<endl;
        cout<<"The size of the object in person class is:
"<<sizeof(this)<<endl;
    }
};
class Student:public Person
{
    int roll_no;
    int grade;
public:
    void display()
    {
        cout<<"The address of the object in student class
is: "<<this<<endl;
        cout<<"The size of the object in student class
is: "<<sizeof(this)<<endl;
    }
};
class Employee:public Person
{
    int employee_id;
    float salary;
public:
    void display()
```

```
    {
        cout<<"The address of the object in employee
class is: "<<this<<endl;
        cout<<"The size of the object in employee class
is: "<<sizeof(this)<<endl;
    }
};
int main()
{
    Person pobj1,pobj2;
    Student sobj1,sobj2;
    Employee eobj1,eobj2;
    pobj1.display();
    pobj2.display();
    sobj1.display();
    sobj2.display();
    eobj1.display();
    eobj2.display();
    return 0;
}
```

5. Write base class that ask the user to enter a complex number and make a derived class that adds the complex number of its own with the base. Finally make third class that is friend of derived and calculate the difference of base complex number and its own complex number.

```
#include<iostream
> using namespace
std; class
complex1
{
protected:
    int
    a1,b1;
public:
    complex1()
    {
        cout<<"Enter the real part for first number:
        "; cin>>a1;
```

```
            cout<<"Enter the imaginary part for first
            number:
  ";
            cin>>b1;
        }
  };
  class complex3;
  class complex2:public complex1
  {
      int
  a2,b2;
  public:
      complex2()
    {
        cout<<endl<<"Enter the real part for second
number: ";
        cin>>a2;
        cout<<"Enter the imaginary part for second
number: ";
        cin>>b2;
    }
    void sum()
    {
        a2=a1+a2;
        b2=b1+b2;
        cout<<endl<<"The              sum              is:
"<<a2<<"+"<<b2<<"i"<<endl;
    }
    friend class complex3;
};
class complex3
{
    int a3,b3;
public:
    complex3()
    {
        cout<<endl<<"Enter the real part for third
number: ";
```

```
        cin>>a3;
        cout<<"Enter the imaginary part for third number:
";
        cin>>b3;
    }
    void diff(complex2 obj)
    {
        a3=a3-obj.a1;
        b3=b3-obj.b1;
    }
    void display()
    {
        cout<<endl<<"The          difference          is:
"<<a3<<"+"<<b3<<"i"<<endl;
    }
};
int main()
{
    complex2 obj1;
    complex3 obj2;
    obj1.sum();
    obj2.diff(obj1);
    obj2.display();
    return 0;
}
```