Lab 1 :

```
//Porgram to illustrate input/output stream and manipulators
# include <iostream>
# include <string>
int main(){
   std::string str1;
   std::cout << "Enter a string : " << std::endl;
   std::cin >> str1;
   std::cout << "Output : \n" << str1 << std::endl;

   return 0;
}

/*
Enter a string :
ASCOL
Output :
ASCOL
*/

//Porgram to illustrate dynamic memory with new and delete

# include <iostream>

int main(){
   std::cout << "Enter array size : " << std::endl;
   int sz;
   std::cin >> sz;

   int *arr = new int[sz];
   std::cout << "Enter array elements : " << std::endl;

   for(int i =0; i < sz; ++i){
      std::cin >> arr[i];
   }

   std::cout << "OUTPUT : " << std::endl;
   for(int i =0; i < sz; ++i){
      std::cout << arr[i] << " ";
   }
   std::cout << std::endl;
   delete[] arr;
   return 0;
}

/*
Enter array size :
```

```
5
Enter array elements :
1 2 3 4 5
OUTPUT :
1 2 3 4 5
*/


//Porgram to illustrate function overloading & inline function

# include <iostream>

inline int add(int a, int b){ //inline function
   return a+b;
}

inline int add(int a, int b, int c){ //inline function
   return a+b+c; //overloaded function
}

int main(){
   std::cout << "First add function (2 arguments) : "
   << add(5,8) << std::endl;
   std::cout << "Second add fucntion (3 arguments) : "
   << add(5,8,10) << std::endl;
   return 0;
}

/*
OUTPUT:
First add function (2 arguments) : 13
Second add fucntion (3 arguments) : 23
*/

//Porgram to illustrate default arguments

# include <iostream>
# include <string>

int mul(int a, int m=10){ // multiply given number with 10
   return a * m; //m is default argument
}

int main(){
   std::cout << "OUTPUT: " << std::endl;
   std::cout << "Passing one argument :" << mul(3) << std::endl;
   std::cout << "Passing two argument :" << mul(3,20) << std::endl;
```

```
    return 0;
}

/*
OUTPUT:
Passing one argument :30
Passing two argument :60
*/

// program to illustrate pass by reference and return by reference

# include <iostream>

void test(int & ref){ //take a reference to int
    ref *= 10;
}

int & test2(int & ref){ // takes a reference to int
    ref/=10;
    std::cout << "In Fucntion : " << ref << std::endl;
    return ref; //return a reference to int
}

int main(){
    int a = 20, b = 300;
    int &ref = a; //reference to a
    int &ref2 =b;

    std::cout << "OutPut : " << std::endl;
    std::cout << "Before function call : " << a << std::endl;
    test(ref);
    std::cout << "After function call : " << a << std::endl;

    std::cout << "Before function call : " << b <<std::endl;
    test2(b) = 5000; //retuns a reference and change the value using that reference
    std::cout << "After function call : " << b << std::endl;
    return 0;
}

/*
OutPut :
Before function call : 20
After function call : 200
Before function call : 300
In Fucntion : 30
After function call : 5000*/
```

Lab 2 :

```
// program to illustrate class and object, constructor and destructor

# include <iostream>
# include <initializer_list>

class List{
    private:
        int *list;
        int size;

    public:
        List(int sz):size(sz), list(new int[sz]){}//parameterize constructor

        List(): //default constructor
            List(10){} // create default list of size 10


        List(int sz, std::initializer_list<int> lst){
            size = sz;
            list = new int[size];
            if(sz != lst.size()){
                std::cerr << "Give size and size of provided list doesn't match!" << std::endl;
            }
            else{
                auto b = lst.begin();
                for(int i =0 ; i < size; ++i){
                    list[i] = *(b+i);
                }
            }
        }

        List(const List& other){ //copy constructor
            size = other.size;
            list = new int[size];
            for(int i =0; i<size; ++i){
                list[i] = other.list[i];
            }
        }

        int getsize()const{
            return size;
        }

        void insert(std::initializer_list<int> lst){
            if(size != lst.size()){
                std::cerr << "Give size and size of provided list doesn't match!"
                << std::endl;
            }
            else{
                auto b = lst.begin();
                for(int i =0 ; i < size; ++i){
                    list[i] = *(b+i);
                }
            }
        }

        void insert(unsigned int pos, int val){
            if(pos > size){
                std::cerr << "Out Of Range!" << std::endl;
                return;
            }
            else{
                list[pos] = val;
            }
        }

        void print()const{
            for(int i = 0; i < size; ++i){
                std::cout << list[i] << " ";
            }
            std::cout << std::endl;
        }

        ~List(){
            delete[] list;
        }
};

int main(){
    List l1{5, {1,2,3,4,5}};
    l1.print();

    List l2{5};
    l2.insert({6,7,8,9,10});
    l2.insert(3, 100);
    l2.print();

    List l3 = l2;
    l3.print();

    return 0;
}
```

```
/*
Output:
1 2 3 4 5
6 7 8 100 10
6 7 8 100 10
*/

// Program to illustrate object as a function parameter and
//returning object from a function

# include <iostream>
# include <string>


class Data{
   private:
      std::string name;
      int age;

   public:
      Data(std::string n, int a): name(n), age(a){}
      Data(): Data("",0){}

      Data(const Data& other){
         name = other.name;
         age = other.age;
      }

      void insert(std::string s, int a){
         name = s;
         age = a;
      }

      void print(){
         std::cout << "Name : " << name << std::endl;
         std::cout << "Age : " << age << std::endl;
      }
};

Data getdata(Data d){ //takes objects of class Data as a parameter and returns a object of
                                        //class Data
   std::string n;
   int a;
   std::cout << "Enter name : " << std::endl;
   std::cin >> n;
   std::cout << "Enter age : " << std::endl;
```

```
    std::cin >> a;
    d.insert(n,a);
    return d;
}


int main(){
    Data d1;
    d1 = getdata(d1);
    d1.print();
    return 0;
}

/*
Output:
Enter name :
Testname
Enter age :
23
Name : Testname
Age : 23
*/
```

Lab 4

```
/*Data conversion in cpp :
primitive -> user-define
user-define -> primitive
user-define -> user -define
*/

# include <iostream>
# include <cstdio>

class Time{
   private:
      int hrs, mins, secs;

   public:
      Time() = default;

      Time(int hr, int min, int sec):hrs(hr),
         mins(min), secs(sec){}

      Time(int time){
         //convert int(primitive) data type to user defined type.
         //while creating an object of Time class
         hrs = time/3600;
         mins = (time%3600)/60;
         secs = (time%3600)%60;
      }

      int hours()const{
         return hrs;
      }

      void operator=(int time){
         //convert int(primitive) data type to user defined type.
         //while assigning an int to an obj of class Time
         hrs = time/3600;
         mins = (time%3600)/60;
         secs = (time%3600)%60;
      }

      operator int(){
         //convert user defined data type to int(primitive) date type
         //while creating an int or assigning to an int
         return (hrs*3600 + mins*60 + secs);
      }

      void print(){
            printf("\n Hours : %d \n Minutes : %d \n Seconds : %d\n\n", hrs, mins, secs);
        }


};

class Days{
    private:
        int days;

    public:
        Days() = default;

        Days(int day): days(day) {}

        Days(const Time& time){
            //convert time into day(user-defined type to user-define type)
            //while creating an obj of Days class
            days = time.hours()/24;
        }

        void operator= (const Time& time){
            //convert time day (user-defined type to user-define type)
            //while assigning Time obj to an obj of Days class type
            days = time.hours()/24;
        }

        void operator= (int day){
            //convert int into Days(primitive type to user-define type)
            //while assigning an int to an obj of Days class
            days = day;
        }

        void print(){
            std::cout << " Days : " << days << std::endl;
        }
};

int main(){
    //declear and initilize Time obj
    Time t = {120,45,30};
    t.print();

    //assign new value to Time obj using int
    t = 34000230;
    t.print();
    //declear and initilize Days obj using Time obj
    Days d = t;
    d.print();

    //assign new value to Days obj using int;
    d = 45;
    d.print();
    return 0;
}

/*

 Hours : 120
 Minutes : 45
 Seconds : 30


 Hours : 9444
 Minutes : 30
 Seconds : 30

 Days : 393
 Days : 45
*/
```

Lab 5

```
// Program to illustrate base class and derived class
//public inheritance and constructor in derived class
//member function overloading

# include <iostream>
# include <string>

class Data{ // Base class
   private:
      std::string name;
      int age;

   public:
      Data(std::string n, int a): name(n), age(a){}
      Data(): Data("",0){}

      Data(const Data& other){
         name = other.name;
         age = other.age;
      }

      void insert(std::string s, int a){
         name = s;
         age = a;
      }

      void print(){
         std::cout << "Name : " << name << std::endl;
         std::cout << "Age : " << age << std::endl;
      }
};

class Subdata: public Data{ //Derived class
   private:
      std::string address;

   public:
      Subdata(std::string a): address(a){}
      Subdata():Subdata(" "){}

      void insert(std::string n, std::string ad, int a){
         Data::insert(n,a);
         address = ad;
      }

      void print(){
            Data::print();
            std::cout << "Address : " << address << std::endl;
        }

        void print(char a){
            //member function overloading
            std::cout << "Address : " << address << std::endl;
        }


};

int main(){
    Subdata d;
    d.insert("TestName", "xxxxzzzz", 34);
    d.print();
    std::cout << std::endl;
    d.print('a');

    return 0;
}

/*
Output:
Name : TestName
Age : 34
Address : xxxxzzzz

Address : xxxxzzzz
*/
```

Lab 6:

```
// Program to illustrate friend class and friend function
// static function and this pointer

# include <iostream>
# include <string>

class Data; // class prototype
class Contact; //friend class prototype
void printdata(Data); // friend function prototype

class Data{ // Base class
   private:
      std::string name;
      int age;
      friend class Contact;
      friend void printdata(Data);

   public:
      Data(std::string n, int a): name(n), age(a){}
      Data(): Data("",0){}

      Data(const Data& other){
         name = other.name;
         age = other.age;
      }

      void insert(std::string s, int a){
         name = s;
         age = a;
      }

      void print(){
         std::cout << "Using this pointer : " << std::endl;
         std::cout << "Name : " << this->name << std::endl;
         std::cout << "Age : " << this->age << std::endl;
      }

      static void statprint();
};

void Data::statprint(){
   std::cout << "static Function for class Data" << std::endl;
}
```

```
class Contact{ //friend class
   private:
      std::string address;
      std::string phone;
      Data d;

   public:
      Contact(std::string a,std::string p): address(a), phone(p){}
      Contact():Contact(" ", " "){}

      void insert(std::string ad, std::string ph){
         address = ad;
         phone = ph;
      }
      void insert(std::string n, std::string ad, std::string ph, int a){
         d.name = n;
         d.age = a;
         address = ad;
         phone = ph;
      }

      void print(){
         std::cout << "Address : " << address << std::endl;
         std::cout << "Phone : " << phone << std::endl;
      }

      void print(char f){
         std::cout << "From friend class : " << std::endl;
         d.print();
         print();
      }


};

void printdata(Data d){
   std::cout << "From friend function : " << std::endl;
    std::cout << "Name : " << d.name << std::endl;
   std::cout << "Age : " << d.age << std::endl;
}


int main(){
   Contact d;
   d.insert("TestName", "xxxxzzzz", "8384394394", 34);
   d.print('f');
```

```
    Data dt("TestName2", 55);
    printdata(dt);

    dt.print();
    dt.statprint();

    return 0;
}

/*
Output:
From friend class :
Using this pointer :
Name : TestName
Age : 34
Address : xxxxzzzz
Phone : 8384394394
From friend function :
Name : TestName2
Age : 55
Using this pointer :
Name : TestName2
Age : 55
static Function for class Data
*/


//Program to illustrate abstract class and pure vitual function

#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;
}

/*
Enter the length of the square: 4
Area of square: 16

Enter radius of the circle: 5
Area of circle: 78.5
*/
```

Lab 7:

```
// program to illustrate function template and class template
# include <iostream>

template<typename T> //class template
class Complex{
   private:
      T real;
      T img;

   public:
      Complex(T r, T i):real(r), img(i){}
      Complex():Complex(0,0){}

      Complex(const Complex& other){
         real = other.real;
         img = other.img;
      }

      void print(){
         std::cout << real << "+(" << img <<"i)" << std::endl;
      }
};

template<typename t> //function template
t add(t a, t b){
   return a+b;
}

int main(){
   Complex<int> c(30, 45);
   std::cout << "Class template output : " << std::endl;
   c.print();

   std::cout << "Function template output: " << std::endl;
   std::cout << "add(4,5) : " << add(4,5) << std::endl;
   std::cout << "add(3.455, 8.43) : " << add(3.455, 8.43) << std::endl;
   return 0;
}
```

```
/*
Output:
Class template output :
30+(45i)
Function template output:
add(4,5) : 9
add(3.455, 8.43) : 11.885
*/

//program to illustrate exception handaling

#include <iostream>
using namespace std;

int main(){
   int x = -1;
   // Some code
   cout << "Before try \n";
   try {
      cout << "Inside try \n";
      if (x < 0)
      {
      throw x;
      cout << "After throw (Never executed) \n";
      }
   }
   catch (int x ) {
      cout << "Exception Caught \n";
   }
   cout << "After catch (Will be executed) \n";
   return 0;
}

/*
Output:
Before try
Inside try
Exception Caught
After catch (Will be executed)
*/
```

Lab 8:

```
//program to illustrate ifstream, ofstream, fstream and
// open and close a file

# include <iostream>
# include <fstream>
# include <string>

using namespace std;

int main(){
   ofstream fout;
   string line;
   fout.open("sample.txt");

   cout << "Enter data for text file : " << endl;
   while (fout) {
      getline(cin, line);
      if (line == "-1")
      break;
      fout << line << endl;
   }

   fout.close();

   ifstream fin;
   fin.open("sample.txt");
   cout << "Data read from text file : " << endl;
   while (fin) {
      getline(fin, line);
      cout << line << endl;
   }
   fin.close();

   return 0;
 }

 /*
 Output:
 Enter data for text file :
```

```
This is a sample text file.
-1
Data read from text file :
This is a sample text file.
*/

// program to illustrate file access pointer and manipulator

#include<iostream>
#include<fstream>

using namespace std;

int main()
  {
  fstream fp;
  char hi[100];
  int pos;
  fp.open("text.txt", ios :: out | ios :: ate);

  cout << "\nWriting to a file ... " << endl;
  fp << "This is a one line" << endl;
  fp << "This is a another line\n" << endl;
  pos = fp.tellp();

  cout << "\nCurrent position of put pointer : " << pos << endl;
  fp.seekp(-10, ios :: cur);
  fp << endl << "Writing at a random location ";
  fp.seekp(7, ios :: beg);
  fp << " Hello World ";
  fp.close();

  cout << "\nWriting Complete .... " << endl;
  fp.open("text.txt", ios :: in | ios :: ate);
  cout << "\nReading from the file ... \n" << endl;
  fp.seekg(0);

  while (!fp.eof())
  {
    fp.getline(hi, 100);
    cout << hi << endl;
    }

    pos = fp.tellg();
    cout << "\nCurrent Position of get pointer : " << pos << endl;
    return 0;
}

/*
Output:

Writing to a file ...

Current position of put pointer : 43

Writing Complete ....

Reading from the file ...

This is Hello World his is a anot
Writing at a random location

Current Position of get pointer : -1
*/
```

Object Oriented Programming Lab Report

Amrit Science Campus

Submitted By: Nischal Bikram Bhandari

Symbol No: 23164