Theory :-

Constructor :-

A special method of the class that will be automatically invoked when an instance of the class is created is called as constructor. Following are the most useful features of constructor.

1) Constructor is used for Initializing the value 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 is of Default, Parameterized & copy constructors. The various types of Constructor are as follows :-

Constructors can be classified into 3 types.

1) Default constructor.
2) Parameterized Constructor
3) Copy Constructor.

1. Default Constructor :- Default constructor is also called as Empty constructor which has no arguments and it is automatically called when we create the object of class but. Remember name of constructor is same as name of class and constructor never declared with the help

of void return type. if we never pass or declare any arguments then this is called as the Copy constructors.

2. Parameterized Constructor:- This is another type constructor which has some Arguments and same name as class name but it uses some Arguments so for this, we have to create object of Class by passing some Arguments to the constructor then this will automatically pass the argument to the constructor and the value will retrieve by the Respective Data members of the class.

3. Copy Constructor:- This is also another type of Constructor. In this constructor we pass the object of class into the another object of same class. As name suggests, you copy, means copy the values of one object into the another object of class. This is used for copying the value of class object into an another object of class so we call them as copy constructor and for copying the value we have to pass the name of object whose values we wants to copying and when we are using or passing an object to a constructor then we must have to use the & Ampersand or Address operator.

Page No.
Date

Destructor:- As we know that constructor is that which is used for Assigning some values to data members and for assigning some values this may also used some memory so that to free up the memory which is Allocated by constructor, destructor is used which gets Automatically called at the end of the program and we doesn't have to explicitly call a Destructor and Destructor can't be parametrized or a copy this can be only one means default destructor which have no arguments. For declaring a destructor, we have to use ~ tiled symbol in front of destructor.

Static members:-

A class can contain static members, either data or functions. A static member variable has following properties:-

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- Only one copy of that member is created for the entire class and is shared by all the objects of that class.
- It is the visible only within the class but its lifetime is the entire program. Static data members of class are also known as "class variable", because there is only one unique value for all the objects of that

same class. Their content is not different from one object static members have the same properties as global variables but they enjoy class scope. For that reason, and to avoid them to be declared several times, we can only include the prototype (its declaration) in the class declaration but not its definition outside the class, in the global scope of this class to another. Because it is unique variable value for all the objects of the same class, it can be refferred to as a member of any object of that class or even directly by the class name (of course there is only valid for static members.

A static member functions have following properties:-

- A static function can have access to only other static members (fun or var) declared in the same class.
- A static function can be called using the class name instead of its object name

Class_name :: fun_name;

Static members functions are considered to have class scope. In contrast to non static member functions, these functions have no implicit this argument; therefore, they can use only static data members, enumerators, or nested types directly. Static member functions can be accessed without using an object of the

corresponding class type.
1. They cannot access non static class member data using the member selection operators. (. or ->).
2. They cannot be declared as virtual.
3. They cannot have the same name as a non-static function that has the same arguments types.

E.g. // Static members in classes.

```
Class staticTest {
  private:
  static int x;
  public: static int count()
    { return x; }
  };
int StaticTest :: x=9;
int main()
{
  printf("%d\n", staticTest :: count());
}
```

Output

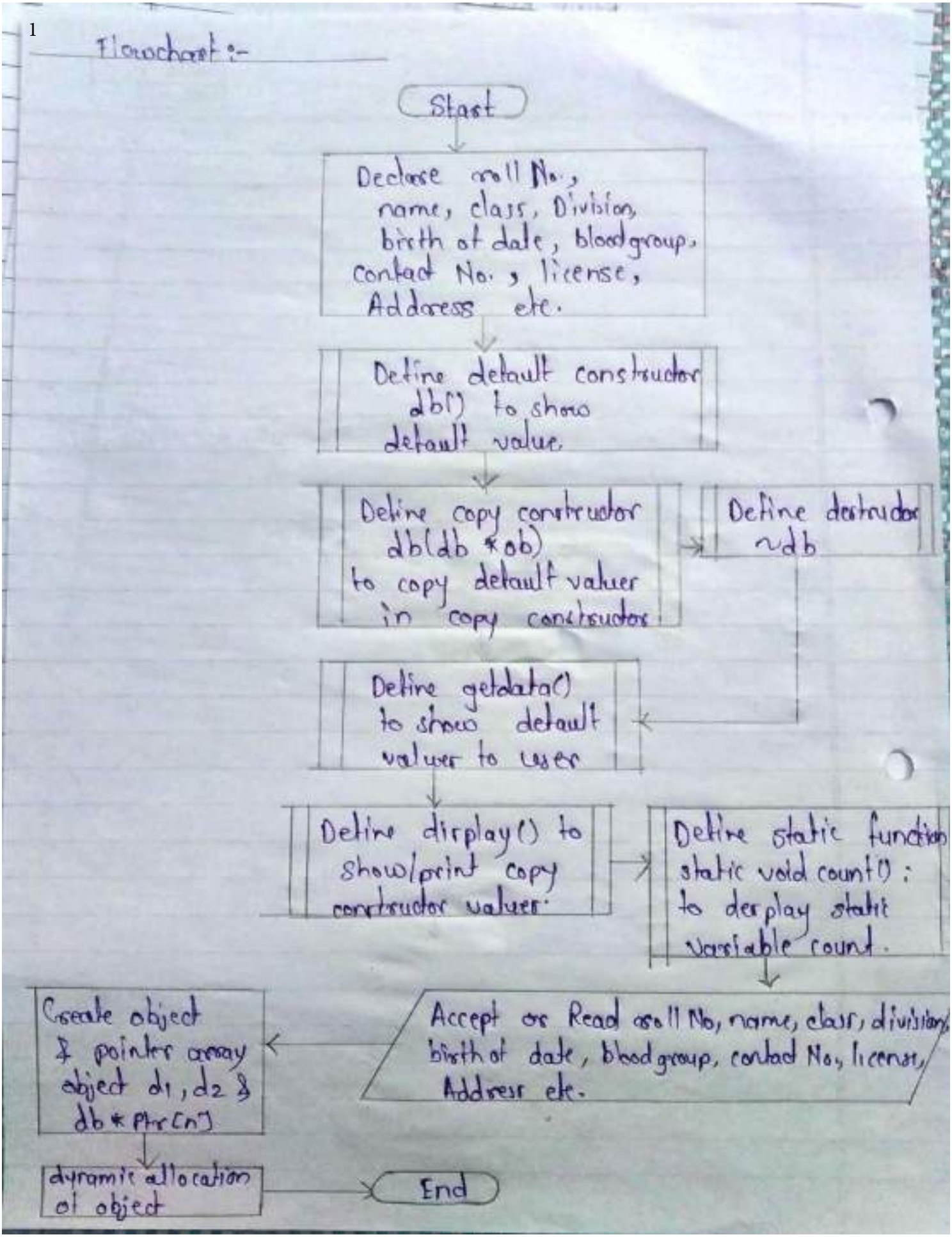
9

Conclusion:-

Hence, we have successfully studied concept of constructor, default constructor, copy constructor, destructor, static member functions.

1

Flowchart :-


Start
Declare roll No., name, class, Division, birth of date, blood group, contact No., license, Address etc.
Define default constructor db() to show default value.
Define copy constructor db(db *ob) to copy default values in copy constructor
Define destructor ~db
Define getdata() to show default values to user
Define display() to show/print copy constructor values.
Define static function static void count() : to display static variable count.
Accept or Read roll No, name, class, division, birth of date, blood group, contact No, license, Address etc.
Create object & pointer array object d1, d2 & db * Ptr[n]
dynamic allocation of object
End

Name :- Sonu Shriram Vishwakarma
Roll No.:- SE63

Page No.
Date

Assignment No. :- 02

Title:- Student Database using constructor, friend class, this pointer, inline code and dynamic memory allocation.

Problem Statement:- Develop an object oriented program in C++ to create a database of student containing the following information: Roll No., Name, Date of Birth, Blood group, Class, Div, Contact address, telephone number, driving license no. etc. Construct the database with suitable member functions for initializing and destroying the data using friend class, this pointer, inline code and dynamic memory allocation operators new and delete as well as exception handling.

Prerequisites:- Object Oriented Programming

Objectives:-

To learn the concept of constructor, default constructor, copy, destructor, Static members functions, friend class, this pointer, inline code and memory allocation operations new and delete.

Theory:-

Friend functions:- In principle, private and protected

members of a class cannot be accessed from outside the same class in which they are declared. However, this rule does not affect friends. Friends are functions or classes declared as such. If we want to declare an external function as a friend of a class, thus allowing this function to have access to the private and protected members of this class, we do it by declaring a prototype of this external function within the class, and preceding it with the keyword friend.

Properties of 'friend function':-

- It is not in the scope of the class to which it has been declared as friend.
- Since it not in the scope of the class, it cannot be called using the object of that class.
- It can be invoked like a normal function w/o the help of any object.
- It can be declared in private or in the public part of the class.
- Unlike member functions, it cannot access the member names directly and has to use an object name and dot operator with each member name.

```
// friend function
#include <iostream>
using namespace std;
```

```
class CRectangle {
    int width, height;
  public:
    void set_values (int, int);
    friend CRectangle duplicate (CRectangle);
};
void CRectangle::set_values (int a, int b)
{
    width = a; height = b; }
CRectangle duplicate (CRectangle rect param)
{
    CRectangle rectres;
    rectres.width = rectparam.width * 2;
    rectres.height = rectparam.height * 2;
    return (rectres);
}
int main()
{ CRectangle rect; rectb;
  rect.set_values (2, 3);
  rectb = duplicate (rect);
  cout << rectb.area();
  return 0;
}
```

The duplicate function is a friend of CRectangle. From within that function we have been able to access the members width and height of different objects of type CRectangle, which are private members. Notice that neither in the declaration of duplicate() nor in its later use in main() have we considered duplicate a member of class CRectangle.

Page No.
Date

Friend classes:-

Just as we have the possibility to define a friend function, we can also define a class as friend of another one, granting that second class access to the protected and private members of the first one.

```
// friend class
# include<iostream>
using namespace std;
class CSquare;
class CRectangle
{
  int width, height;
  public:
   int area()
  {return (width * height);}
  void convert (CSquare a);
  };
  class CSquare{ :
    private:
     int side;
    public:
     void set_side(int a)
    { side=a; }
     friend class CRectangle;
    };
  void CRectangle:: convert (CSquare a){ width=a.side;
       height = a.side; }
  int main()
  { CSquare sqr; CRectangle rect; sqr.set_side(4);
    rect.convert(sqr); cout << rect.area();
    return 0; }
```

In this example, we have declared CRectangle as a friend of CSquare so that CRectangle member functions could have access to the protected and private members of CSquare, more concretely to CSquare::side, which describes the side width of the square.

Pointers:-

A pointer is a derived data type data type that refers to the another data variable by storing the variables memory address rather than data.

Declaration of pointer variable is in the following form:-

```
Data_type *ptr_var;
```

Eg. int *Ptr;

here ptr is a pointer variable and 'points to an integer data type.

We can initialize pointer variable as follows

```
int a, *ptr; // declaration.
ptr = &a  // initialization.
```

pointers to objects:

Consider the following eg.

```
item X; // where item is class and X is
```

object Similarly we can define a pointer it_ptr of type item. as follow item *it_ptr;

Object pointers are useful in creating objects at runtime. We can also access public members of the class using pointers.

Page No.
Date

Eg. item X;
item *ptr = &X;
the pointer 'ptr' is initialized with address of X.
we can access the member functions and data using pointers as follow ptr->getdata();
ptr->show();

this pointer:-
C++ uses a unique keyword called this to represent an object that invokes a member function. this is a pointer that points to the object for which this function was called. This unique pointer is automatically passed to the member functions when it is called.

Important notes on this pointer:-
- this pointer stores the address of the class instance to enable pointer access of the members to the member function of the class.
- this pointer is not counted for calculating the size of the object.
- this pointer are not accessible for static member functions.
- this pointer are not modifiable.

Facilities:-
Linux Operating Systems, C++

Algorithms:-
1) Start
2) Read personnel information such as Name, Date of Birth, Blood group, Height, Weight

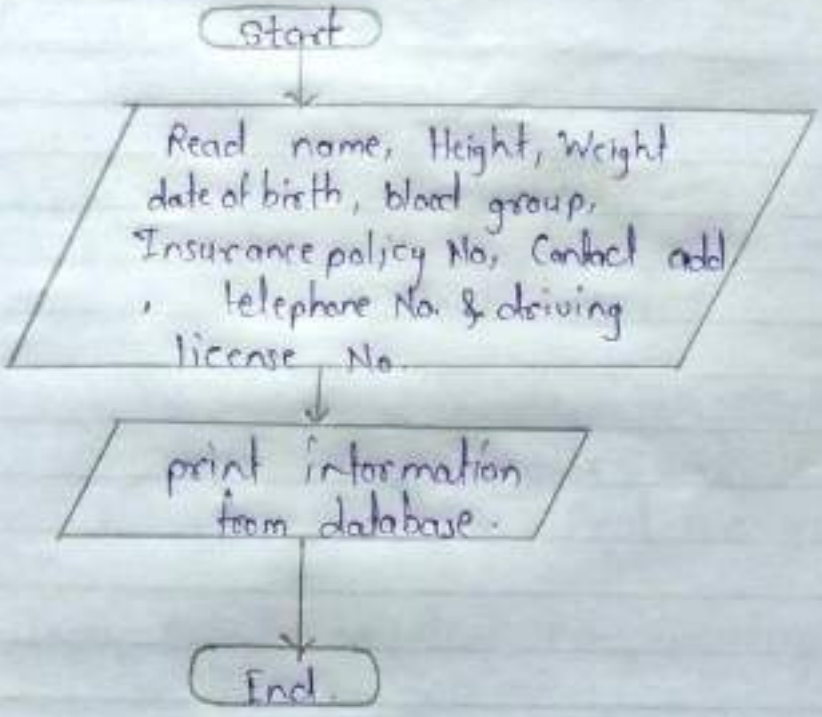
Insurance policy, number, Contact address, telephone number, driving license no.
3) Print all information from database.
4) Stop.

Input:-
Personnel information such as name, Date of Birth, Blood group, Height, Weight, Insurance policy, number, contact address, telephone number, driving license No.

Output:-
Display peonel information from database. The result in following format: Name BOD .... Driving license No
1
2
:
N

Conclusion:-
Hence, we have successfully studied concept of constructor, friend class, this pointer, inline code and dynamic memory allocation operators new & delete.

Flowchart :-


Start
Read name, Height, Weight date of birth, blood group, Insurance policy No, Contact add, telephone No. & driving license No.
print information from database.
End.

Name:- Sonu Shriram Vishwakarma.
Roll No.:- SE63

Page No.
Date

Assignment No. 03

Title / Problem Statement :-

Implement a class complex which represents complex number data type. Implement the following operations.

i) Constructor
ii) Overloaded operator + to add two complex numbers.
iii) Overloaded operator * to multiply two complex numbers.
iv) overloaded << and >> to print and read complex numbers.

Prerequisites :-

C programming language

Objectives :-

To learn the concept of operator overloading.

Theory :-

In C++ the overloading principle applies not only to functions but to operators too. That is of operators can be extended to work not just with built-in types but also classes. A programmer can provide his or her operator to a class by overloading the built-in operator to perform some specific computation when the

operator is used on objects of that class. Is operator overloading really useful in real-world implementations? It certainly can be, making it very easy to write code that feels natural. (we'll see some example soon). On the other hand, operator overloading, like any advanced c++ feature, makes the language more complicated. In addition, operators tend to have very specific meaning, and most programmers don't expect operators to do a lot of work, so overloading operator can be abused to make code unreadable. But we won't do that.

An example of operator Overloading:-

```
Complex a(1.2,1.3); // this class is used to represent complex numbers.
complex b(2.1,3); // Notice the construction taking 2 parameters for the real and imaginary part.
complex c=a+b; // for this to work the application addition operator must be overloaded.
```

The addition without having overloaded operator + could look like this:-

```
complex c= a.Add(b);
```

This piece of code is not as readable as the first example though. we're dealing with numbers so doing addition should be natural.

In order to allow operations like complex c=a+b, in above code, we overloaded the "+" operator.

The overloading syntax is quite simple, similar to function overloading, the keyword operator must be followed by the operator we want to overload:

```
class Complex
{
  public
    complex (double re, double im): real(re), imag(im)
          {};
    Complex operator + (const Complex & other);
    Complex operator = (const Complex & other);
private:
    double real;
    double imag;
};
Complex 'Complex :: operator + (const Complex & other).
{
  double result_real = real + other.real;
  double result_imaginary = imag + other.imag;
  return Complex (result_real, result_imaginary);
}
```

The assignment operator can be overloaded similarly. Notice that we did not have to call any accessor functions in order to get the real & imaginary parts from the parameter other since the overloaded operator is a member of the class and has full access to all private data. Alternatively, we could have defined the addition operator globally and called a member to do the

the actual work. In that case, we'd also have to make the method a friend of the class, or use an accessor method to get at the private data:

```
friend Complex operator+(complex);
Complex operator+(const Complex& num1, const complex
                                    & num2)
{
    double result_real = num1.real + num2.real;
    double result_imaginary = num1.imag + num2.
                                        imag;
    return Complex(result_real, result_imaginary);
}
```

Why would you do this? When the operator is a class member, the first object in the expression must be of that particular type. It's as if you were writing:

```
Complex a(1,2);
Complex a(2,2);
Complex c = a.operator+(b);
```

When it's global function, the implicit or user-defined conversion can allow the operator to act even if the first operand is not exactly of the same type:

```
Complex c = 2+b; // if the integer 2 can be
converted by Complex class, this expression
```

is valid. By the way, the number of operands to a function is fixed; that is a binary operator takes two operands, a unary only one, and you can't change it. The same is true

for the precedence of operators too; for example, the multiplication operator is called before addition. There are some operators that need the first operand to be assignable, such as: operator=, operator(), operator[] overloaded globally. The operator=, operator& and operator, (sequencing) have already defined meanings by default for all objects, but their meanings can be changed by overloading or erased by making them private.

Almost all operators can be overloaded in C++:

+	-	*/	%.	^	&	|	
~	!	,=		=	!=	&&	||
++	--<<		>>	==	&=	|=	*=
+=	-=	/=	%=	^=			
<<=>>=	[]	()	->	->*	new	delete	

The only operators that can't be overloaded are the operators for scope resolution (::) member selection (.) and member selection through a pointer to a function (.*). Overloading assumes you specify a behaviour for an operator that acts on user defined type and it can't be used just with general pointers. The standard behaviour of operators for built-in (primitive) types cannot be changed by overloading, that is you can't overload operator +(int, int).

Page No.
Date

Facilities:-
Linux operating Systems, GCC, codeblocks.

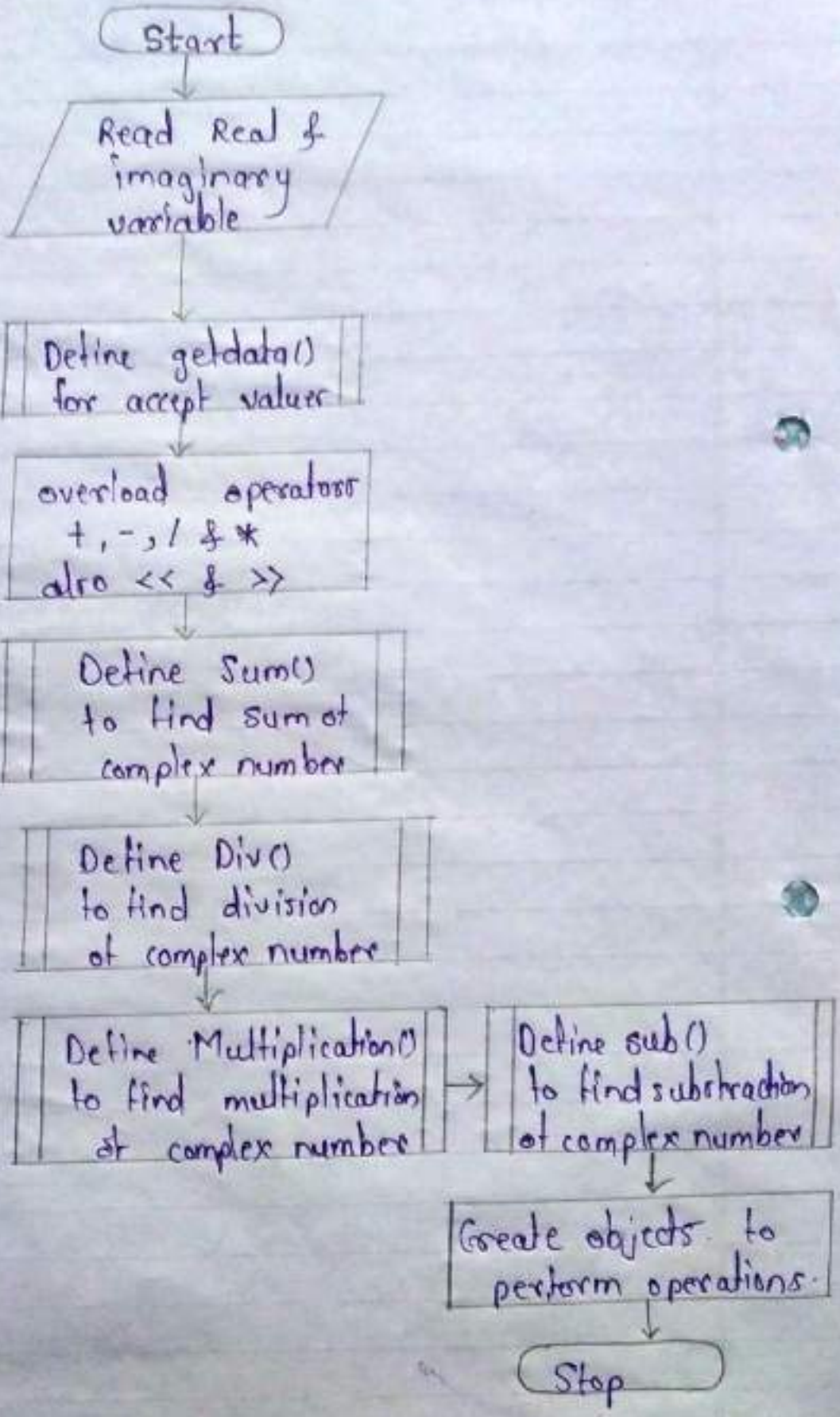
Algorithm:-
1. Start
2. Read all required variables
3. Define function 'getdata() for accepting Values.
4. Overload operators +, -, /, *
5. Define functions Sum, Div, Multiplication etc.
6. Using objects of class a manipulate the values.
7. Stop.

Input:-
2 relational numbers.

Output:-
Addition, subtraction, Multiplication & division of Rational numbers.

Conclusion:-
Hence we have successfully studied concept of operator overloading.

Flowchart :-


Start
Read Real & imaginary variable
Define getdata() for accept values
overload operators +, -, / & * also << & >>
Define Sum() to find sum of complex number
Define Div() to find division of complex number
Define Multiplication() to find multiplication of complex number
Define sub() to find subtraction of complex number
Create objects to perform operations.
Stop

Name:- Sonu Shriram Vishwakarma
Roll No.:- SE63

Page No.
Date

Assignment No: 4

Title:- Creating a class which uses the concept of inheritance, displays data and data members and uses the concept of exception handling.

Problem Statement:- Imagine a publishing company which does marketing for book and audio cassette versions. Create a class publication that stores the title (a string) and price (type float) of publications. From this class derive two classes: book which adds a page count (type int) and tape which adds a playing time in minutes (type float). Write a program that instantiates the book and tape class, allows user to enter data and displays the data members. If an exception is caught, replace all the data members values with zero values.

Prerequisites:-

Object oriented programming

Objectives:-

To learn the concept of inheritance and explain handling.

Page No.
Date

Theory:-

Inheritance:-

Inheritance in object oriented programming can be described as a process of creating new classes from existing classes. New classes inherit some of the properties and behaviours of the existing classes. An existing class that is "parent" of a new class is called a base class. New class that inherits properties of the base class is called a derived class. Inheritance is a technique of code reuse. It also provides possibility to extend existing classes by creating derived classes.

The basic syntax of inheritance is:
Class DerivedClass: accessSpecifier BaseClass

There are 3 access specifiers:
Namely public, private and protected, public.

This inheritance mode is used mostly. In this protected members of Base class becomes protected members of Derived class and public become public.

Protected:-

In protected mode, the public and protected members of Base class becomes protected members of Derived class.

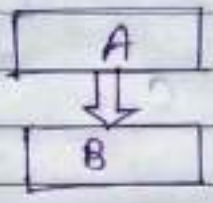
Private:-

In private mode the public and protected

members of Base class become private members of Derived class.

Types of Inheritance:
In C++, we have 5 different types of inheritance Namely,
1. Single Inheritance
2. Multiple inheritance.
3. Hierarchical inheritance.
4. Multilevel inheritance
5. Hybrid inheritance.

Single Inheritance:-



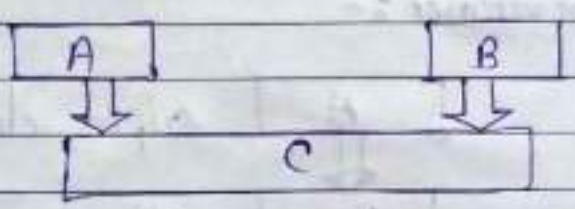
super class
sub class

Syntax:
Class subclass_name:
access_mode base_class
// Single Inheritance.
#include <iostream> using namespace std;
class Vehicle.
{ public:
Vehicle()
{
cout<<"This is a Vehicle"<<endl;
}
};

```
int main()
{
  Car obj;
  return 0;
}
```

Output :-
This is a vehicle

Multiple Inheritance :-
In this type of inheritance a single derived class may inherit from two or more than two base classes.



Syntax :-
class subclass_name : access_mode base_class1, access_mode base_class2 ....
{ // body of subclass };

```
// Multiple inheritance.
# // C++ program to explain
// multiple inheritance.
# include <iostream>
using namespace std;
// first base class
class Vehicle {
public:
  Vehicle()
  { cout << "This is a Vehicle" << endl; }
};
```

```
// Second base class
class FourWheeler {
  public:
    FourWheeler()
    {
    cout << " This is a 4 wheeler vehicle" << endl;
    }
    };
// Sub class derived from two base classes
class Car : public Vehicle, public FourWheeler {
 };
 // Main function
 int main()
 { // Creating object of sub class will
   // invoke the constructors of base classes
     Car obj;
      return 0;
   }
// Multilevel Inheritance
# include <iostream>
 using namespace std;
class Vehicle
{ public:
  Vehicle()
  { cout << "This is a Vehicle" << endl; }
  };
Class FourWheeler: public Vehicle
  { public: FourWheeler()
    { cout << "Objects with 4 wheels are vehicles"
          << endl; } };
```

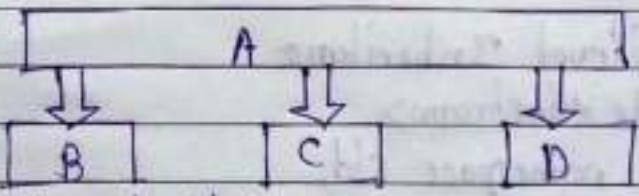
```
class Car: public fourWheeler {
  public:
    car()
  { cout<< "Car has 4 Wheels" <<endl;
   } };
  int main()
  { car obj
    return 0;
    }
```

output:-

This is a Vehicle
Objects with 4 wheels are vehicles.
Cars har 4 wheels.

Hierarchical Inheritance:-

In this type of inheritance, multiple derived classes inherits from a single base class.

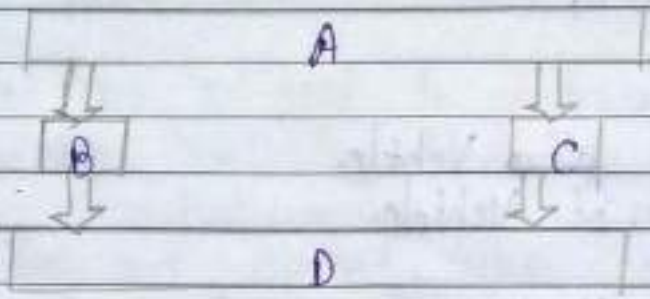



```
// Hierarchical Inheritance
# include <iostream>
  using namespace std;
class Vehicle
{ public: Vehicle()
  { cout<< " This is a Vehicle" <<endl; } };
class Car: public Vehicle
  { };
Class Bus: public Vehicle { };
```

Page No.
Date

Hybrid Inheritance:

Hybrid inheritance is combination of any 2 or more types of inheritance.:




```
// Hybrid Inheritance
#include <iostream>
using namespace std;
class Vehicle
{ public: Vehicle()
  { cout << "This is Vehicle" << endl;
  }
};
class Fare
{ public:
    Fare()
  {
   cout<<" Fare of Vehicle" << endl;
  }
};
class Car: public Vehicle
{
};
class Bus: public Vehicle, public Fare
{
};
```

```
int main()
{
  Bus obj 2;
  return 0;
}
```

Output:-

This is a Vehicle.
face of Vehicle.

Exception Handling:-

Exception handling is part of C++ and object oriented programming. they are added in C++ to handle the unwanted situations during exception handling is to identify and report the runtime error in the program. famous examples are divide by zero, array index out of bound error, file not found, device not found, etc.

C++ exception handling is possible with three keywords. try, catch and throw. Exception handling performs for the following tasks:-

- Find: the problem in the given code. It is also as hit exception.
- It informs error has occured. It is called as throwing the exception.
- We receive the info. It is called as catching the exception.
- It takes the corrective action. It is called as exception handling.

TRY :- It is block code in which there are chances of runtime error. This block is followed by one or more catch block. Most error prone code is added in try Block.

CATCH :- This is used to catch th exception thrown by the try block. In catch block we take corrective action on throwing exception. If files are opened, we can take corrective action like closing file.

```
// Exception Handling.
#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 throw (Never executed)\n";
 }
 }
cout<<" After catch (will be executed)\n";
  return 0;
 }
```

Page No.
Date

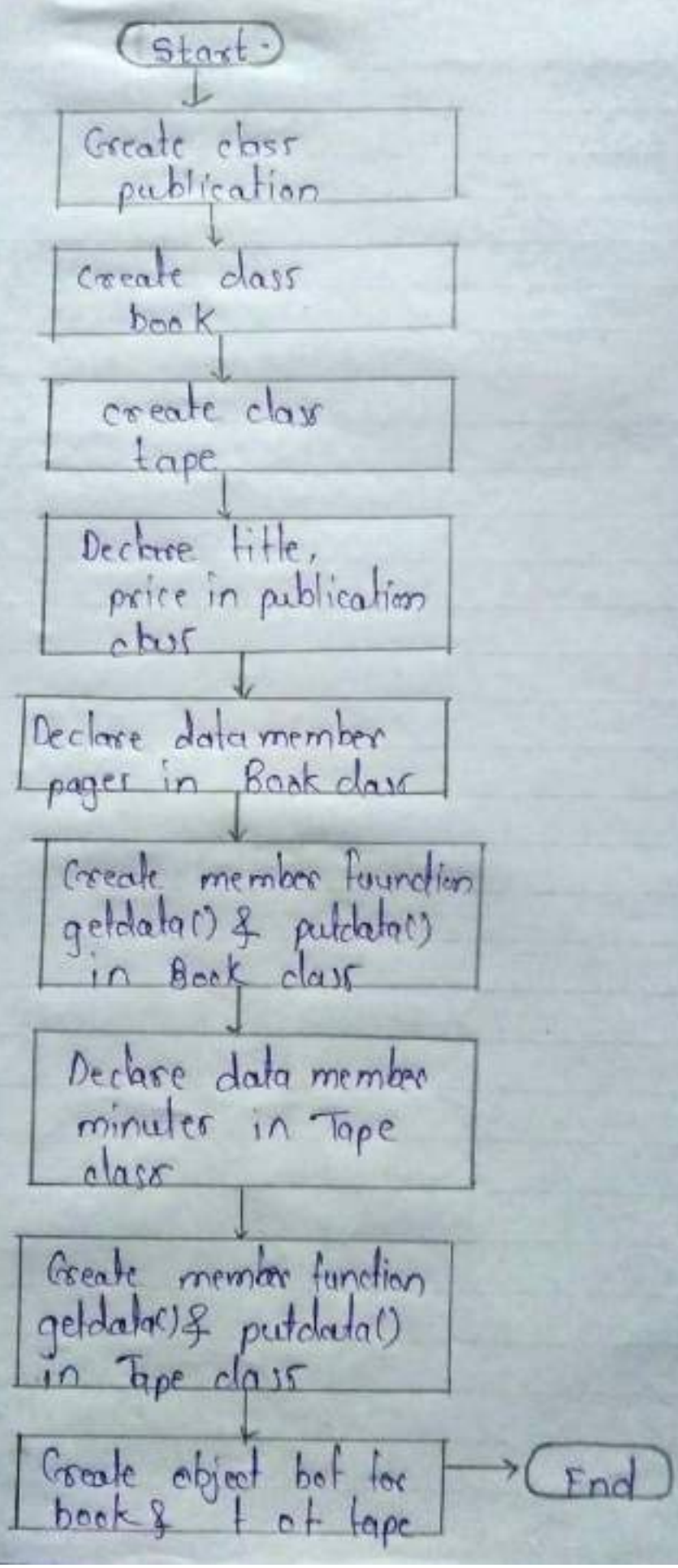
Output:
Before try Inside try
Exception Caught
After catch (will be executed.)

Algorithm:
1) Start
2. Create classes Publication, book & tape.
3. Publication class having data members title, price.
4. Class Book having data members pages & members functions getdata() and putdata().
5. Class Tape having data members, minutes and member functions getdata() and putdata().
6. Create an object bt of class book and object t of class tape
7. Stop.

Display title and price from publication class. The result in following format: Enter Title: OOP
Enter Price: 300
Enter Pages: 250
Enter Title: POP Enter Price: 200
Enter Minutes: 60 Title: OOP
price 300
pages 250 Title: POP Price: 200
Minutes: 60

Conclusion: Hence, we have successfully studied concept of inheritance.

Flowcharts-


Start
Create class publication
Create class book
create class tape
Declare title, price in publication class
Declare data member pages in Book class
Create member function getdata() & putdata() in Book class
Declare data member minutes in Tape class
Create member function getdata() & putdata() in Tape class
Create object bot for book & t of tape
End

Name:- Sanu Shriram Vishwakarma.
Roll No. :- SE63

Page No
Date

Assignment No. :- 5

Title:- Write a function template for selection sort that inputs, sorts and outputs an integer array and a float array.

problem statement:- Write a function template for selection sort that inputs, sorts and outputs an integer array and a float array.

Prerequisites:- Object oriented Programming

Objectives:- To learn the concept of Template.

Theory:-

Templates:-

Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.

A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept. There is single definition of each container, such as vector, but we can define many different kinds of vectors for example, vector<int> or vector<string>.

You can use templates to define functions as well as classes, let us see how do they work.

Function Template:-

The general form of template function is shown here:

```
template <classtype> ret-type func-name(parameter list)
{
  // body of function
}
```

Here, type is a placeholder name for a data type used by the function. This name can be used within the function definition.

The following is the example of a function template that returns the maximum of two values:

```
#include <iostream>
#include <string>
using namespace std;
template <typename T>
inline T const & Max (T const & a , T const & b)
{
 return a < b ? b : a;
}
int main()
{ int i = 39 , int j = 20;
 cout << "Max(i,j): " << Max(i,j);
 << endl;
 double f1 = 13.5 ; double f2 = 20.7;
 cout << "Max(f1 , f2);" << Max(f1, f2) << endl;
 string s1 = "Hello";
 string s2 = "World";
 cout << "Max(s1, s2) : " << Max(s1, s2) << endl;
 return 0;
}
```

If we compile and run above code, this would produce the following result: Max(i,j): 99
Max(f1, f2): 20.7
Max(s1, s2): World

Class Templates-

Just as we can define function templates, we can also define class templates. The general form of generic class declaration is shown here:

```
template <class type> class class-name
{
  .
  .
  .
};
```

Here, type is the placeholder type name, which will be specified when a class is instantiated. You can define more than one generic data type by using a comma-separated list.

Following is the example to define class Stack<> and implement generic methods to push and pop the elements from the stack:

```
#include <iostream>
#include <vector>
#include <stdlib>
#include <string>
#include <stdexcept>
using namespace std;
template <class T> class Stack
{
  private:
    vector<T> elems;  // elements
```

```
public:
 void push(T const&); // push element
 void pop(); // pop element
 T top() const(); // return top element bool
                  empty() const.
 { // Return true if empty: return elements.
    empty(); }
  };
 template <class T>
 void Stack <T> :: push (T const& elem)
 {
  // append copy of passed element elems.push_back
     (elem); }
template<class T> void Stack <T>:: pop()
{
 if (elems.empty())
 {
  throw out_of_range ("Stack <>:: pop(): empty stack");
 }
 // remove last element
 elems.pop_back();
 }
 template <class T>
 T Stack<T>:: top() const
 { if (elems.empty())
  { throw out_of_range ("Stack <>:: top() : empty stack");
 }
 // return copy of last element
 return elems.back();
 }
```

Page No.
Date

Selection Sort :-

Selection sort is a sorting algorithm, specifically an in-place comparison sort. It has $O(n^2)$ time complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and it has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

How selection sort works?

Example.

14	33	27	10	35	19	42	44

For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that 10 is the lowest value.

So we replace 14 with 10. After one iteration 10, which happens to be the minimum value in the list appears in the first position of sorted list.

For the second position, where 33 is residing, we start scanning the rest of the list in linear manner.

We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

After two iterations, two least values are positioned at the beginning in the sorted manner. The same process is applied on

Page No.
Date

the rest of the items in the array.
Pictorial depiction of entire sorting process is as follows :-

Facilities :-
Linux operating Systems, G++

Algorithm :-
1) Start.
2) Declare the template parameter T.
3) Define template function for selection sort.
4) In main() Define two arrays, one for integer and another for float. and take a input for both the arrays and call sorting function template to sort the number.
5) Stop.

Input :-
Selection sort Integer Element :-
Enter how many elements you want 5
Enter the integer element 7
5
8
9
3
Float Element
Enter how many elements you want 5
Enter the float element 3.8
9.4
5.5
2.2
6.7

Page No.
Date

Output :-

Sorted list =

3 5 7 8 9

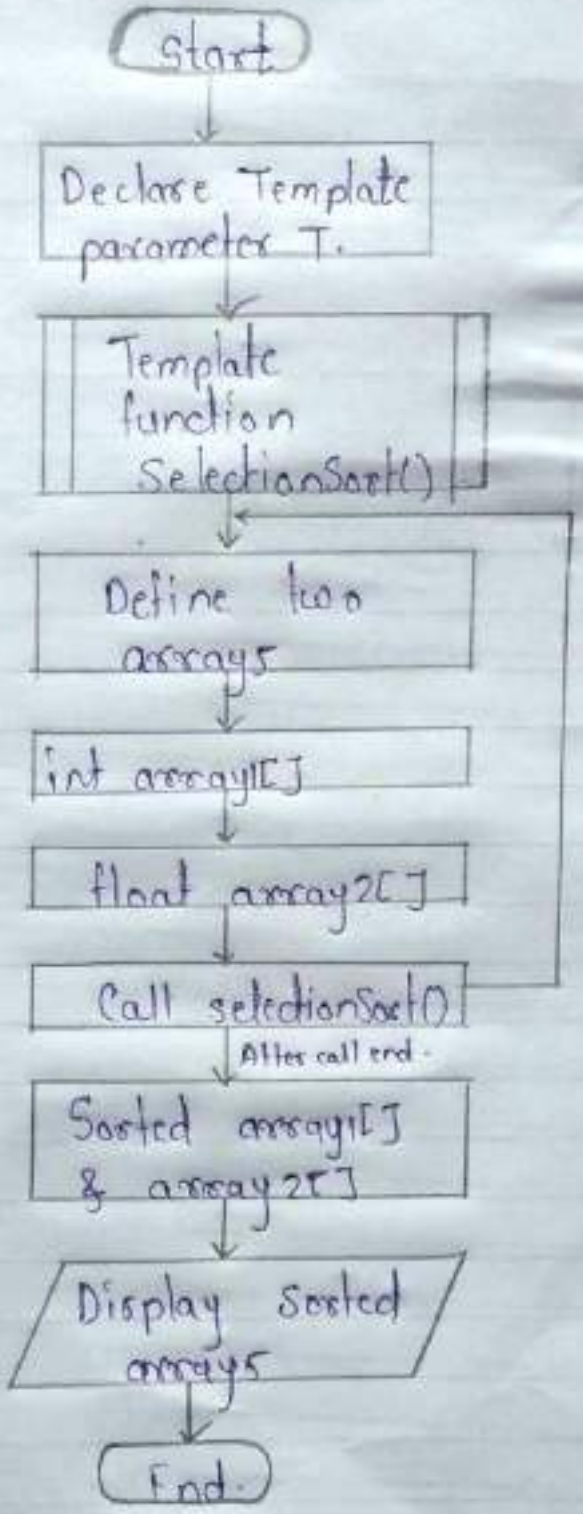
Sorted list =

2.2 3.8 5.5 6.7 9.4

Conclusion :-

Hence, we have studied concept of function Template.

Flowchart:-


Start
Declare Template parameter T.
Template function SelectionSort()
Define two arrays
int array1[]
float array2[]
Call selectionSort()
After call end.
Sorted array1[] & array2[]
Display Sorted arrays
End.

Name :- Sonu Shriram Vishwakarma.
Roll No. :- SE63

Page No.
Date

Assignment No:- 06

Title :- File handling

problem statement :-
Write C++ program that create an output file, writes information to it, closes the file and open it again as an input file and read the information from the file.

Prerequisites :- Object Oriented programming

Objectives :- To learn the concept of File handling.

Theory :-

Stream :-
A stream is a sequence of bytes. It acts as source from which the input data can be obtained or as a destination to which the output can be sent.

1. Input Stream :-
Input stream are used to hold input from a data producer, such as a keyboard, a file, or a network. The source stream that provides data to the program is called the input stream. A program extracts the bytes from the input stream. In most cases the standard input device is the

keyboard. With the cin and "extraction" operator (>>) it is possible to read input from the keyboard.

2. OutputStreams:-

Output Streams are use to hold output for the particular data consumer, such as a monitor, a file or a pointer. The destination stream that receives data from the program is called the output stream. A program insert the byte into an output stream. By default, the standard output of a program points at the screen. So with the cout operator and the "insertion" operator (<<) you can print a message onto the screen. iostream standard library provides cin and cout methods for reading from standard input & writing to standard output respectively.

File handling provides three new datatypes:-

Data type	Description
ofstream	This data type represents the output file stream and is used to create files and to write information to files.
ifstream	This data type represents the input file stream and is used to read information from files.
fstream	This data type represent the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read informations from files.

To perform file processing in C++, header file <iostream> and <fstream> must be included in your C++ source file.

Opening a File.

- A file must be opened before you can read it or write to it.
- Either the ofstream or fstream object may be used to open a file for writing and if stream object is used to open a file for reading purpose only.
- Following is the standard syntax for open() function which is a member of fstream, ifstream and ofstream objects.

void open(const char * filename, ios::openmode mode);

Mode Flag	Description
ios::app	Append mode. In this All output to that file to be appended to the end.
ios::ate	Open a file for output and move the read/write control to the end of the file.
ios::in	Open a file for reading.
ios::out	Open file for writing.
ios::trunk	If the file already exists, its contents will be truncated before opening the file.

- Here, the first argument specifies the name and location of the file to be opened and the

second argument of the open() member function defines the mode in which the file should be opened.

- You can combine two or more of these values by ORing them together.
- For example, if you want to open a file in write mode and want to truncate it in case it already exists, following will be the syntax:-

```
fstream afile;
afile.open("file.dat", ios::out | ios::in);
```

Similar way, you can open a file for reading and writing purpose as follows:-

Closing a File:-

- When a C++ program terminates it automatically closes flushes all the streams, release all the allocated memory and close all the opened files.
- It is always a good practice that a programmer should close all the opened files before programming.
- following is the standard syntax of closing a file close() function, which is a member of fstream, ifstream, and ofstream objects.

```
void close();
```

Writing to a File:-

- While doing C++ programming, you write information to a file from your program using

Page No.
Date

the stream insertion operator (<<) just as you use that operator to output information to the screen.
- The only difference is that you use an ofstream or fstream object instead of the cout object.

Reading from a file:-
- You read information from a file into your program using the stream extraction operator (>>) just as you use that operator to input information from the keyboard.
- The only difference is that you use an ifstream or fstream object instead of an the cin object.

Example:-
```
file.read ((char*)&V , sizeof(V));
file.write ((char*)&V; sizeof(V));
```
- These function take two arguments. The first is the address of the variable V, and the second is the length of the variable in bytes. The address of the variable must be cast to type char* (i.e. pointer to character type).

Facilities:- Linux operating System, Gtt.

Algorithm:-
1) Start
2) Create class
3) Define data members roll number and name.

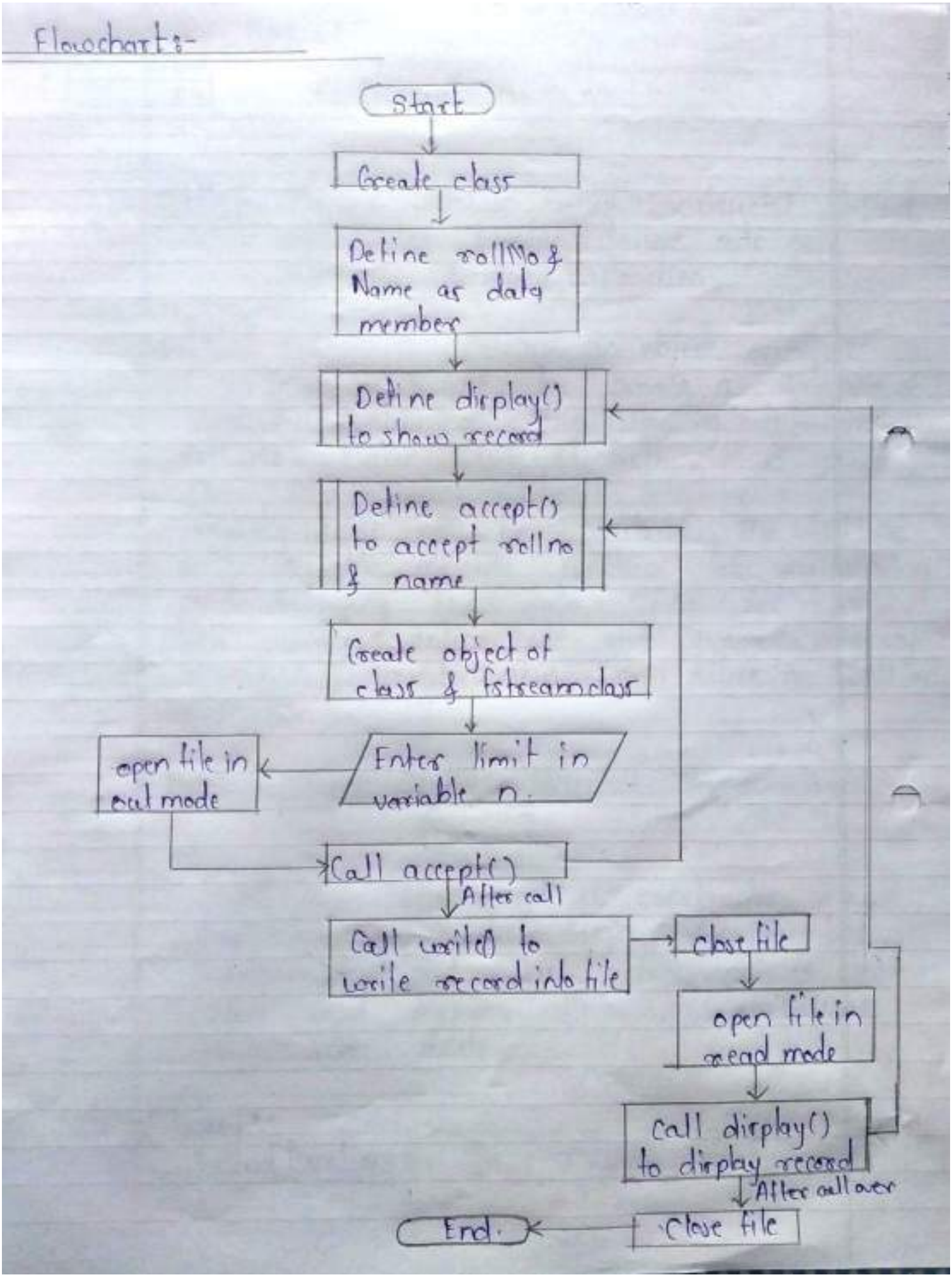
4) Define accept() to take name and roll number from user.
5) Define display() to display the record.
6) In main() create the object of class and fstream class.
7) Take a limit from user in n variable.
8) Open the file in out mode, call accept() to take record from user, then call write() to write that record into the file and at the end close that file.
9) Open the file in in mode, read the record from the file, call display() function to display the record and at the end close that file.
10) Stop.

Input :-
How many record you want 3
1 abc
2 pqr
3 xyz

Output:-
name = abc , Roll = 1 , name = pqr , Roll = 2
name = xyz , Roll = 3

Conclusion:-
Hence, we have studied concept of file handling.

Flowchart:-


Start
Create class
Define rollNo & Name as data members
Define display() to show record
Define accept() to accept rollno & name
Create object of class & fstream class
Enter limit in variable n.
open file in out mode
Call accept()
After call
Call write() to write record into file
close file
open file in read mode
Call display() to display record
After call over
close file
End.


sample - Notepad
File Edit Format View Help
Sonu Vishwakarma 63
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

Name:- Sonu Shriram Vishwakarma
Roll No.:- SE69

Page No.
Date

Assignment No.:- 07

Title:- Item records storing system using sorting and searching for STL and vector containers.

Problem statement:-
Write C++ program using STL for sorting and searching user defined records such as item records(Item code, name, cost, quality, quantity etc) using vector container.

Prerequisites:- Object Oriented Programming

Objectives:- To learn the concept of STL searching, sorting and vector container.

Theory:-

STL:-

The standard Template library (STL) is a set of C++ templates classes to provide common programming data structures and functions such as lists, stacks, arrays etc. It is a library of container classes, Algorithms, A working knowledge of template classes is a prerequisite for working with STL.
STL has four components:-

- Algorithms
- Containers
- Functions
- Iterators

Page No.
Date

Algorithms:-

- The algorithm defines a collection of functions especially designed to be used on ranges of elements. They act on containers and provide means for various operations for the contents of the containers.
- Algorithms:-
  - Sorting
  - Searching
  - Important STL algorithms
  - Useful Array algorithms
  - Partition operations.
  - Numeric.

Containers:-

- Containers or containers classes store object and data. There are in total seven standard "first-class" containers classes and three containers adaptor classes and only seven header files that provides access to these containers or container adaptors.
- Sequence Containers:- Implement data structures which can be accessed in a sequential manner.
  - Vector
  - list
  - deque
  - arrays
  - forward-list (Introduced in C++ 11).

- Container adaptors :- Provide a different interface for sequential containers.
  - queue.
  - priority_queue
  - stack
- Associative Containers :- Implement sorted data structures that can be quickly searched (O log(n) complexity).
  - set
  - multiset
  - map
  - multimap
- Unordered Associative Containers :- Implement unordered data structures that can be quickly searched.
  - Unordered_set
  - Unordered_multiset
  - Unordered_map
  - Unordered_multimap

Functions :-

- The STL includes classes that overload the function call operator. Instances of such classes are called function objects or functors. Functors allow the working of the associated function to be customized with help of parameters to be passed.

Iterators :-

- As the name suggests, iterators are used

for working upon a sequence of values. They are the major features that allow generality in STL.

Utility library:-

- Defined in header <utility>
- pairs.

Sorting:-

It is one of the most basic functions applied to data. It means arranging the data in a particular fashion. Which can be increasing or decreasing. There is a built in function in C++ STL by the name of sort(). These function internally uses Introsort. In more detail implemented using hybrid of QuickSort, HeapSort and insertionSort. By default, it uses QuickSort is doing unfair partitioning and taking more than N*log N time, It switches to HeapSort and when the array becomes really small, it switches to InsertionSort The prototype for sort is:-

Searching:-

It is a widely used searching algorithms that requires the array to be stored before search is applied. The main idea behind this algorithm is to keep dividing the array in half (divide and conquer). until the element is found, or all the elements are exhausted.

It works by comparing the middle item of the array with the target, if it matches, it returns true.
Otherwise if the middle term is greater than the target, the search is performed in the left sub-array. If the middle term is less than target, the search is performed in the right sub-array.
The prototype for binarySearch is;

binary_Search(startaddress, endaddress, value to find);
startaddress: the address of the first element of the array.
endaddress. the address of the last element of the array, value to find.
the target value which we have to search for.

```
// Searching
# include <algorithm>
# include <iostream>
using namespace std;
void show(int a[], int arraysize)
{ for (int i=0; i<arraysize; ++i)
    cout<<a[i]<<" "; }
Int main()
{ int a[]= {1,5,8,9,6,7,3,4,2,0};
int size = sizeof(a) / sizeof(a[0]); cout<<"\n The
array is: ";
    show(a, asize);
cout<<"\n\nLet's say we want to search for 2 in
    the array";
```

```
cout << "\n So, we first sort the array";
Sort (a, a+size);
cout << "\n The new array after sorting is"; show(a,size);
cout << "\n Now, we do the binary search";
if (binary_search (a,a+10, 2))
cout << "\nElement found in the array";
else
  cout << "\n Element not found in the array";
cout << "\n Now, say we want to, search for
      10 ";
if (binary search (a, a+10, 10))
cout << "\n Element found in the array";
  else
  cout << "\n Element not found in the array";

  return 0;
}
```

Outputs-

The array is : 1 5 8 9 0 6 7 3 4 2 0 .1

Let's say we want to search 2 in the array.

So, we first sort the array

The array after sorting is : 0 1 2 3 4 5 6 7 8 9

Now, we do the binary search

Element found in the array

Now, say we want to search for 10

Element not found in the array. facilities.

Facilities :- Linux operating system, Btt..

Page No.
Date

Algorithm :-

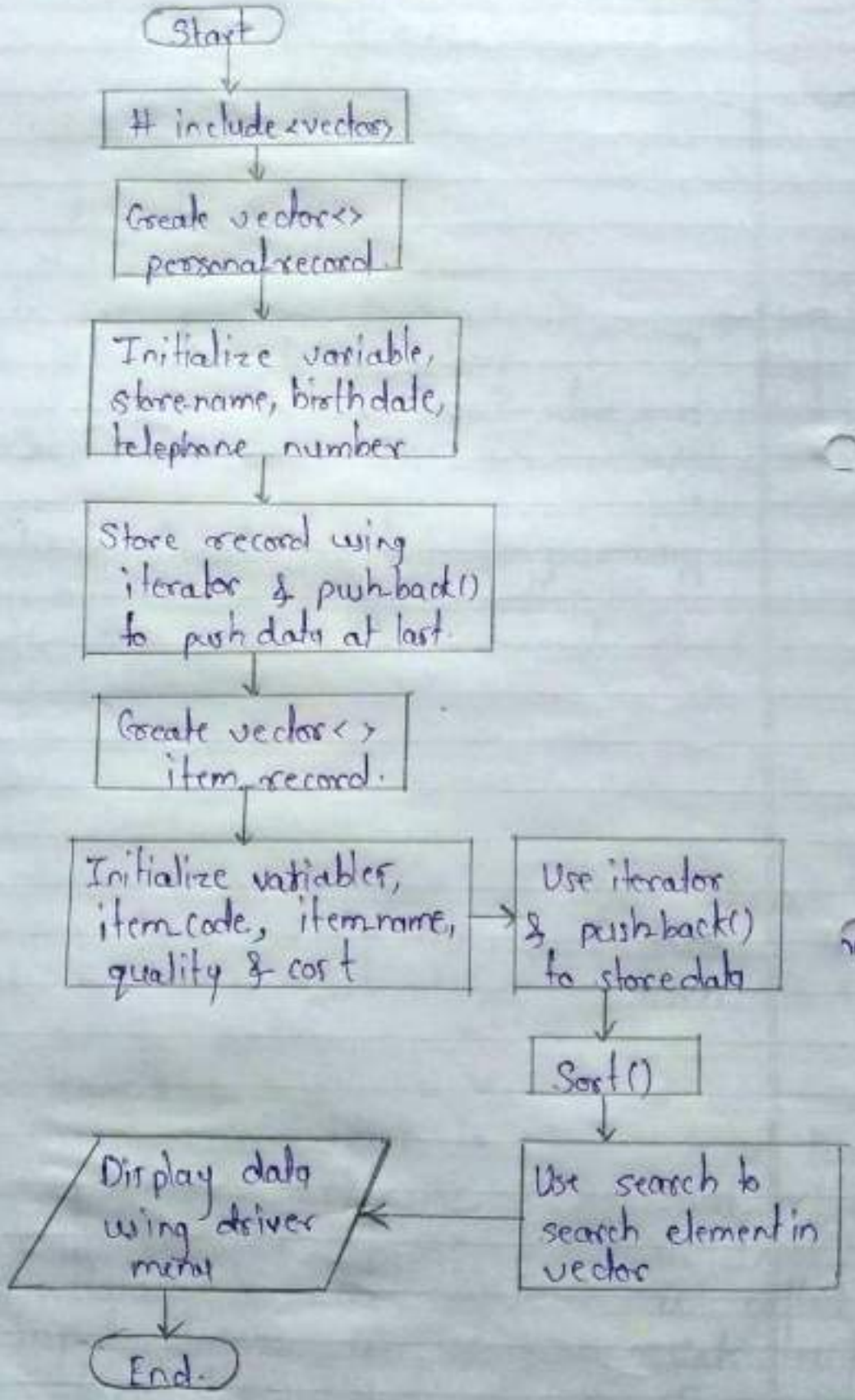
1) Start
2) Give a header file to use 'vector'
3) Create vector naming 'personal_record'.
4) Initialize variable to store name, birth date & telephone number.
5) Using iterators store as many records you want to store using predefined functions as push_back().
6) Create another vector 'item_record'.
7) Initialize variable to store item code, item name, quality and cost.
8) Using iterators and predefined functions store the data.
9) Using predefined function sort(), sort the data stored according to the user requirements.
10) Using predefined function search, search the element from the vector the user want to check.
11) Display and call the functions using menu.
12) End.

Input :-
Personal information such as, name, BOD, telephone number.

Output :- Sorted array (vector)

Conclusion :- Hence, we have successfully studied the concept of STL (Standard template library) and how it makes many data structures easy. It briefs about the predefined functions of STL and their use such as search() and sort().

Flowchart :-


Start
include <vector>
Create vector<> personal record
Initialize variable, store name, birth date, telephone number
Store record using iterator & push back() to push data at last
Create vector< > item record.
Initialize variables, item code, item name, quality & cost
Use iterator & push back() to store data
Sort()
Use search to search element in vector
Display data using driver menu
End.

Name :- Sonu Shriram Vishwakarma.
Roll No. :- SEC3

Page No.
Date

Assignment No. :- 08

Title :- To use map associative containers.

Problem statement :-
Write a program in C++ to use map associative container. the key will be the name of states and values will be the populations of the states. When the programs runs, the user prompted to type the name of a state. The program then looks in the map, using the state name as an index and returns the population of the state.

Prerequisites :- Object Oriented Programming

Objectives :- To learn the concept of map associative container.

Theory :-
Map associative container :-
are associative containers that store elements in a mapped fashion. Each value in map associative with mapped value. No two map values can have same key values.
This operator is the bounds of the size of map, while this operator cause undefined behaviours. used to the at() function th the position is not in.

Page No.
Date

Syntax:-
mapname[key] Parameters:-
key value mapped to the element to be fetched.
Returns:
Direct reference to the element at the given key value.

Examples:-
Input : map mymap;
mymap['a'] = 1;
mymap['a'];
Output : 1
Input : map mymap;
mymap["abcd"] = 7;
mymap["abcd"];
Output : 7

// program
```
#include <map>
#include <iostream>
#include <string>
using namespace std;
int main()
{
// map declaration map<int, string> mymap;
// mapping integer to string mymap[1] = "Hi";
mymap[2] = "This";
mymap[3] = "is";
mymap[4] = "NBN";
// Using operator[] to print string.
```

// mapped to integer 4
cout << mymap['G'];
return 0;
}

Output:- NBN

Facilities:- Linux operating System, Gtt

Algorithm:-
1) Start
2) Give a header file to map associative containers
3) Insert states name so that we get values or population of that same.
4) Use populationMap.insert().
5) Display the population of states.
6) End.

Input:-
Information such as state name to map associative container.

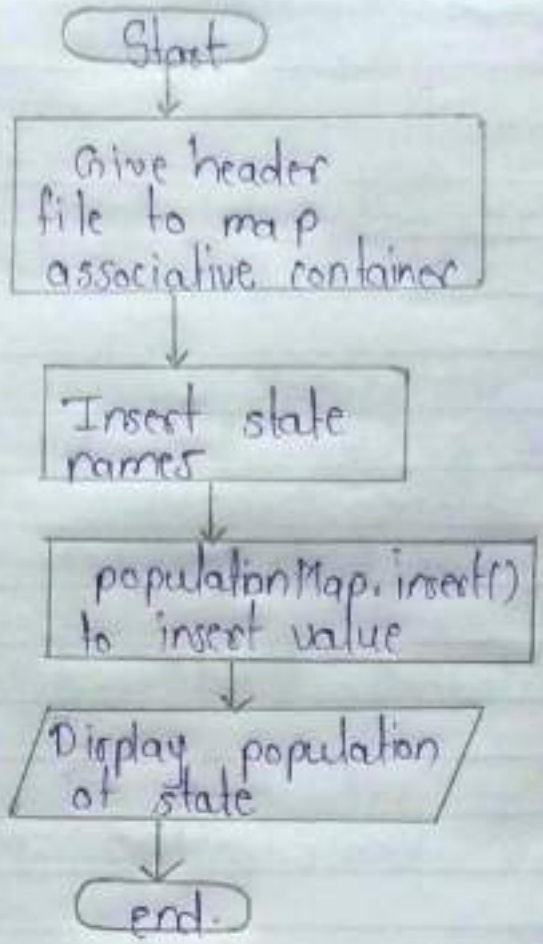
Output:-
Size of population Map: 5 Brasil : 199 milion.
china: 1389 million
India : 1187 million
Indonesia: 234 million
Pakistan : 170 million
Indonesia's populations is 234 million.

Page No.
Date

Conclusion :-

Hence, we have successfully studied the concept of map associative container.

Flowchart :-


Start
Give header file to map associative container
Insert state names
populationMap.insert() to insert value
Display population of state
end.