Assignment - 2

→ Akurathi Sai Vikas
2402200039

(1.) In a list adding element to end

~~SimpleList (int value) {~~.

```
class SimpleList {
    private :
        int arr[100];
        int length = 0;
    public :
        SimpleList (value) {
            if (length < 100) {
                arr[length] = value;
                length ++;
            } → else <<cout<<" list is full";
        };
```

(ii) Removing an element from the list:

In the problem it is Shifting towards right.

after 5 we need to shift the elements left.

```
void. remove (int value){
    int pos = -1;
for (int i=0; i<length; i++) {
    if (arr[i] == value){
        pos = i;
        break;
    }
}
if (pos != -1){
    for (int i= pos; length-1; i++) {
        arr[i] = arr[i+1];
    }
    length --
    cout << "value removed successfully"
    }
else{
    cout << " value not found";
    }
};
```

Question 3 - write a program to demonstrate single inheritance. Create a base class shape and a derived class rectangle. Include methods calculate the area and perimeter. Explain why inheritence is preferred over writing seperate classes

```
#include <iostream>
using namespace std;
class shape{
  public:
    float ~~rectan~~ length;
    float breadth;
  Shape(float len, float bx){
      this -> length = len;
      this -> breadth = bx;
    }
};
class rectangle : public ~~rectangle~~ shape {
  public
    rectangle(float len, float bx) : shape (fiaat len, float bx){
    }
  float area(){
      return length*breadth
  float perimeter{
      return 2*(length+ breadth);
  }
  void display(){
      cout << " Area" << area()<< endl;
      Cout << " perimeter" << perimeter(); << endl;
  }
};
int main()
  rectangle r(5,10)
    r.display();
  }
```

→ here shape gives common features length and breadth.
Rectangle → adds specialized behaviour (area, perimeter)

With inheritence you have the access for length and breadth, Soo you can print any shape by using class Shape

(Ans.) Implement program demonstrating multilevel inheritence. For example, person → employee → manager. Show the data and functions are passed through the inheritence chain. Explain why access specifiers matter here.

```
#include <iostream>
using namespace std;

(1) class person{
    protected:
        string name;
        int age;
    public:
        set_person(string name, int age);
            this -> name = name;
            this -> age = age;
        }.
        void display_person(){
            cout<<"Name"<< name<<" age "<<age;
        }.
};

(2) class employee{
    protected:
        int id;
        float Sal;
    public:
        set_employee( int id; float Sal){
            this -> id = id;
            this -> sal = Sal;
        }.
        void display_employee(){
            cout <<" id" <<id<<"Sal"<<Sal<<endl;
        }
};

(3) class Manager{
    protected;
        string branch;
        float bonus;
    Public
        set_manger(string branch, float bonus){
            this -> branch = branch;
            this -> bonus = bonus;
        }
    void display_manager(){
        display_student();
        display employee;
        cout<<" branch"<<branch<<"Bonus"<<bonus;

int main(){
    manager m1;
    m1.setperson("vikas", 19);
    m1.set_employee(91, 50000);
    m1.set_manger("Datas", 2000);
    m1.display manager();
}
```

In multi level inheritence data functions are passed from one class to another through inheritence chain (person→employee→manager). This helps reuse code and avoid repetition.

Access specifiers control how members are shared. private members are not inherited, protected members are accesible to derived classes and public members are accesible to everywhere. using protected keep data safe but usable in inheretence chain

Q5 Multiple Inheritence and Ambiguity Resolution

Multiple inheritence means one class can inherit from more than one base class.

If both parent classes have the same function name, it creates ambiguity - the compiler gets confused about which function to call.

We can solve this using scope resolution operator (::) by using virtual base classes.

This helps combining features from multiple sources without repeating code.

Example:

```
#include <iostream>, using namespace std;
class person{
public:
    void show(){
        cout<<" person details";
    }
};
class Athlete {
    public
        void show(){
            cout <<" Athlete details";
        }
    };

class Sportsperson: public person, public Athlete {
    public :
        void display(){
            person:: show(); // resolving ambiguity
            Athlete : : Show();
        }
};
int main(){
    Sports person s;
    s.display();
    return 0;
    }
```

(Q6) Friend function for sharing data between classes

A friend function allows two classes to share private data safely. for Example, a student and a sports class may need to calculate total marks.

Friend functions help access both classes private data directly. we use it instead of normal member functions because they belong to neither class but still can access their data.

(Q7) Sorting Algorithms using oop

Sorting Algorithms like bubble sort Algorithm insertion sort Algorithm can be implemented using classes

By using oop, we can keep the data (array) and functions (sorting and display), inside one class.

This makes program more organized modular, and Easier to maintain if changes are needed, we only modify that class, not the entire program oop improves clarity and avoids Code repetetion

(Q8) Function overloading and overriding.

- Function overloading → Same function name diffent parameters (compile time polymorphism)
- Function overriding → A derived class defines the Same function as the base class it happens at runtime (run time polymorphism)

overloading useful when one task is done in diffent ways with diffent inputs, overriding is useful when a subclass needs its own version of base class

(Q9) Encapsulation in BankAccount class

Encapsulation means wrapping data and functions into single unit and hiding the data from direct access.

In a BankAccount class, the Balance and Account numbers are are made private and can be changed only through deposit or withdraw function. This prevents wrong (or) unauthorized Changes to balances

In the Banking Systems, encapsulation ensures data Safety, Security, and prevents misue of Sensitive information.

Q10 Abstract class and pure virtual function

An Abstract class is a class that cannot be used to create objects. It contains at least one pure virtual function. which means the function has no body and must be defined in the derived class.

Ex: a base class shape with a pure virtual function area() can be inherited by circle and square, which can define their own area formulas.

Abstract classes provide a common structure and achive abstraction in oop.

Q2. in this ~~prob~~ program, aclass is used to store key-pair values like a dictionary. Each key is linked to a value and we can add, search and display all pairs using class functions

Encapsulation is important here because the data. (key-value) is kept private, and can be accessed only through class methods.

This prevents accidental changes or misuses of the stored data. and keeps the dictionary organized and safe