1.1 What is a reference variable in C++? Provide an example of how reference variables are used and explain their advantages in programming. (5marks)

Reference variables receive the address (memory location) of the actual parameter, reference variable can pass one or more values from a function and can change the value of the actual parameter.

Example

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

void increment(int &value) {
  value++;  // Increments the original variable
}

int main() {
  int num = 5;
  cout << "Before increment: " << num << endl;
  increment(num);  // Pass num by reference
  cout << "After increment: " << num << endl;
  return 0;
}
```

Explain their advantages in programming.

- it saves memory and processing time.
- Simplicity – Using references can make the code cleaner and easier to read.
- Automatic Dereferencing – When using references, you don't need to explicitly dereference them.
- Maintain original value.
- When the value of the actual parameter needs to be changed.
- When you want to return more than one value from a function (recall that the return statement can return only one value).
- When passing the address would save memory space and time relative to copying a large amount of data.

1.2 Discuss the components of the Standard Template Library (STL), which plays a crucial role in effective data manipulation within programs.

Container – is used to manage an object of a given type.
Iterator – are used to step through the elements of a container.
Algorithms – are used to manipulate data using functions such as sort() and find()

1.3 **Data structure** is a specialized format for organizing, processing, retrieving and storing of data. A **class** is a blueprint for creating objects in object-oriented programming (OOP).
Importance of Data Structures and Classes in Programming

- Organization of Data
- Efficiency

- Modularity
- Reusability

1.4 **Vector** container stores and manages its objects in a dynamic array. The elements of a vector can be accessed randomly.
List containers store the elements sequentially and are implemented as doubly linked lists. Every element in a list point to its immediate predecessor and to its immediate successor.

➔ I would choose a List container because item insertion in the middle or beginning of a vector is time consuming, especially if the vector is large.

2.1

```
#include <iostream>

using namespace std;

class Student
{
    string name;
    int ID;
    int grades[3] = {50, 54, 60};

    public:
        Student(string studentName, int studentID, int studentGrades[3])
        {
            name = studentName;
            ID = studentID;
            for (int i = 0; i < 3; ++i)
                grades[i] = studentGrades[i];
        }

        double Average()
        {
            double sum = 0;

            for (int i : grades)
                sum += i;

            double grade_avg = sum / 3;

            cout<<name<<" Of"<<" ID"<<ID<<" You got: "<<grade_avg<<endl;

            return grade_avg;
        }
};

int main()
{
    int studentGrades[3] = { 50, 54, 60 };

    Student objStudent("Les", 402308182, studentGrades);

    cout << "Average Grade: " << objStudent.Average();

    return 0;
}
```

2.2

```cpp
#include <iostream>

using namespace std;

int factorial(int n)
{
    if (n <= 0)
        return 1;
    else
        return n * factorial(n - 1);
}

int main()
{
    int n;
    cout << "Enter a non-negative integer: ";
    cin >> n;

    cout << "Factorial of " << n << " is: " << factorial(n) << endl;
}
```

3.1

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>

using namespace std;

int fib(int n)
{
    if (n <= 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
}

vector<int> generateFibonacci(int count)
{
    vector<int> sequence;
    for (int i = 0; i < count; i++)
        sequence.push_back(fib(i));

    return sequence;
}

int main()
{
    vector<int> mathSequence = generateFibonacci(10);

    // Output the initial sequence
    cout << "First 10 Fibonacci numbers: ";
    for (int num : mathSequence) {
        cout << num << " ";
    }
    cout << endl;

    // Extend the sequence by adding the next 5 Fibonacci numbers
    for (int i = 10; i < 15; i++) {
        mathSequence.push_back(fib(i));
```

```
    }

    // Output the extended sequence
    cout << "Extended Fibonacci sequence (15 terms): ";
    for (int num : mathSequence) {
        cout << num << " ";
    }
    cout << endl;

    // b) Remove all even numbers from the sequence.

    for (int i : mathSequence)
    {
        if (i % 2 != 0) {
            cout << i << " ";
        }
    }
    cout << endl;

    // c) Replace each number with its square root.

    double sum = 0;

    for (int i : mathSequence)
    {
        double sqrtNum = sqrt(i);
        cout << setprecision(2) << sqrtNum << " ";

        sum += sqrtNum;
    }
    cout << endl;

    // d) Find the sum of all elements in the vector.
    cout << "Sum of square roots: " << fixed << setprecision(2) << sum << endl;
    cout << endl;

    return 0;
}
```

Dynamic Sizing, they can grow and shrink in size automatically

Ease of Use Provide built-in methods for adding, removing, and accessing elements.

Automatic Memory Management Handle memory allocation and deallocation automatically.

Safety Features which can help prevent out-of-bounds access errors.

Standard Library Integration Are part of the C++ Standard Template Library (STL), allowing you to use algorithms and functions from the STL directly.

Ease of Iteration Support range-based for loops, making iteration straightforward and less error-prone.

3.2

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Node
{
   string name;
   int StudentNumber;
   double DP;

   Node *prev;
   Node *next;
};

Node *head = nullptr, *last = nullptr, *newNode;

Node *createNode(string name, int StudentNumber, double DP)
{
   newNode = new Node;

   newNode->name = name;
   newNode->StudentNumber = StudentNumber;
   newNode->DP = DP;

   newNode->prev = nullptr;
   newNode->next = nullptr;

   return newNode;
}

void *addStudent(string name, int StudentNumber, double DP)
{
   Node *newStudent = createNode(name, StudentNumber, DP);
   if (head == nullptr)
      head = newStudent;
   else
   {
      Node *temp = head;
      while (temp->next != nullptr)
         temp = temp->next;
      temp->next = newStudent;
      newStudent->prev = temp;
   }
}

void displayList(Node *head)
```

```
{
    Node *temp = head;

    while (temp != nullptr)
    {
        cout << temp->name << " " << temp->StudentNumber << " " << temp->DP << " " << endl;
        temp = temp->next;
    }
}

void swapFirstLast()
{
    if (head == nullptr || head->next == nullptr)
        return;

    Node *first = head;
    Node *last = head;

    while (last->next != nullptr)
        last = last->next;

    if (first->next == last)
    {
        first->next = nullptr;
        last->prev = nullptr;
        last->next = first;
        first->prev = last;
        head = last;
        return;
    }

    Node *second = first->next;

    first->next = nullptr;
    last->prev = nullptr;
    last->next = second;
    first->prev = last;

    second->prev = last;
    head = last;

    last->next->next = first;
    first->next = nullptr;
}

void findAndPrintMaxDP()
{
    if (head == nullptr)
```

```
    {
        cout << "The list is empty." << endl;
        return;
    }

    Node *maxNode = head;
    Node *temp = head;

    while (temp != nullptr)
    {
        if (temp->DP > maxNode->DP)
        {
            maxNode = temp;
        }
        temp = temp->next;
    }

    cout << "Student with highest DP: " << maxNode->name << " (DP: " << maxNode->DP << ")" <<
endl;
}

void sortWithDP(Node *head)
{
    if (head == nullptr)
        return;

    vector<int> values;
    Node *temp = head;

    while (temp != nullptr)
    {
        values.push_back(temp->DP);
        temp = temp->next;
    }
    sort(values.begin(), values.end());
    temp = head;
    for (int val : values)
    {
        temp->DP = val;
        temp = temp->next;
    }
}

int main()
{
    head = createNode("Alice", 40241001, 67);
    Node *bob = createNode("Bob", 40231002, 84);
    Node *charlie = createNode("Charlie", 40221003, 78);
```

```
    Node *david = createNode("David", 40201004, 67);

    head->prev = nullptr;
    head->next = bob;
    bob->prev = head;
    bob->next = charlie;
    charlie->prev = bob;
    charlie->next = david;
    david->prev = charlie;
    david->next = nullptr;

    cout << "Original list:" << endl;
    displayList(head);
    cout << endl;

    addStudent("Eve", 40221003, 3.6);

    cout << "List after adding Eve:" << endl;
    displayList(head);
    cout << endl;

    cout << "Sort List" << endl;
    sortWithDP(head);
    displayList(head);
    cout << endl;

    swapFirstLast();
    displayList(head);
    cout << endl;

    findAndPrintMaxDP();

    return 0;
}
```