Министерство высшего образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

по лабораторной работе №6

Тема: «Наследование. Виртуальные функции. Полиморфизм»

Выполнил

Студент группы РИС-22-2б

Малых В.О.

Проверил доц. Кафедры ИТАС

Полякова О.А.

Пермь 2023

# Постановка задачи

# Определить класс-контейнер.

# Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.

# Перегрузить операции, указанные в варианте.

# Реализовать класс-итератор. Реализовать с его помощью операции последовательного доступа.

# Написать тестирующую программу, иллюстрирующую выполнение операций.

ВАРИАНТ 15:

Класс- контейнер СПИСОК с ключевыми значениями типа int.

Реализовать операции: [] – доступа по индексу; int() – определение размера списка; \* вектор – умножение элементов списков a[i]\*b[i]; +n - переход вправо к элементу с номером n ( с помощью класса-итератора).

**Контрольные вопросы**

*1. Что такое абстрактный тип данных? Привести примеры АТД.*

Абстрактный тип данных (АТД) - это некоторый тип данных, который определяется не только его структурой и способом хранения, но и операциями, которые над ним могут быть выполнены. АТД представляет собой абстракцию, то есть концептуальный подход к описанию объектов и действий над ними без уточнения его реализации на языке программирования. Примерами АТД являются стек, очередь, связный список.

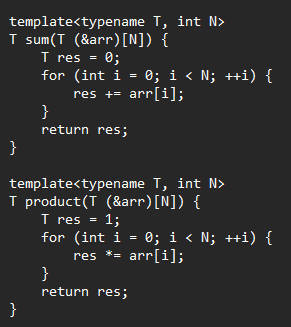
Стек - АТД, в котором элементы добавляются и удаляются только в одном направлении - сверху. Реализация стека может быть выполнена при помощи массива или связного списка. Операции над стеком - это добавление элемента на вершину, удаление элемента с вершины стека и получение элемента с вершины стека без его удаления.

Очередь - АТД, которое обеспечивает упорядоченный доступ к элементам данных, в которой элементы добавляются в одном конце и удаляются из другого конца. Реализация очереди может быть выполнена при помощи массива или связного списка. Операции над очередью - это добавление элемента в конец, удаление элемента из начала и получение элемента из начала.

Связный список - АТД, который представляет собой линейную структуру данных, состоящую из узлов, каждый из которых содержит данные и ссылку на следующий узел в списке. Реализация связного списка может быть выполнена при помощи указателей. Операции над связным списком - это добавление элемента, удаление элемента и получение элемента по индексу.

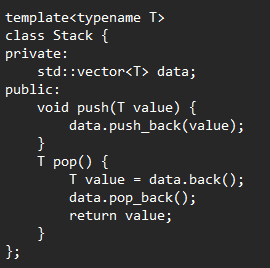
*2. Привести примеры абстракции через параметризацию.*

Примером абстракции через парамтеризацию могут быть шаблоны функций для выполнения арифметических операций над массивами, которые могут быть параметризованы не только типом данных, но и размером массива.



*3. Привести примеры абстракции через спецификацию.*

Примером абстракции через спецификацию может быть шаблонный класс Stack, который определяет общие свойства и методы для стеков, и конкретные классы, например, Stack<int> и Stack<double>, которые реализуют этот шаблон.



*4. Что такое контейнер? Привести примеры.*

Контейнер в языке программирования C++ - это объект, который хранит другие объекты (например, переменные или объекты классов) и управляет их распределением и доступом к ним. В языке C++ имеется множество контейнеров, таких как:

1. std::vector - упорядоченный массив элементов, позволяющий быстро добавлять, удалять и получать доступ к элементам;

2. std::list - двусвязный список элементов, позволяющий быстро добавлять и удалять элементы в середину списка;

3. std::map и std::unordered\_map - ассоциативные контейнеры, которые хранят пары ключ-значение;

4. std::set и std::unordered\_set - контейнеры, которые хранят уникальные элементы в упорядоченном или неупорядоченном порядке.

*5. Какие группы операций выделяют в контейнерах?*

В контейнерах выделяют основные группы операций:

- операции добавления (добавление элемента в контейнер)

- операции удаления (удаление элемента из контейнера)

- операции доступа (получение значения элемента по индексу или итератору)

- операции поиска (поиск элемента в контейнере)

- операции изменения (изменение значения элемента по индексу или итератору)

- операции размера (получение количества элементов в контейнере)

- операции сортировки (упорядочивание элементов контейнера по заданному правилу)

- операции сравнения (сравнение двух контейнеров на равенство или отличие)

*6. Какие виды доступа к элементам контейнера существуют? Привести примеры.*

Существует три основных вида доступа к элементам контейнера:

1) Доступ по индексу - используется для доступа к элементам массивов и упорядоченных контейнеров, таких как std::vector и std::array.

std::vector<int> vec = {1, 2, 3};

int first\_element = vec[0];

2) Итераторный доступ - используется для обхода элементов неупорядоченных контейнеров, таких как std::set и std::unordered\_map.

std::set<int> my\_set = {1, 2, 3};

for(auto it = my\_set.begin(); it != my\_set.end(); ++it)

{

std::cout << \*it << " ";

}

3) Доступ через методы - используется для доступа к элементам контейнеров с помощью методов, таких как at() и find().

std::map<int, std::string> my\_map = {{1, "one"}, {2, "two"}, {3, "three"}};

std::string value\_of\_2 = my\_map.at(2);

auto it = my\_map.find(3);

if(it != my\_map.end())

{

std::cout << "Found key 3 with value " << it->second << std::endl;

}

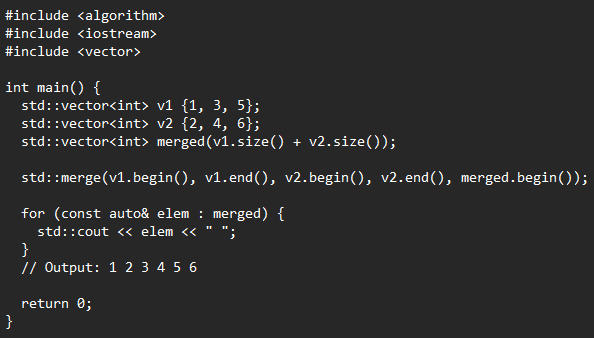
*7. Что такое итератор?*

Итератор - это объект, который позволяет последовательно перебирать элементы коллекции (например, массива, списка или дерева) без необходимости знать их внутреннюю структуру. Итераторы используются для обращения к элементам контейнеров STL (Standard Template Library) в C++. Они позволяют получить доступ к элементам коллекции, изменять их, добавлять и удалять элементы из коллекции и т.д. Итераторы бывают разных типов, например, forward iterator, который позволяет проходить коллекцию в одном направлении, и random access iterator, который позволяет получать доступ к элементам коллекции по индексу.

*8. Каким образом может быть реализован итератор?*

Итератор в C++ может быть реализован с помощью создания класса, который будет иметь определенные методы, такие как operator++, operator\*, operator!= и т.д. Эти методы помогут перемещаться по элементам коллекции и получать доступ к их данным. В классе-контейнере нужно определить методы begin() и end(), которые будут возвращать итераторы на начало и конец коллекции соответственно. Это позволит использовать стандартные алгоритмы на данных коллекции. Кроме того, можно создать пользовательский итератор для любой структуры данных, которая может быть перебрана в цикле.

*9. Каким образом можно организовать объединение контейнеров?*

Один из способов организовать объединение контейнеров в языке C++ - использовать функцию std::merge из библиотеки algorithm, которая принимает два отсортированных диапазона и создает новый отсортированный диапазон, содержащий элементы обоих контейнеров.  


*10. Какой доступ к элементам предоставляет контейнер, состоящий из элементов «ключ-значение»?*

Контейнер, состоящий из элементов «ключ-значение», обеспечивает доступ к значениям по ключу. Например, это может быть контейнер map или unordered\_map из стандартной библиотеки языка C++.

*11. Как называется контейнер, в котором вставка и удаление элементов выполняется на одном конце контейнера?*

Такой контейнер называется стеком (stack).

*12. Какой из объектов (a,b,c,d) является контейнером? a. int mas=10; b. 2. int mas; c. 3. struct {char name[30]; int age;} mas; d. 4. int mas[100];*

d. int mas[100];

*13. Какой из объектов (a,b,c,d) не является контейнером? a. int a[]={1,2,3,4,5}; b. 2. int mas[30]; c. 3. struct {char name[30]; int age;} mas[30]; d. 4. int mas;*

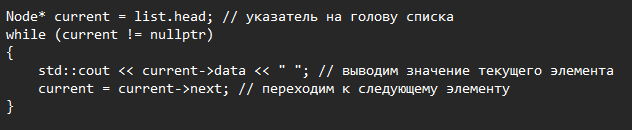
d. int mas

*14. Контейнер реализован как динамический массив, в нем определена операция доступ по индексу. Каким будет доступ к элементам контейнера?*

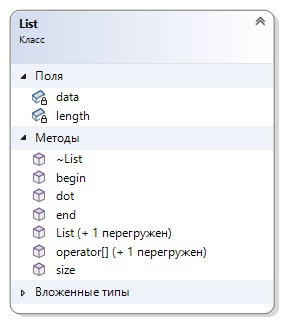
Доступ к элементам контейнера будет осуществляться по индексу, например, через оператор [] или метод at().

*15. Контейнер реализован как линейный список. Каким будет доступ к элементам контейнера?*

Доступ к элементам контейнера будет осуществляться посредством обхода списка в порядке его элементов (от начала до конца) и обращения к каждому элементу по ссылке или указателю. Например, для доступа к первому элементу списка можно использовать указатель на голову списка и оператор разыменовывания:



UML – диаграмма



Код программы

**List.h**

#pragma once

#include <initializer\_list>

class List {

public:

List();

List(const std::initializer\_list<int>& lst);

~List();

int& operator[](int index);

int operator[](int index) const;

int size() const;

int dot(const List& other) const;

class Iterator {

public:

Iterator(List& lst, int index);

Iterator operator+(int n);

int& operator\*();

bool operator!=(const Iterator& other) const;

Iterator& operator++();

private:

List& list;

int currentIndex;

};

Iterator begin();

Iterator end();

private:

int\* data;

int length;

};

**List.cpp**

#include "List.h"

List::List() : data(nullptr), length(0) {}

List::List(const std::initializer\_list<int>& lst) : data(new int[lst.size()]), length(lst.size()) {

int i = 0;

for (auto& el : lst) {

data[i] = el;

++i;

}

}

List::~List() {

delete[] data;

}

int& List::operator[](int index) {

return data[index];

}

int List::operator[](int index) const {

return data[index];

}

int List::size() const {

return length;

}

int List::dot(const List& other) const {

int result = 0;

for (int i = 0; i < length; ++i) {

result += data[i] \* other[i];

}

return result;

}

List::Iterator::Iterator(List& lst, int index) : list(lst), currentIndex(index) {}

List::Iterator List::Iterator::operator+(int n) {

return Iterator(list, currentIndex + n);

}

int& List::Iterator::operator\*() {

return list[currentIndex];

}

bool List::Iterator::operator!=(const Iterator& other) const {

return currentIndex != other.currentIndex;

}

List::Iterator& List::Iterator::operator++() {

++currentIndex;

return \*this;

}

List::Iterator List::begin() {

return Iterator(\*this, 0);

}

List::Iterator List::end() {

return Iterator(\*this, length);

}

**main.cpp**

#include <iostream>

#include "list.h"

int main() {

List a{ 1, 2, 3 };

List b{ 4, 5, 6 };

std::cout << "a[1] = " << a[1] << std::endl;

std::cout << "b[2] = " << b[2] << std::endl;

std::cout << "a size = " << a.size() << std::endl;

std::cout << "b size = " << b.size() << std::endl;

std::cout << "a \* b = " << a.dot(b) << std::endl;

List::Iterator it = a.begin() + 1;

std::cout << "Second element of a = " << \*it << std::endl;

++it;

std::cout << "Third element of a = " << \*it << std::endl;

return 0;

}