**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №18.6 на тему

«Абстрактные типы данных, списки»

Вариант №11

Выполнил студент группы ИВТ-20-2б

Сабуров Павел Алексеевич

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –научиться работать с одним с абстрактными типами данных и с итераторами.

**Постановка задачи**

Задача – реализовать на языке программирования C++ программу, демонстрирующую работу абстрактных классов, контейнеров, итераторов.

**Исходные данные для варианта №11:**

* Контейнер – список;
* Итератор – абстрактный указатель, реализованный в виде класса;

**Анализ задачи**

Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code

Объявление класса «Список» в заголовочном файле:

#pragma once

#include "ListIterator.h"

class ListInt

{

private:

int\* \_data;

int \_length;

ListIterator \_begin;

ListIterator \_end;

ListIterator \_current;

public:

ListInt(int length, int fillValue);

ListInt(const ListInt& copied);

~ListInt();

ListIterator Begin() { return \_begin; }

ListIterator End() { return \_end; }

ListIterator SelectedItem() { return \_current; }

int& operator [] (int index);

ListInt& operator = (const ListInt& assigned);

ListInt operator + (const ListInt& adder);

int Length() { return \_length; }

void MoveSelector(int offset);

};

Помимо самих данных, данный список содержит в себе три индекса:

* Начало списка;
* Выбранный индекс;
* Конец списка;

Это и есть те самые итераторы, работы с которыми будет показана ниже.

Объявление класса-итератора в заголовочном файле:

#pragma once

class ListIterator

{

friend class ListInt;

private:

int\* \_pointedElement;

public:

ListIterator() { \_pointedElement = nullptr; }

ListIterator(const ListIterator& copied) { \_pointedElement = copied.\_pointedElement; }

bool operator == (const ListIterator& compared) { return \_pointedElement == compared.\_pointedElement; }

bool operator != (const ListIterator& compared) { return \_pointedElement != compared.\_pointedElement; }

int& operator \*() const { return \*\_pointedElement; }

void operator ++() { ++\_pointedElement; }

void operator --() { --\_pointedElement; }

void operator += (int addValue);

void operator -= (int subValue);

};

Реализация операторов перемещения указателя итератора:

void ListIterator::operator+=(int addValue)

{

for (int i = 0; i < addValue; i++)

\_pointedElement++;

}

void ListIterator::operator-=(int subValue)

{

for (int i = 0; i < subValue; i--)

\_pointedElement--;

}

Основная программа, демонстрирующая работу итератора:

#include <iostream>

#include <windows.h>

#include <string>

#include "Header files\ListInt.h"

using namespace std;

void PrintList(ListInt& list)

{

for (int i = 0; i < list.Length(); i++)

cout << list[i] << ' ';

cout << endl;

}

int main()

{

ListInt list(24, 2);

PrintList(list);

ListInt copied(list);

PrintList(copied);

list[5] = 45;

list[21] = 38;

PrintList(list);

ListInt added = list + copied;

PrintList(added);

list[23] = 99;

PrintList(list);

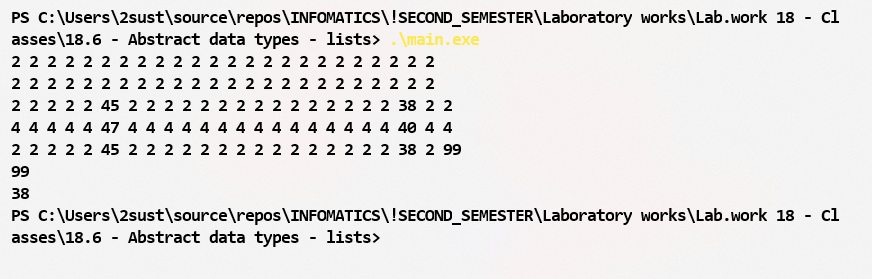
cout << \*(list.End()) << endl;

list.MoveSelector(21);

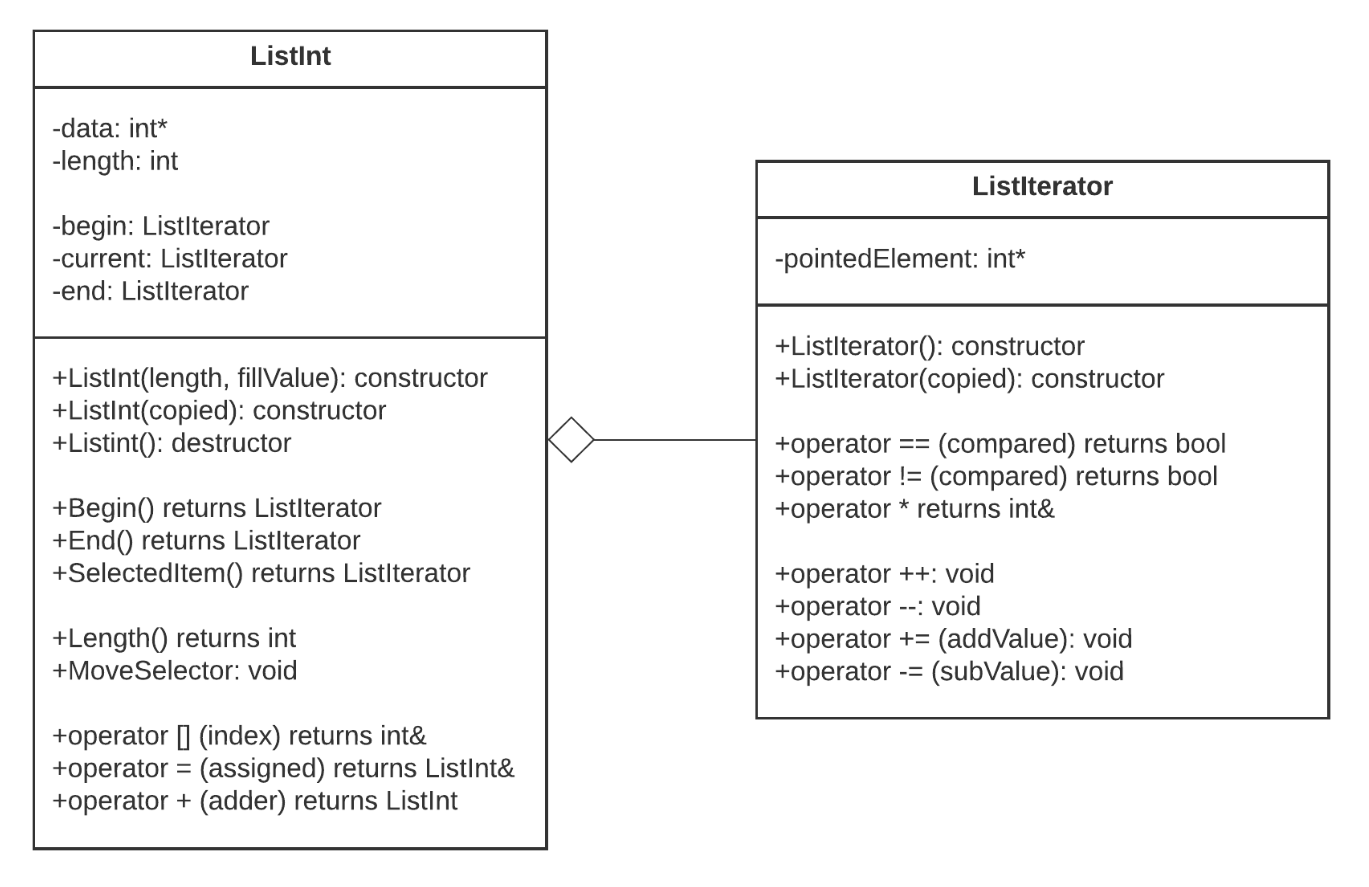
cout << \*(list.SelectedItem()) << endl;

}

**Скриншот выполненной программы:**



**UML-диаграмма классов программы:**



**Ответы на вопросы:**

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

АТД - тип данных, определяемый только через операции, которые могут выполняться над соответствующими объектами безотносительно к способу представления этих объектов.

Примеры: Класс, Array, List, Map, Queue, Stack, Table, Tree, Vector

**Привести примеры абстракции через параметризацию.**

#include <stdio.h>

struct date

{

int month; // месяц

int day; // день

int year; // год

};

void set\_date(date\* f, int d, int m, int y)

{

f->day = d;

f->month = m;

f->year = y;

}

void print\_date(date\* f)

{

printf("%d.%d.%d", f->day, f->month, f->year);

}

int main()

{

date today;

set\_date(&today, 2, 4, 2014);

print\_date(&today);

getchar();

return 0;

}

**Привести примеры абстракции через спецификацию.**

#include <stdio.h>

struct date

{

int month; // месяц

int day; // день

int year; // год

};

void set\_date(date\* f, int d, int m, int y)

{

f->day = d;

f->month = m;

f->year = y;

}

void print\_date(date\* f)

{

printf("%d.%d.%d", f->day, f->month, f->year);

}

int main()

{

date today;

set\_date(&today, 2, 4, 2014);

print\_date(&today);

getchar();

return 0;

}

**Что такое контейнер? Привести примеры.**

Контейнер – набор однотипных элементов. Пример: Встроенный массив.

**Какие группы операций выделяют в контейнерах?**

Операции доступа к элементам, которые обеспечивают и операцию замены значений элементов, Операции добавления и удаления элементов или групп элементов, Операции поиска элементов и групп элементов, Операции объединения контейнеров

Специальные операции, которые зависят от вида контейнера

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

Последовательный (список, через указатель на следующий элемент), прямой (массив, через индекс) и ассоциативный (словарь, через значение парной переменной).

**Что такое итератор?**

Итератор – это объект, который обеспечивает последовательный доступ к элементам контейнера.

**Каким образом может быть реализован итератор?**

Итератор может быть реализован как часть класса-контейнера в виде набора методов.

v.first() перейти к первому элементу

v.last() перейти к последнему элементу

v.next() перейти к следующему элементу

v.prev() перейти к предыдущему элементу

v.skip(n) перейти на n элементов вперед

v.skip(-n) перейти на n элементов назад

v.current() получить текущий элемент

**Каким образом можно организовать объединение контейнеров?**

Простое сцепление двух контейнеров: в новый контейнер попадают сначала элементы первого контейнера, потом второго, операция не коммутативна.

Объединение упорядоченных контейнеров, новый контейнер тоже будет упорядочен, операция коммутативна.

Объединение контейнеров как объединение множеств, в новый контейнер попадают только те элементы, которые есть хотя бы в одном контейнере, операция коммутативна.

Объединение контейнеров как пересечение множеств, в новый контейнер попадают только те элементы, которые есть в обоих контейнерах, операция коммутативна.

Для контейнеров-множеств может быть еще реализована операция вычитания, в контейнер попадают только те элементы первого контейнера, которых нет во втором, операция не коммутативна.

Извлечение части элементов из контейнера и создание нового контейнера. Эта операция может быть выполнена с помощью конструктора, а часть контейнера задается двумя итераторами.

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

Ассоциативный.

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

Стек.

**Какой из объектов (a,b,c,d) является контейнером?**

**a. int mas=10;**

**b. int mas;**

**c. struct {char name[30]; int age;} mas;**

**d. int mas[100];**

D.

**Какой из объектов (a,b,c,d) не является контейнером?**

**a. int a[]={1,2,3,4,5};**

**b. int mas[30];**

**c. struct {char name[30]; int age;} mas[30];**

**d. int mas;**

D.

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

Прямой.

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

Последовательный.

Полный исходный код программы на языке программирования C++:

ListInt.h

#pragma once

#include "ListIterator.h"

class ListInt

{

private:

int\* \_data;

int \_length;

ListIterator \_begin;

ListIterator \_end;

ListIterator \_current;

public:

ListInt(int length, int fillValue);

ListInt(const ListInt& copied);

~ListInt();

ListIterator Begin() { return \_begin; }

ListIterator End() { return \_end; }

ListIterator SelectedItem() { return \_current; }

int& operator [] (int index);

ListInt& operator = (const ListInt& assigned);

ListInt operator + (const ListInt& adder);

int Length() { return \_length; }

void MoveSelector(int offset);

};

ListInt.cpp

#include "..\Header files\ListInt.h"

#include "..\Header files\ListIterator.h"

#include <iostream>

using namespace std;

ListInt::ListInt(int length, int fillValue)

{

\_length = length;

\_data = new int[\_length];

\_begin.\_pointedElement = &\_data[0];

\_end.\_pointedElement = &\_data[\_length - 1];

\_current = \_begin;

for (int i = 0; i < \_length; i++)

\_data[i] = fillValue;

}

ListInt::ListInt(const ListInt& copied)

{

\_length = copied.\_length;

\_data = new int[\_length];

\_begin = copied.\_begin;

\_current = copied.\_current;

\_end = copied.\_end;

for (int i = 0; i < \_length; i++)

\_data[i] = copied.\_data[i];

}

ListInt::~ListInt()

{

delete[] \_data;

\_data = nullptr;

}

int& ListInt::operator[](int index)

{

try

{

if (index < 0)

throw - 1;

if (index >= \_length)

throw 1;

return \_data[index];

}

catch (int code)

{

switch (code)

{

case -1:

cerr << "Error: index is out of range => less than 0;\n";

break;

case 1:

cerr << "Error: index is out of range => more than length;\n";

break;

}

}

}

ListInt& ListInt::operator=(const ListInt& assigned)

{

\_length = assigned.\_length;

delete[] \_data;

\_data = new int[\_length];

\_begin = assigned.\_begin;

\_current = assigned.\_current;

\_end = assigned.\_end;

for (int i = 0; i < \_length; i++)

\_data[i] = assigned.\_data[i];

return \*this;

}

ListInt ListInt::operator+(const ListInt& adder)

{

try

{

if (\_length != adder.\_length)

throw - 1;

ListInt temporarilyList(\_length, 0);

for (int i = 0; i < \_length; i++)

temporarilyList[i] = \_data[i] + adder.\_data[i];

return temporarilyList;

}

catch (int code)

{

if (code == -1)

cerr << "Error: list length are not the same;\n";

return \*this;

}

}

void ListInt::MoveSelector(int offset)

{

if (offset < 0)

\_current -= -offset;

else

\_current += offset;

}

ListIterator.h

#pragma once

class ListIterator

{

friend class ListInt;

private:

int\* \_pointedElement;

public:

ListIterator() { \_pointedElement = nullptr; }

ListIterator(const ListIterator& copied) { \_pointedElement = copied.\_pointedElement; }

bool operator == (const ListIterator& compared) { return \_pointedElement == compared.\_pointedElement; }

bool operator != (const ListIterator& compared) { return \_pointedElement != compared.\_pointedElement; }

int& operator \*() const { return \*\_pointedElement; }

void operator ++() { ++\_pointedElement; }

void operator --() { --\_pointedElement; }

void operator += (int addValue);

void operator -= (int subValue);

};

ListIterator.cpp

#include "..\Header files\ListIterator.h"

void ListIterator::operator+=(int addValue)

{

for (int i = 0; i < addValue; i++)

\_pointedElement++;

}

void ListIterator::operator-=(int subValue)

{

for (int i = 0; i < subValue; i--)

\_pointedElement--;

}

main.cpp

#include <iostream>

#include <windows.h>

#include <string>

#include "Header files\ListInt.h"

using namespace std;

void PrintList(ListInt& list)

{

for (int i = 0; i < list.Length(); i++)

cout << list[i] << ' ';

cout << endl;

}

int main()

{

ListInt list(24, 2);

PrintList(list);

ListInt copied(list);

PrintList(copied);

list[5] = 45;

list[21] = 38;

PrintList(list);

ListInt added = list + copied;

PrintList(added);

list[23] = 99;

PrintList(list);

cout << \*(list.End()) << endl;

list.MoveSelector(21);

cout << \*(list.SelectedItem()) << endl;

}