Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №18.8**

Дисциплина: «Информатика»

Тема: Программа, управляемая событиями

Вариант 4

Выполнил:

Студент группы РИС-20-1б

Еске Вячеслав Сергеевич

Проверила:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь, 2021**

**Цель задачи**

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Реализация класса-контейнера.

Для достижения поставленной цели необходимо решить следующие задачи:

* Проанализировать теоретические сведения о ООП
* Абстрактные типы данных. Контейнеры
* Реализовать алгоритм поставленной задачи
* Создать отдельные файлы: .cpp для описания методов класса, .cpp для описания главной функции, .h для описания класса

Постановка задачи

Базовый класс: ПЕЧАТНОЕ\_ИЗДАНИЕ(PRINT) Название– string Автор – string Производный класс КНИГА (BOOK) Количество страниц - int Издательство - string Группа – Вектор (Vector). Команды:

• Создать группу (формат команды: m количество элементов группы).

• Добавить элемент в группу (формат команды: +)

• Удалить элемент из группы (формат команды -)

• Вывести информацию об элементах группы (формат команды: s)

• Вывести информацию о названии элемента группы с номером k (формат команды: z k, где k – целое число)

• Конец работы (формат команды: q)

Анализ задачи

1. Определить какие операции должны быть выполнены по заданию:

* Создание класса

class Person

* Создание множества и заполнение класса
* Создание методов получения размера множества, пересечений и доступа по индексу
* Отчистка памяти

1. Для решения задачи используются переменные:

Несколько переменных целочисленного значения для ввода чисел и реализации меню

int s, in, menu = 4;

Указатель для работы с динамической памятью

1. Ввод данных осуществляется посредством функции

cin >> rub;

Вывод данных осуществляется посредством функции

cout << "Искомый элемент = " << a.give(in);

1. Создание метода класса , который ищет пересечения класса:

void Vector::cross()

{

int k = 0;

for (int i = 0; i < size - 1; i++)

{

for (int j = i + 1; j < size; j++)

{

if (data[i] == data[j])

k++;

}

if (k != 0)

cout << "\nЭлемент " << data[i] << " встречается " << k + 1 << " раз";

k = 0;

}

1. Создание методов класса для нахождения размера и доступа по индексу

int give(int index);

int givesize();

Создание контейнера

Employee(int o, string n, float z, string d)

{

old = o;

name = n;

zp = z;

dol = d;

}

Код

#include <iostream>

#include "Book.h"

#include "Object.h"

#include "Print.h"

#include "Vector.h"

#include "Event.h"

#include <windows.h>

#include "Dialog.h"

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

/\*Print\* a = new Print;//создаем объект класса Print

a->Input();

Book \*b=new Book; //создаем объект класса Lorry

b->Input();

Vector v(10);//Создаем вектор

Object\* p = a;//ставим указатель на объект класса Print

v.Add(p);//добавляем объект в вектор

p=b;//ставим указатель на объект класса Lorry

v.Add(p); //добавляем объект в вектор

v.Show();//вывод вектора

v.Del();//удаление элемента

cout << "\nРазмер вектора=" << v();\*/

Dialog D;

D.Execute();

}

#pragma once

#include <iostream>

#include "Vector.h"

#include "Event.h"

using namespace std;

class Dialog : public Vector

{

public:

Dialog(void);//контруктор

virtual ~Dialog(void);//деструктор

virtual void GetEvent (TEvent &event);//получить событие

virtual int Execute();//главный цикл обработки событий

virtual void HandleEvent (TEvent& event); //обработчик

virtual void ClearEvent (TEvent& event);//очистить событие

int Valid();//проверка атрибута EndState

void EndExec();//обработка события «конец работы»

protected:

int EndState;

};

#include "Dialog.h"

//конструктор

Dialog::Dialog(void):Vector()

{

EndState = 0;

}

//деструктор

Dialog::~Dialog(void) { }

//получение события

void Dialog::GetEvent(TEvent &event)

{

string OpInt = "+-szqam"; //строка содержит коды операций

string s;

string param;

char code;

cout << '>';

cin >> s;

code = s[0];//первый символ команды

if(OpInt.find(code)>=0)//является ли символ кодом операции

{

event.what = evMessage; switch (code)

{

case 'm':event.command = cmMake; break;//создать группу

case '+': event.command=cmAdd;break;//добавить объект в группу

case '-': event.command=cmDel;break;//удалить объект из группы

case 's': event.command=cmShow;break;//просмотр группы

case 'q': event.command = cmQuit;break; //конец работы

case 'z': event.command = cmGet; break;

}

//выделяем параметры команды, если они есть

if(s.length()>1)

{

param = s.substr(1, s.length() - 1);

int A = atoi(param.c\_str());//преобразуем парметр в число

event.a=A;//записываем в сообщение

}

}

else event.what = evNothing;//пустое событие

}

int Dialog::Execute()

{

TEvent event; do {

EndState = 0;

GetEvent(event); //получить событие

HandleEvent(event); //обработать событие

} while (!Valid()); return EndState;

}

int Dialog::Valid()

{

if (EndState == 0) return 0;

else return 1;

}

void Dialog::ClearEvent(TEvent & event)

{

event.what = evNothing;//пустое событие

}

void Dialog::EndExec()

{

EndState = 1;

}

//обработчик событий

void Dialog::HandleEvent(TEvent& event) {

if (event.what == evMessage) {

switch (event.command) {

case cmMake://создание группы

size = event.a; //размер группы

beg = new Object \* [size];//выделяем память под массив указателей

cur = 0; //текущая позиция

ClearEvent(event);//очищаем событие

break;

case cmAdd://добавление

Add();

ClearEvent(event);

break;

case cmDel:

Del(); //удаление

ClearEvent(event);

break;

case cmShow:

Show(); //просмотр

ClearEvent(event);

break;

case cmQuit:

EndExec(); //выход

ClearEvent(event);

break;

default:Vector::HandleEvent(event);

};

};

};

#pragma once

#include "Book.h"

#include "Object.h"

#include "Print.h"

#include <iostream>

using namespace std;

class Vector

{

public:

Vector(void);

Vector(int);//конструктор с параметрами

~Vector(void);//деструктор

void Add();//добавление элемента в вектор

void Del();

void Show();

int operator()();//размер вектора

protected:

Object\*\* beg;//указатель на первый элемент вектора

int size;//размер

int cur;//текущая позиция

void HandleEvent(const TEvent& e);

};

#include "Vector.h"

#include "Event.h"

Vector::Vector(void) {}

//деструктор

Vector::~Vector(void)

{

if (beg != 0) delete[] beg;

beg = 0;

}

//конструктор с параметрами

Vector::Vector(int n)

{

beg = new Object \* [n];

cur = 0;

size = n;

}

//добавление объекта, на который указывает указатель p в вектор

void Vector::Add()

{

Object\* p;

//выбор из объектов двух возможных классов

cout<<"1.Print"<<endl;

cout<<"2.Book"<<endl;

int y;

cin>>y;

if(y==1)//добавление объекта класса Car

{

Print\* a = new (Print);

a->Input();//ввод значений атрибутов

p=a;

if (cur < size)

{

beg[cur] = p;//добавление в вектор

cur++;

}

}

else

if (y == 2) //добавление объекта класса Lorry

{

Book\* b = new Book;

b->Input();

p = b;

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

else return;

}

//просмотр вектора

void Vector::Show() {

if (cur == 0) cout << "Empty" << endl;

Object\*\* p = beg;//указатель на указатель типа Object

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

{

(\*p)->Show();//вызов метода Show() (позднее связывание)

p++;//передвигаем указатель на следующий объект

}

}

//операция, которая возвращает размер вектора

int Vector::operator ()() { return cur; }

//удаление элемента из вектора, память не освобождается!

void Vector::Del()

{

if (cur == 0) return;//пустой

cur--;

}

void Vector::HandleEvent(const TEvent& e)

{

if (e.what == evMessage)

{

Object\*\* p = beg;

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

{

if (i==e.a-1) (\*p)->HandleEvent(e);//вызов метода (позднее связывание)

p++;//передвигаем указатель на следующий объект

}

}

}

#pragma once

#include "Print.h"

class Book :

public Print

{

public:

Book(void);

~Book(void);

void Show();

void Input();

Book(string, string, int, string);

Book(const Book&);

int Get\_page() { return page; }

void Set\_page(int);

string Get\_izdat() { return izdat; }

void Set\_izdat(string);

Book& operator=(const Book&);

protected:

int page;

string izdat;

};

#include "Book.h"

Book::Book(void) :Print()

{

page = 0;

izdat = "";

}

Book::~Book(void) {}

Book::Book(string N,string A, int P, string I ) : Print(N, A)

{

page = P;

izdat = I;

}

Book::Book(const Book& B)

{

name = B.name;

author = B.author;

page = B.page;

izdat = B.izdat;

}

void Book::Set\_page(int P)

{

page = P;

}

void Book::Set\_izdat(string I)

{

izdat = I;

}

Book& Book::operator=(const Book& b)

{

if (&b == this) return \*this;

name = b.name;

author = b.author;

page = b.page;

izdat = b.page;

return \*this;

}

void Book::Show()

{

cout << "\nНазвание : " << name;

cout << "\nАвтор : " << author;

cout << "\nКоличество страниц : " << page;

cout << "\nИздательство : " << izdat;

cout << "\n";

}

void Book::Input() {

cout << "\nНазвание:";

cin >> name;

cout << "\nАвтор:";

cin >> author;

cout << "\nКоличество страниц:";

cin >> page;

cout << "\nИздательство: ";

cin >> izdat;

}

#pragma once

#include "Object.h"

#include <string>

#include <iostream>

using namespace std;

class Print : public Object

{

public: public:

Print(void);//конструктор без параметров

virtual ~Print(void);//деструктор

void Show();//функция для просмотра атрибутов класса с помощью указателя

void Input();//функция для ввода значений атрибутов

Print(string,string);//конструктор с параметрами

Print(const Print&);//конструктор копирования

//селекторы

string Get\_name() { return name; }

string Get\_author() { return author; }

//модификаторы

void Set\_name(string);

void Set\_author(string);

Print& operator=(const Print&);//перегрузка операции присваивания

void HandleEvent(const TEvent& e);

protected:

string name;

string author;

};

#include "Print.h"

//конструктор без параметров

Print::Print(void) {

name = "";

author = "";

}

//деструктор

Print::~Print(void) { }

//констрктор с параметрами

Print::Print(string N,string A) {

name = N;

author = A;

}

//конструктор копирования

Print::Print(const Print& print) {

name = print.name;

author = print.author;

}

//селекторы

void Print::Set\_name(string N) {

name = N;

}

void Print::Set\_author(string A) {

author = A;

}

//оператор присваивания

Print& Print::operator=(const Print&p) {

if (&p == this) return \*this;

name = p.name;

author = p.name;

return \*this;

}

//метод для просмотра атрибутов

void Print::Show()

{

cout << "\nНазвание : " << name;

cout << "\nАвтор : " << author;

cout << "\n";

}

//метод для ввода значений атрибутов

void Print::Input()

{

cout << "\nНазвание:";

cin >> name;

cout << "\nАвтор:";

cin >> author;

}

void Print::HandleEvent(const TEvent& e) {

if (e.what == evMessage)//событие-сообщение

{

switch (e.command)

{

case cmGet:cout << "Название: " << Get\_name() << endl;

break;

}

}

}

#pragma once

#include "Event.h"

class Object

{

public:

Object(void);

virtual void Show() = 0;

virtual void Input() = 0;

virtual ~Object(void);

virtual void HandleEvent(const TEvent& e) = 0;

};

#include "Object.h"

Object::Object(void) { }

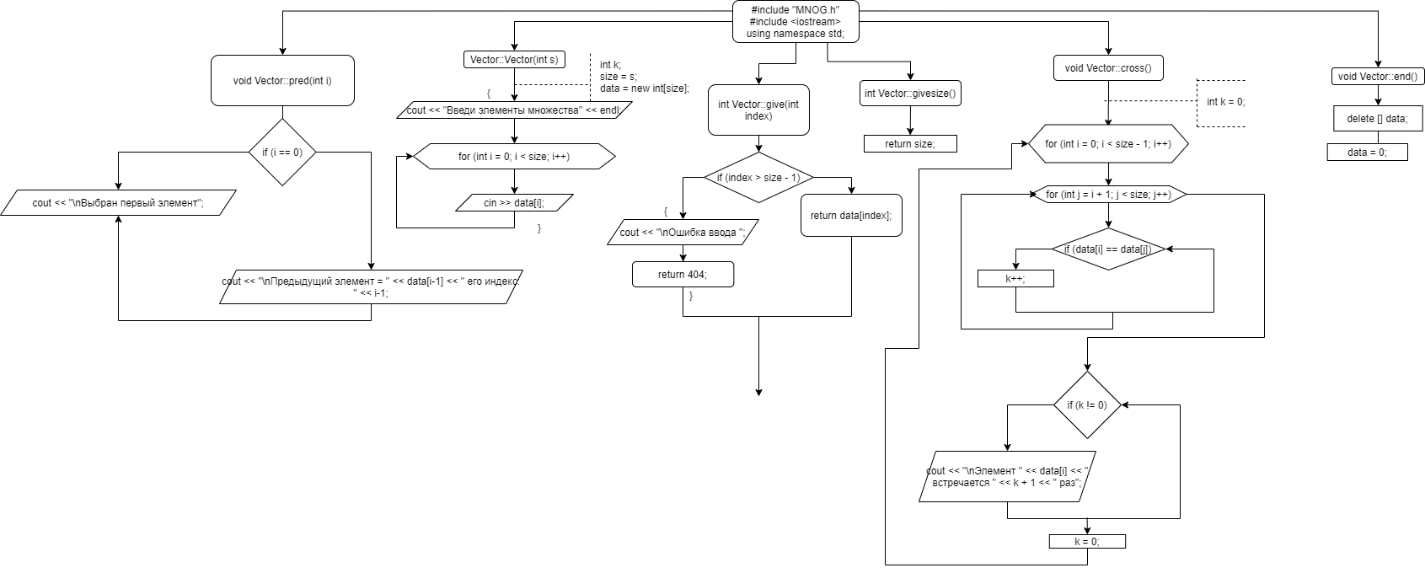
Object::~Object(void) { }

Блок-схема:

Main.cpp:



Pair.cpp:



Pair.h:



Работа кода

