Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное учреждение высшего образования  
«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т  
по лабораторной работе**

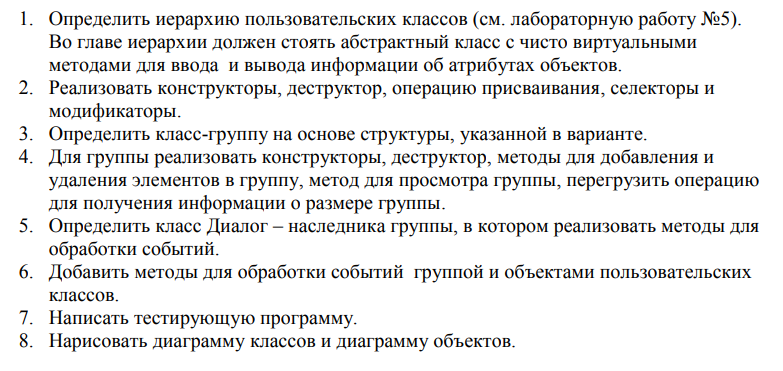
Дисциплина: «Основы алгоритмизации и программирования»

Выполнил  
студент группы ИВТ-21-1б  
Мифтахов И.Г

Проверил  
ст. п. кафедры ИТАС  
Яруллин Д.В.

Пермь 2022

Постановка задачи:

 Анализ задачи:

1. String title – название, string author – автор, int pages – кол-во страниц, int what – какое событие, int command – код команды, int message – сообщение, object\*\* beg – указатель на указатель объекта, int size – размер, int cur – счетчик, int EndState – статус.
2. Главный класс в иерархии – абстрактный класс object с чистыми виртуальными ф-иями ввода и вывода. Далее, print, класс имеет конструкторы, геттеры и сеттеры, перегрузки операторов, переопределенные чистые виртуальные ф-ии. Следующий класс magazin можно описать аналогично его предку print. Класс List ответственен за ф-ии команд, создание списка происходит не только в этом классе, также этот класс имеет наследованные конструкторы, как и класс magazin. Последний класс Dialog обрабатывает команды и через систему switch-ей отдает команды и делает проверки, благодаря нему происходит контакт с юзером.

Ответы на вопросы:

1. Объект, в который включены другие объекты. Огород из растений, поливов, удобрений. Автомобиль из колес, каркаса и двигателя.
2. template <class Type, class Allocator= allocator<Type>>

class list

1. List(void)

{

beg = 0;

size = 0;

cur = 0;

}

List(int n)

{

beg = new object \*[n];

size = n;

cur = 0;

}

List(const List& l)

{

beg = l.beg;

size = l.size;

cur = l.cur;

}

1. ~List(void)

{

}

1. //просмотр списка

void Info\_All()

{

if (cur == 0) cout << "Empty\n";

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

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

{

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

++p;//передвигаем указатель на след объект

}

}

1. Иерархию объектов, построенную на агрегации.
2. Для комфортного наследования.
3. Событие – какое-либо действие(создание, модификация, удаление объекта) как реакция на ввод команды. События используются для системы диалогов между пользователем и программой.
4. Код команды и самое сообщение(содержание).
5. struct TEvent

{

int what;

union

{

int command;

struct

{

int message;

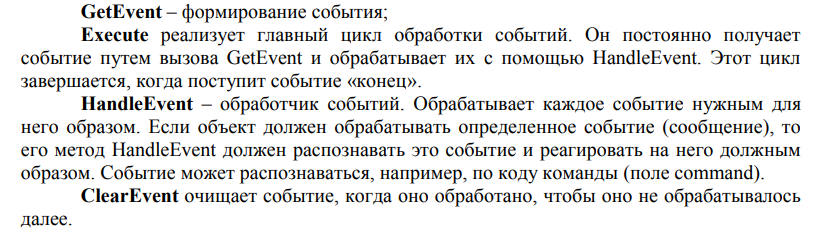
int a;

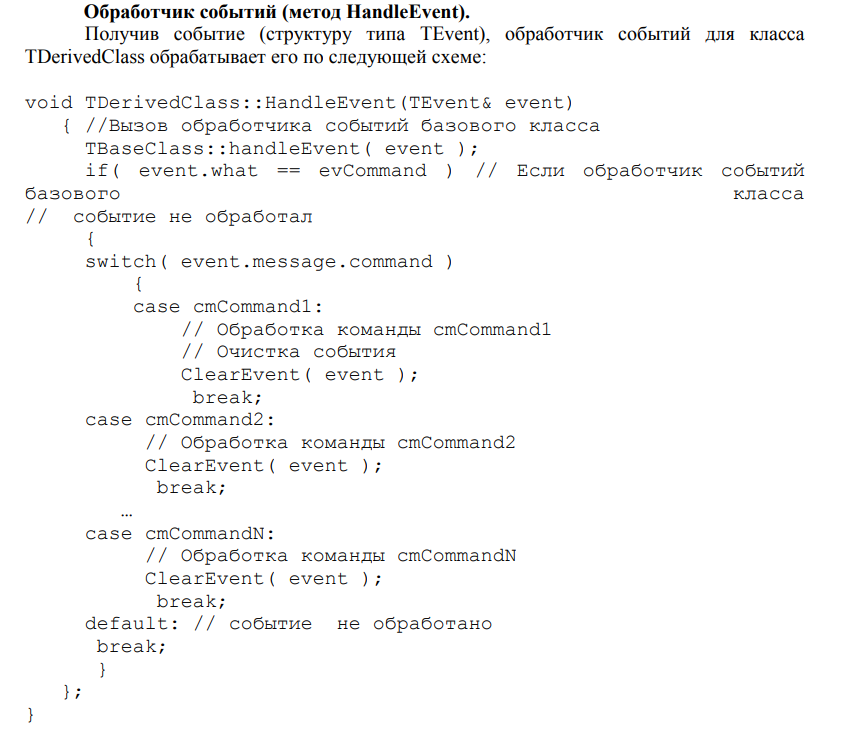
};

};

};

1. 0, 1, 2 в соотвествие с порядком полей.
2. Значение берутся от обработки команды(параметр) в случае действия пользователя.
3. Код ссобщения, а – преобразованный параметр(команда) в число.



15. 

16. Очищает событие, чтобы оно не обрабатывалось далее. Отправляет в структуру пустое событие.

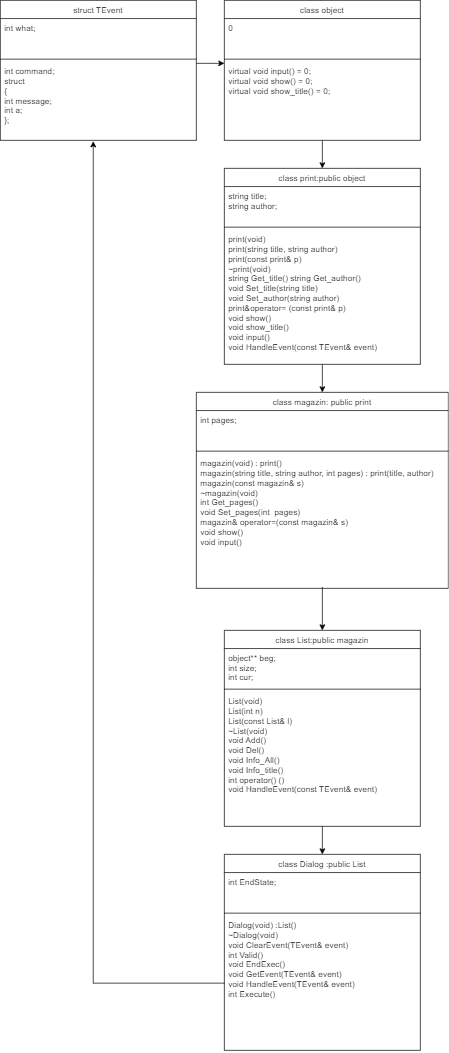
17. Обработчик событий, работает по коду команды.

18. Формирование события.

19. Конечное состояние программы. Содержится в классе Dialog.

20. Для проверки команды на ее «адекватность» и подходит ли она.

Блок-схема:



Код программы:

#include<iostream>

#include<string>

const int evNothing = 0;

const int evMessage = 100;

const int cmAdd = 1;

const int cmDel = 2;

const int cmGet = 3;

const int cmShow = 4;

const int cmMake = 6;

const int cmQuit = 101;

struct TEvent

{

int what;

union

{

int command;

struct

{

int message;

int a;

};

};

};

using namespace std;

class object

{

public:

virtual void input() = 0;

virtual void show() = 0;

virtual void show\_title() = 0;

};

class print:public object

{

protected:

string title;

string author;

public:

//конструкторы

print(void)//без параметров

{

title = "";

author = "";

}

print(string title, string author)//с параметрами

{

this->title = title;

this->author = author;

}

print(const print& p)//копирования

{

title = p.title;

author = p.author;

}

//деструктор

~print(void)

{

}

//селекторы

string Get\_title() { return title; }

string Get\_author() { return author; }

//модификаторы

void Set\_title(string title) { this->title = title; }

void Set\_author(string author) { this->author = author; }

//перегрузка оператора присваивания

print&operator= (const print& p)

{

title = p.title;

author = p.author;

return \*this;

}

//метод для вывода атрибутов

void show()

{

cout << "Название: " << title << "\n";

cout << "Автор: " << author << "\n\n";

}

void show\_title()

{

cout << "Название: " << title << "\n";

}

//метод для ввода атрибутов

void input()

{

cout << " Введите название >> "; cin >> this->title;

cout << "Введите автора >> "; cin >> this->author;

cout << "\n";

}

void HandleEvent(const TEvent& event)

{

}

};

class magazin: public print

{

protected:

int pages;

public:

//конструкторы

magazin(void) : print()

{

pages = 0;

}

magazin(string title, string author, int pages) : print(title, author)

{

this->pages = pages;

}

magazin(const magazin& s)

{

title = s.title;

author = s.author;

pages = s.pages;

}

//деструктор

~magazin(void)

{

}

//селектор

int Get\_pages() { return pages; }

//модификатор

void Set\_pages(int pages) { this->pages = pages; }

//перегрузка оператора присваивания

magazin& operator=(const magazin& s)

{

title = s.title;

author = s.author;

pages = s.pages;

return \*this;

}

//метод для вывода атрибутов

void show()

{

cout << "Название: " << title << "\n";

cout << "Автор: " << author << "\n";

cout << "Количество страниц: " << pages << "\n\n";

}

//метод для ввода атрибутов

void input()

{

cout << "Название >> "; cin >> title;

cout << "Автор >> "; cin >> author;

cout << "Количество страниц >> "; cin >> pages;

cout << "\n";

}

};

class List:public magazin

{

protected:

object\*\* beg;

int size;

int cur;

public:

//конструкторы

List(void)

{

beg = 0;

size = 0;

cur = 0;

}

List(int n)

{

beg = new object \*[n];

size = n;

cur = 0;

}

List(const List& l)

{

beg = l.beg;

size = l.size;

cur = l.cur;

}

//деструктор

~List(void)

{

}

//добавление объекта на который указывает указатель р в список

void Add()

{

object\* p;

//выбор из объектов 2 возможных классов

cout << "1.print\n2.magazin\nEnter num type >> ";

int y;

cin >> y; cout << "\n";

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

{

print\* a = new (print);

a->input();

p = a;

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

else if (y == 2)//добавление объекта класса magazin

{

magazin\* b = new (magazin);

b->input();

p = b;

if (cur < size)

{

beg[cur] = p;

cur++;

}

}

else return;

}

//удаление элемента из списка, но при этом память не освобождается

void Del()

{

if (cur == 0) return;

cur--;

}

//просмотр списка

void Info\_All()

{

if (cur == 0) cout << "Empty\n";

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

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

{

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

++p;//передвигаем указатель на след объект

}

}

//просмотр элемента из списка

void Info\_title()

{

if (cur == 0) cout << "Empty\n";

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

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

{

(\*p)->show\_title();//вызов метода Show() (позднее срабатывание)

++p;//передвигаем указатель на след объект

}

}

//операция возвращающая размер списка

int operator() ()

{

return cur;

}

void HandleEvent(const TEvent& event)

{

if (event.what == evMessage)

{

object\*\* p = beg;

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

{

(\*p)->show();

++p;

}

}

}

};

class Dialog :public List

{

protected:

int EndState;

public:

//конструктор

Dialog(void) :List()

{

EndState = 0;

}

//деструктор

~Dialog(void)

{

}

//очищение события

void ClearEvent(TEvent& event)

{

event.what = evNothing;//пустое событие

}

//проверка EndState

int Valid()

{

if (EndState == 0) return 0;

else return 1;

}

//конец работы

void EndExec()

{

EndState = 1;

}

//получения события

void GetEvent(TEvent& event)

{

string OpInt = "m+-s?q";//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

case '-':

event.command = cmDel; break;//удалить объект

case 's':

event.command = cmShow; break;//вывести список

case 'z':

event.command = cmGet; break;//вывести атрибуты 1 объекта

case 'q':

event.command = cmQuit; break;//выход

}

//выделяем параметры команды если они есть

if (s.length() > 1)

{

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

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

event.a = A;//записываем параметр в сообщение

}

}

else event.what = evNothing;//пустое событие

}

//обработчик событий

void HandleEvent(TEvent& event)

{

if (event.what == evMessage)

{

switch (event.command)

{

case cmMake:

size = event.a;

beg = new object \*[size];

cur = 0;

ClearEvent(event);

break;

case cmAdd:

Add();

ClearEvent(event);

break;

case cmDel:

Del();

ClearEvent(event);

break;

case cmShow:

Info\_All();

ClearEvent(event);

break;

case cmGet:

Info\_title();

ClearEvent(event);

break;

case cmQuit:

EndExec();

ClearEvent(event);

break;

default:

HandleEvent(event);

}

}

}

//главный цикл обработки событий

int Execute()

{

TEvent event;

do

{

EndState = 0;

GetEvent(event);

HandleEvent(event);

} while (!Valid());

return EndState;

}

};

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Команды:\n+: добавить\n-: удалить\nm: создать лист\ns: Вывод элементов: \nz: Вывод названий элементов\nq: выход\n\n";

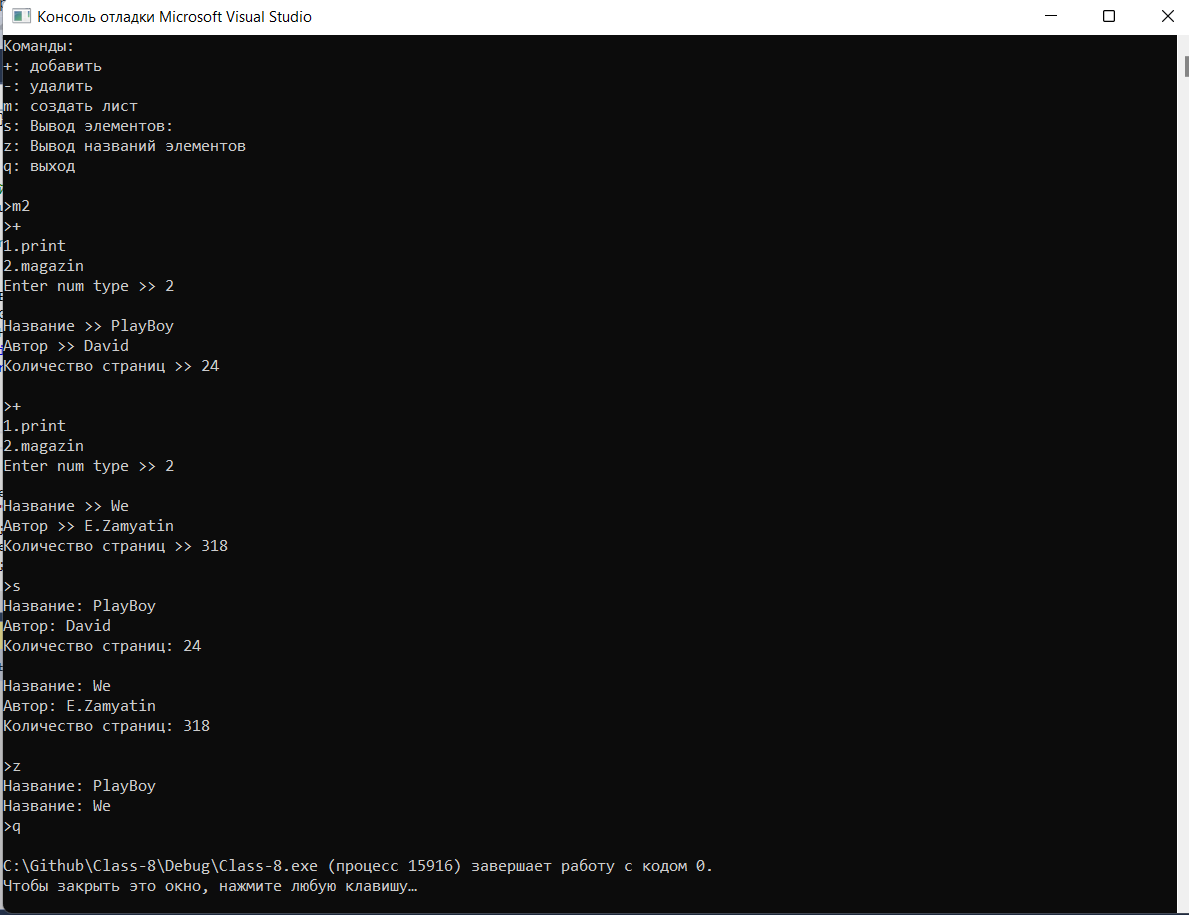
Dialog D;

D.Execute();

return 0;

}

Скриншоты результатов:



Анализ результатов:

Программа работает корректно. Создается список, вносятся элементы, выводится вся информация об элементах, информация только о названиях, корректный выход из программы.