**Лабораторная работа №5**

*Программная реализация отношения простого наследования*

**Цель работы:**

Изучить способы создания производного класса и особенности работы с ним, правила инициализации и доступа к элементам производного класса; приобрести практические навыки наследования.

**Задание:**

Создайте производный класс для АТД, реализованного по заданию лабораторной работы 4, используя одиночное наследование.

Проверьте работоспособность АТД на тестовом наборе данных.

#include <string.h>

#include <iostream>

using namespace std;

template <class T1 = int, class T2 = float>

class MemoryDevice {

public:

MemoryDevice(const char\* programa, T1 v, T2 p);

MemoryDevice<T1, T2>();

void print();

MemoryDevice<T1, T2>(const MemoryDevice& a);

bool operator==(const MemoryDevice& a);

MemoryDevice<T1, T2> operator+(const MemoryDevice& a);

MemoryDevice<T1, T2>& operator++();

MemoryDevice<T1, T2>& operator=(const MemoryDevice& a);

~MemoryDevice<T1, T2>();

protected:

char\* programm; T1 cost; T2 size;

};

template <class T1, class T2>

MemoryDevice<T1, T2>::MemoryDevice<T1, T2>() {

programm = new char[strlen("EVA 255") + 1];

strcpy(programm, "EVA 255");

cost = 200;

size = 300;

};

template <class T1, class T2>

MemoryDevice<T1, T2>::MemoryDevice<T1, T2>(const char\* programa, T1 v, T2 p) {

programm = new char[strlen(programa) + 1];

strcpy(programm, programa);

cost = v;

size = p;

};

template <class T1, class T2>

void MemoryDevice<T1, T2>::print() {

cout << "Название: " << programm << endl;

cout << "Стоимость: " << cost << endl;

cout << "Размер: " << size << endl;

};

template <class T1, class T2>

MemoryDevice<T1, T2>::MemoryDevice<T1, T2>(const MemoryDevice& a) {

programm = new char[strlen(a.programm) + 1];

strcpy(programm, a.programm);

cost = a.cost;

size = a.size;

};

template <class T1, class T2>

bool MemoryDevice<T1, T2>::operator==(const MemoryDevice& a) {

return ((!strcmp(programm, a.programm)) &&

cost == a.cost &&

size == a.size);

}

template <class T1, class T2>

MemoryDevice<T1, T2>::~MemoryDevice<T1, T2>() {

delete[] programm;

};

template <class T1, class T2>

MemoryDevice<T1, T2> MemoryDevice<T1, T2>::operator+(const MemoryDevice & a) {

MemoryDevice <T1, T2> v;

v.programm = new char[strlen(a.programm) + strlen(programm) + 2];

strcpy(v.programm, programm);

strcat(v.programm, ",");

strcat(v.programm, a.programm);

v.cost = cost + a.cost;

v.size = size + a.size;

return v;

};

template <class T1, class T2>

MemoryDevice<T1, T2>& MemoryDevice<T1, T2>:: operator=(const MemoryDevice & a) {

programm = new char[strlen(a.programm) + 1];

strcpy(programm, a.programm);

cost = a.cost;

size = a.size;

return(\*this);

};

template <class T1, class T2>

MemoryDevice<T1, T2>& MemoryDevice<T1, T2>::operator++() {

cost += 1;

size += 1;

return(\*this);

};

//Производный класс

template <class T1 = int, class T2 = float>

class korpys : public MemoryDevice<T1, T2> {

public:

korpys<T1, T2>();

korpys(T1 seria, bool сooling, const char\* inf, const char\* programa, T1 v, T2 p);

~korpys<T1, T2>();

void print1();

korpys<T1, T2>(const korpys& a);

korpys& operator = (const korpys<T1, T2>& a);

protected:

T1 seria; bool cooling; char\* inf;

};

template <class T1, class T2>

korpys<T1, T2>::korpys<T1, T2>() : MemoryDevice<T1, T2>() {

seria = 0;

cooling = false;

inf = new char[strlen("нету") + 1];

strcpy(inf, "нету");

};

template <class T1, class T2>

korpys<T1, T2>::korpys<T1, T2>(T1 seria, bool cooling, const char\* inf, const char\* programa, T1 v, T2 p) : MemoryDevice<T1, T2>(programa, v, p) {

this->seria = seria;

this->cooling = cooling;

this->inf = new char[strlen(inf) + 1];

strcpy(this->inf, inf);

};

template <class T1, class T2>

void korpys<T1, T2>::print1() {

cout << "Вывод " << endl;

korpys<T1, T2>::print();

cout << endl << "Серия-" << seria << endl;

cout << "Охлаждение-";

if (cooling) cout << "Есть" << endl;

else cout << "Нет" << endl;

cout << "Информация - " << inf << endl;

};

template <class T1, class T2>

korpys<T1, T2>::korpys<T1, T2>(const korpys& a) {

inf = new char[strlen(a.inf) + 1];

strcpy(inf, a.inf);

seria = a.seria;

cooling = a.cooling;

this->programm = new char[strlen(a.programm) + 1];

strcpy(this->programm, a.programm);

this->cost = a.cost;

this->size = a.size;

};

template <class T1, class T2>

korpys<T1, T2>& korpys<T1, T2>::operator=(const korpys& a) {

inf = new char[strlen(a.inf) + 1];

strcpy(inf, a.inf);

seria = a.seria;

cooling = a.cooling;

this->programm = new char[strlen(a.programm) + 1];

strcpy(this->programm, a.programm);

this->cost = a.cost;

this->size = a.size;

return(\*this);

};

template <class T1, class T2>

korpys<T1, T2>::~korpys() {

delete[] inf;

};

int main()

{

setlocale(LC\_ALL, "Russian");

char\* programa = new char[10];

char\* inf = new char[10];

double cost, size, seria;

bool cooling;

cout << "Введите название запоминающего устройства ";

cin >> programa;

cout << endl << "Введите стоимость запоминающего устройства ";

cin >> cost;

cout << endl << "Введите размер запоминающего устройства ";

cin >> size;

cout << endl << "Базовый класс " << endl;

MemoryDevice<float, int>a(programa, cost, size);

a.print();

cout << endl << "Производный класс " << endl;

cout << endl << "Конструктор с значениями " << endl;

cout << "Введите информацию о запоминающем устройстве:" << endl;

cout << endl << "Серия запоминающего устройства ";

cin >> seria;

cout << endl << "Наличие охлаждения ";

cin >> cooling;

cout << endl << "Информация о запоминающем устройстве ";

cin >> inf;

korpys<float, int> ml(seria, cooling, inf, programa, cost, size);

ml.print1();

cout << endl << "Конструктор без значений " << endl;

korpys<float, int> mlv;

mlv.print1();

cout << endl << "Копирующий конструктор " << endl;

korpys<float, int> cp(ml);

cp.print1();

cout << endl << "Оператор = " << endl;

mlv = ml;

mlv.print1();

cout << endl << "Тестирование функций базового класса через производный" << endl;

cout << endl << "1 оператор ++ и вывод " << endl;

++mlv;

mlv.print();

cout << endl << "2 оператор ==" << endl;

if (mlv == ml)cout << "Равны" << endl;

else cout << "Не равны" << endl;

cout << endl << "3 оператор +" << endl;

a = cp + mlv;

a.print();

return 0;

}

## Контрольные вопросы

**1. Дайте определение наследования**

Наследование – это механизм получения нового класса на основе существующего класса. Существующий класс может быть дополнен или изменен для создания производного класса. Наследование осуществляется с помощью конструкции:

class имя\_класса:

public|protected|private)opt имя\_базового\_класса

{

объявления членов

};

**2. Какие модификаторы прав доступа к производному классу вы знаете и в чем их особенности?**

Модификаторы прав доступа к членам класса могут применяться в объявлении класса в любом порядке и сколько угодно раз. Открытый член доступен во всей области видимости, где виден класс. Закрытый член доступен другим функциям-членам своего класса. Защищенный член доступен не только другим функциям-членам своего класса, но и функциям-членам класса, унаследованного непосредственно от данного класса. Если все члены открыты, то ключевое слово class можно заменить ключевым словом struct.

**3. Как выполняется конструктор при наследовании?**

При наследовании и инициализации членов класса конструкторы выполняются в следующем порядке:

1. Базовые классы инициализируются в порядке объявления.

2. Члены инициализируются в порядке объявления.

**4. Дайте определение чисто виртуальной функции.**

Чисто виртуальная функция – это виртуальная функция-член, тело которой не определено. Она объявляется внутри класса:

virtual прототип\_функции = 0;

**5. Какие операторы используются для разыменования указателя на член класса?**

Для разыменования указателя на член класса используются два оператора: .\* и →\*.

Сначала разыменовывается указатель для получения переменной-члена, а затем происходит доступ к члену объекта.

**Результат работы программы:**



