**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.

**Код программы**

#include "pch.h"

#include <iostream>

#include <string.h>

using namespace std;

template <class T = int, class F = float> //шаблонный БК

class send

{

public:

send<T, F>(const char\* c, T b, F d, const char\* y);

send<T, F>(const send<T, F> &a);

send<T, F>();

~send<T, F>();

void print();

bool operator==(send<T, F> a);

send operator+(send<T, F> &a);

send &operator=(const send<T, F> &a);

protected:

char\* type;

T weight;

F price;

char\* marka;

};

class send2 { // БК без шаблона

public:

send2(const char\* c, int b, double d, const char\* y);

send2();

~send2();

void print();

protected:

char\* type;

int weight;

double price;

char\* marka;

};

template <class T, class F> // реализация шаблонного БК

send<T, F>::send() {

type = new char[10];

strcpy(type, "-");

weight = 0;

price = 0;

marka = new char[10];

strcpy(marka, "-");

}

template <class T, class F>

send<T, F>::send(const send<T, F> &a) {

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

strcpy(type, a.type);

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

strcpy(marka, a.marka);

weight = a.weight;

price = a.price;

}

template <class T, class F>

send<T, F>::send(const char\* c, T b, F d, const char\* y)

{

type = new char[strlen(c) + 1];

strcpy(type, c);

weight = b;

price = d;

marka = new char[strlen(y) + 1];

strcpy(marka, y);

}

template <class T, class F>

send<T, F>::~send() {

type = NULL; marka = NULL;

}

template <class T, class F>

void send<T, F>::print()

{

cout << "Тип:" << type << endl;

cout << "Вес:" << weight << endl;

cout << "Стоимость:" << price << endl;

cout << "Марка:" << marka << endl;

}

template <class T, class F>

bool send<T, F>::operator==(send<T, F> a) {

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

weight == a.weight && price == a.price &&

(!strcmp(type, a.type)));

}

template <class T, class F>

send<T, F> send<T, F>::operator+(send<T, F> &a) {

send s;

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

strcpy(s.type, type);

strcat(s.type, a.type);

s.weight = weight + a.weight;

s.price = price + a.price;

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

strcpy(s.marka, marka);

strcat(s.marka, a.marka);

return s;

}

template <class T, class F>

send<T, F> &send<T, F>::operator=(const send<T, F> &a)

{

type = NULL;

type = a.type;

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

strcpy(type, a.type);

weight = a.weight;

price = a.price;

marka = NULL;

marka = a.marka;

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

strcpy(marka, a.marka);

return (\*this);

}

// реализация БК без шаблона

send2::send2() {

type = new char[10];

strcpy(type, "-");

weight = 0;

price = 0;

marka = new char[10];

strcpy(marka, "-");

}

send2::send2(const char\* c, int b, double d, const char\* y)

{

type = new char[strlen(c) + 1];

strcpy(type, c);

weight = b;

price = d;

marka = new char[strlen(y) + 1];

strcpy(marka, y);

}

send2::~send2() {

type = NULL; marka = NULL;

}

void send2::print()

{

cout << "Тип:" << type << endl;

cout << "Вес:" << weight << endl;

cout << "Стоимость:" << price << endl;

cout << "Марка:" << marka << endl;

}

// ПК

template <class T, class F>

class sendrate : public send<double, int > {

public:

sendrate<T,F>(const char\* c, T b, F d, const char\* y, int r);

sendrate<T, F>(const sendrate<T, F> &a);

sendrate<T, F>();

~sendrate<T, F>();

void print();

sendrate& operator = (const sendrate<T, F> &a);

protected:

int R;

};

template <class T, class F>

sendrate<T, F>::sendrate(const char\* c, T b, F d, const char\* y, int r) :send<T, F>(c, b, d, y)

{

R = r;

}

template <class T, class F>

sendrate<T, F>::sendrate(const sendrate<T, F> &a) {

R = a.R;

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

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

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

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

this->weight = a.weight;

this->price = a.price;

}

template <class T, class F>

sendrate<T, F>& sendrate<T, F>::operator=(const sendrate &a) {

R = a.R;

this->price = a.price;

this->weight = a.weight;

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

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

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

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

return(\*this);

};

template <class T, class F>

sendrate<T, F>::sendrate() : send<T,F>() {

R = 0;

}

template <class T, class F>

void sendrate<T, F>::print()

{

send::print();

cout << "Рейтинг:" << R << endl;

}

template <class T, class F>

sendrate<T, F>::~sendrate() {

send<T, F>::~send();

}

template <class TT> //шаблонный ПК

class sendBC :public send2 {

public:

sendBC<TT>(const char\* c, int b, double d, const char\* y, TT r);

sendBC<TT>();

sendBC<TT>(const sendBC<TT> &a);

void print();

~sendBC<TT>();

protected:

int R;

};

template <class TT>

sendBC<TT>::sendBC() : send2()

{

R = 0;

}

template <class TT>

sendBC<TT>::sendBC(const char\* c, int b, double d, const char\* y, TT r):send2(c,b,d,y) {

R = r;

}

template <class TT>

sendBC<TT>::sendBC(const sendBC<TT> &a) {

R = a.R;

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

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

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

this->weight = a.weight;

this->price = a.price;

}

template <class TT>

void sendBC<TT>::print()

{

send2::print();

cout << "Рейтинг:" << R << endl;

}

template <class TT>

sendBC<TT>::~sendBC() {

send2::~send2();

}

int main()

{

setlocale(LC\_ALL, "Russian");

char\* ty = new char[10];

char\* ty2 = new char[10];

char\* tyBC = new char[10];

int w, w2, r, r2, wBC ;

double f, f2, fBC;

char\* mark = new char[10];

char\* mark2 = new char[10];

char\* markBC = new char[10];

cout << "Базовый шаблонный класс:\n";

cout << "Введите тип модема: ";

cin >> tyBC;

cout << endl << "Введите вес: ";

cin >> wBC;

cout << endl << "Введите стоимость(руб.): ";

cin >> fBC;

cout << endl << "Введите марку модема: ";

cin >> markBC;

cout << endl;

send<double, int> e(tyBC, wBC, fBC, markBC);

e.print();

cout << '\t' << "Базовый шаблонный класс через производный" << endl;

cout << "Введите тип модема: ";

cin >> ty;

cout << endl << "Введите вес: ";

cin >> w;

cout << endl << "Введите стоимость(руб.): ";

cin >> f;

cout << endl << "Введите марку модема: ";

cin >> mark;

cout << endl << "Введите рейтинг модема: ";

cin >> r;

cout << endl;

sendrate<double, int> a(ty, w, f, mark, r); //создание объекта a производного класса sendrate

a.print();// вывод a

sendrate<double, int> b(a); // копирование объекта b производного класса sendrate

b.print(); // вывод b

if (a == b)

cout << "Устройства одинаковые\n";

else cout << "Устройства разные\n";

cout << "Операция =\n";

e = b;

e.print();

cout << endl;

cout << "сложение" << endl;

e = a + b;

e.print();

cout << endl;

cout << '\t' << "Базовый нешаблонный класс через производный шаблонный" << endl;

cout << "Введите тип модема: ";

cin >> ty2;

cout << endl << "Введите вес: ";

cin >> w2;

cout << endl << "Введите стоимость(руб.): ";

cin >> f2;

cout << endl << "Введите марку модема: ";

cin >> mark2;

cout << endl << "Введите рейтинг модема: ";

cin >> r2;

sendBC<char> c(ty2, w2, f2, mark2, r2); // создание объекта c производного класса sendBC

c.print();

cout << endl;

sendBC<char> d(c); // копирование объекта c производного класса sendBC

d.print();

cout << endl;

return 0;

}

## Контрольные вопросы

1. Дайте определение наследования

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

3. Как выполняется конструктор при наследовании?

4. Дайте определение чисто виртуальной функции.

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

**Ответы**

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

virtual прототип\_функции = 0;

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

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

**Тестовые данные**





