**Новосибирский государственный технический университет**

Лабораторная работа №4 по дисциплине **программирование**

Тема: Наследование.

|  |  |
| --- | --- |
| Группа: АВТ-907  Студент: Смоленский И.В. | Преподаватель: Булатов А. Д. |

Новосибирск, 2020

**Задание**:

Вариант 14.

Класс для pаботы с рациональными дробями. Создать наследников класса:

•класс, в котором хранится поле - результат вычисления дроби;

• класс смешанная дробь.

**Теоретические сведения:**

## Простое наследование

Простое наследование описывает родство между двумя классами: один из которых наследует второму. Класс, находящийся на вершине иерархии, называется базовым классом. Прочие классы называются производными классами. Из одного класса могут выводится многие классы, но даже в этом случае подобный вид взаимосвязи остается простым наследованием.

Базовый класс часто называют предком, а производный класс - потомком.

Производный класс наследует из базового данные-члены и функции-члены, но не конструкторы и деструкторы.

class Tbase { //Базовый класс

private:

int count;

public:

TBase() {count = 0;}

void SetCount(int n) {count = n;}

int GetCount(void) {return count;}

};

class TDerived: public TBase { //Производный класс

public:

TDerived(): Tbase() {}

void ChangeCount(int n) {SetCount(GetCount() + n);}

};

Производный класс назван TDerived. За именем следует двоеточие и одно из ключевых слов - **public, protected, private**. После этих элементов имя базового класса TBase.

В иерархии классов соглашение относительно доступности компонентов класса следующее:

***private*** *–* член класса может использоваться только функциями – членами данного класса и функциями – “друзьями” своего класса. В производном классе он недоступен.

***protected*** – то же, что и ***private,*** но дополнительно член класса с данным атрибутом доступа может использоваться функциями-членами и функциями – “друзьями” классов, производных от данного.

***public*** – член класса может использоваться любой функцией, которая является членом данного или производного класса, а также к ***public*** - членам возможен доступ извне через имя объекта.

Обычно в производном классе есть конструктор, если он был в базовом. Он должен вызывать конструктор базового класса:

TDerived(): TBase() {}

Кроме этого в нем могут выполняться и другие действия, например:

TDerived():TBase() {cout <<" I am being initialized" <<endl;}

Можно реализовать объявленный конструктор отдельно:

TDerived::TDerived() : TBase()

{ // операторы

}

Объекты класса конструируются снизу вверх: сначала базовый, потом компоненты-объекты (если они имеются), а потом сам производный класс. Таким образом, объект производного класса содержит в качестве подобъекта объект базового класса.

Уничтожаются объекты в обратном порядке: сначала производный, потом его компоненты-объекты, а потом базовый объект.

Таким образом, порядок уничтожения объекта противоположен по отношению к порядку его конструирования.

Производный класс наследует count из базового класса, но так как он закрыт в TBase доступ к нему возможен только через функции-члены класса TBase. В производном классе можно определить функцию, которая будет обращаться к наследованным функциям-членам класса TBase:

void ChangeCount(int n) {SetCount(GetCount() +n);}

В производном классе деструктор нужен только в том случае, если необходимо что-то удалять.

#include <iostream.h>

#include <string.h>

class TBase { //Базовый класс

private:

char \*basep;

public:

TBase(const char \*s) {basep = strdup(s); }

~TBase() {delete basep;}

const char \*GetStr(void) { return basep; }

};

class TDerived: public TBase { //Производный класс

private:

char \*uppercasep;

public:

TDerived(const char\* s): TBase(s) { uppercasep = strupr(strdup(s));}

~TDerived() {delete uppercasep;}

const char \*GetUStr(void) { return uppercasep;}

};

void main() {

TBase president("George Washington");

cout << "Original string: " << president.GetStr() << endl;

TDerived pres("George Washington");

cout << "Uppercase string:" << pres.GetUStr() << endl;

}

В производном классе обычно добавляются новые данные и функции-члены. Однако существует возможность замещения функций-членов, наследованных от базового класса.

#include <iostream.h>

#include <string.h>

class TBase { //Базовый класс

private:

char \*basep;

public:

TBase(const char \*s) {basep = strdup(s); }

~TBase() {delete basep;}

void Display(void) { cout << basep << endl;}

};

class TState: public TBase { //Производный класс

public:

TState(const char \*s):TBase (s) {}

void Display(void); // замещающая функция

};

void TState::Display(void) {

cout << "State: "; //Новый оператор

TBase::Display(); //Вызов замещенной функции

}

void main() {

TState object("Welcome to Borland C++5 programming!");

object.Display();

}

## 

## Множественное наследование

Построение производного класса на основе нескольких базовых выглядит очень просто: вместо имени одного базового класса (вместе с его атрибутом) используется список имен, разделенный запятыми, например:

class A { /\*…\*/ };

class B { /\*…\*/ }

class C: public A, private B { /\*…\*/ }

Передача аргументов конструкторам базовых классов из конструктора производного класса производится так же как и раньше:

С::С( int a, char\* str) : A(a), B(str) { /\*…\*/ };

**Код:**

**Реализация класса наследника BankNameAddress:**

#include <iostream>

#include <string>

#include "stdio.h"

#include "Bankomat.h"

#include "Bank\_nameAddress.h"

using namespace std;

Bank\_name\_address::Bank\_name\_address(char\* name, char\* address)

{

bankName = name;

bankAddress = address;

}

Bank\_name\_address::Bank\_name\_address(char\* id1, int sumBank1, int sumMaxGet1)

{

id = id1;

sumBank = sumBank1;

sumMaxGet = sumMaxGet1;

}

Bank\_name\_address::~Bank\_name\_address()

{

delete[] bankName;

delete[] bankAddress;

delete[] id;

}

char\* Bank\_name\_address::toString()

{

char\* str = new char[100];

sprintf\_s(str, 100, "Name = %s, Address = %s", this->bankName, this->bankAddress);

return str;

}

**Реализация класса наследника Bank\_withHistory:**

#include <sstream>

#include <iostream>

#include <algorithm>

#include <exception>

#include "Bankomat.h"

#include "Bank\_transition+Bank\_withHistory.h"

#include <ctime>

#include "stdio.h"

Bank\_withHistory::Bank\_withHistory(char\* id1, unsigned sumBank1, unsigned sumMaxGet1)

{

id = id1;

sumBank = sumBank1;

sumMaxGet = sumMaxGet1;

}

Bank\_transition::Bank\_transition()

{

time = std::time(0);

income = 1000;

remain = 10000;

get = true;

}

Bank\_transition::Bank\_transition(unsigned income1, unsigned remain1, bool get1)

{

time = std::time(0);

income = income1;

remain = remain1;

get = get1;

}

Bank\_transition::~Bank\_transition()

{}

Bank\_withHistory::~Bank\_withHistory()

{

delete[] arr;

}

unsigned Bank\_withHistory::loadMoney(unsigned getting)

{

try

{

if (getting < 0)

{

throw std::invalid\_argument("Invalid arg");

}

}

catch (const std::invalid\_argument & err)

{

std::cerr << " Invalid argument: " << err.what() << std::endl;

}

unsigned save = Bankomat::loadMoney(getting);

std::cout << " It's child function " << std::endl;

arr[flag] = Bank\_transition(getting, sumBank, true);

flag = (flag + 1) % 10;

if (capacity < maxSize)

{

capacity++;

}

return save;

}

unsigned Bank\_withHistory::takeMoney(unsigned silver)

{

try // Try catch block works properly

{

if (silver < 0)

{

throw std::invalid\_argument("Invalid arg");

}

}

catch (const std::invalid\_argument & err)

{

std::cerr << " Invalid argument: " << err.what() << std::endl;

}

unsigned save = Bankomat::takeMoney(silver);

std::cout << " It's child function " << std::endl;

arr[flag] = Bank\_transition(silver, sumBank, false);

flag = (flag + 1) % 10;

if (capacity < maxSize)

{

capacity++;

}

return save;

}

char\* Bank\_withHistory::toString()

{

std::stringstream ss;

if (capacity == maxSize)

{

int index = flag;

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

{

ss << arr[index];

index = (index + 1) % 10;

}

}

else

{

int index = flag;

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

{

ss << arr[i] << std::endl;

}

}

return 0;

}

std::ostream& operator<<(std::ostream& os, const Bank\_transition& p)

{

char buff[20];

strftime(buff, 20, "%d/%m/%Y:%H:%M:%S", localtime(&p.time));

std::string s(buff);

os << "Time of transition: " << s << "\nsum of taking/loading " << p.income << "\nsum of remaining: " << p.remain << std::endl;

return os;

}  
  
  
  
**Вывод**:   
Мной были разработаны наследники класса Банкомат, которые имеют все свойства родительского класса и имеют что-то своё.