Лабораторная работа №2

Классы и объекты. Использование конструкторов.

1. Цель задания:

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2) Создание объектов с использованием конструкторов.

2. Теоретические сведения

2.1. Три типа конструкторов

В каждом классе должен быть хотя бы один метод, который предназначен для инициализации объекта. Его имя совпадает с именем класса, и он вызывается автоматически при инициализации объекта. Автоматический вызов конструктора позволяет избежать ошибок, связанных с использованием неинициализированных переменных.
Существует три типа конструкторов:

- конструктор с параметрами, используется для инициализации объекта требуемыми значениями;
- конструктор без параметров, используется для создания «пустого» объекта;
- конструктор копирования, используется для создания объекта, аналогичного тому, который уже существует.

Конструктор без параметров и конструктор копирования создаются по умолчанию.

2.2 Конструктор с параметрами

Конструктор с параметрами, используется для инициализации объекта требуемыми значениями;
Пример:

```
class Person
{
string name;//динамическая строка
int age;
public:
Person(string Name, int Age)//конструктор с параметрами
{
     name=Name
     age=Age;
}
void Set_Person(string Name, int age)
{
...
}
void Print_Person()
{
...

}
...
};
void main()
{
```

```
Person P(”Иванов”,21);
P.Print_Person();
}
```

2.3. Конструктор без параметров

Конструктор без параметров используется для создания «пустого» объекта.

Пример:

```
Person::Person()
{
//пустой объект
name=””;//пустая строка
age=0;
}
```

или

```
Person::Person()
{
//объект с полями
       name=”Ivanov”;//конкретное значение строки
       age=25;
}
```

Конструктор без параметров вызывается следующим образом:

```
Person p;
```

2.4. Конструктор копирования

Конструктор копирования — это специальный вид конструктора, получающий в качестве единственного параметра указатель на объект этого же класса:

```
T::T(const T&) { ... /* Тело конструктора */ } ,
```

где T — имя класса

Пример:

```
Person::Person(const Person&p)
{
name=p.name;
age=p.age;
}
```

Этот конструктор вызывается в тех случаях, когда новый объект создается путем копирования существующего:

- при описании нового объекта с инициализацией другим объектом;
- при передаче объекта в функцию по значению;
- при возврате объекта из функции.

Вызовы конструктора копирования:

```
void print(person a)//объект передается по значению
{
     a.print_person();
}
person set_inf()
{
     person s;//создается временный объект
     s.set_person("Sidorov",5,2);
     return s;//объект возвращается как значение функции
}
void main()
{
```

```
person a;//конструктор без параметров
person b("Ivanov",5,3);//конструктор с параметрами
person c=b;//конструктор копирования

a.print_person();
b.print_person();
c.print_person();

//передача объекта в функцию по значению
print(b);//конструктор копирования
//объект возвращается как результат функции
person d=set_inf();//конструктор копирования
d.print_person();
}
```

Если программист не указал ни одного конструктора копирования, компилятор создает его автоматически. Такой конструктор выполняет поэлементное копирование полей. Если класс содержит указатели или ссылки, это, скорее всего, будетнеправильным, поскольку и копия, и оригинал будут указывать на одну и ту же область памяти.

2.5. Основные свойства конструкторов.

- Конструктор не возвращает значение, даже типа void. Нельзя получить указатель на конструктор.
- Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации (при этом используется механизм перегрузки).
- Конструктор, вызываемый без параметров, называется конструктором по умолчанию.
- Параметры конструктора могут иметь любой тип, кроме этого же класса. Можно задавать значения параметров по умолчанию, но их может содержать только один из конструкторов.
- Если программист не указал ни одного конструктора, компилятор создает его автоматически. Такой конструктор вызывает конструкторы по умолчанию для полей класса. В случае, когда класс содержит константы или ссылки, при попытке создания объекта класса будет выдана ошибка, поскольку их необходимо инициализировать конкретными значениями, а конструктор по умолчанию этого делать не умеет.
- Конструкторы не наследуются.
- Конструкторы нельзя описывать с модификаторами const, virtual и static.
- Конструкторы глобальных объектов вызываются до вызова функции main. Локальные объекты создаются, как только становится активной область их действия. Конструктор запускается и при создании временного объекта (на пример, при передаче объекта из функции).
- Конструктор вызывается, если в программе встретилась какая-либо из синтаксических конструкций:

```
имя_класса имя_объекта [(список параметров)];// Список параметров не должен быть пустым
имя класса (список параметров);// Создается объект без имени (список может быть пустым)
имя_класса имя_объекта = выражение;// Создается объект без имени и копируется
```

Примеры:

```
Person p1(”Ivanov”,23);//конструктор с параметрами
Person p2();//конструктор без параметров
Person p3=p1;// конструктор копирования
```

```
Person p4=Person (”Sidorov”,20);//создается объект без имени и копируется
Person* pp1=new(Person);//указатель на пустой объект
Person*pp2=new Person(”Petrov”,32);//указатель на объект
```

Существует еще один *способ инициализации полей в конструкторе* — с помощью списка инициализаторов, расположенных после двоеточия между заголовком и телом конструктора:

```
Person;:Person(int Age):age(Age)
{
name=new char[8];
strcpy(name,”Ivanov”);
}
```

Поля перечисляются через запятую. Для каждого поля в скобках указывается инициализирующее значение, которое может быть выражением. Без этого способа не обойтись при инициализации *полей-констант, полей-ссылок* и *полей-объектов.* В последнем случае будет вызван конструктор, соответствующий указанным в скобках параметрам.

2.6. Деструктор

Деструктор – это особый вид метода, применяющийся для освобождения ресурсов, выделенных конструктором объекту. Деструктор вызывается автоматически, когда объект удаляется из памяти:

- для локальных объектов это происходит при выходе из блока, в котором они объявлены;
- для глобальных — как часть процедуры выхода из main;
- для объектов, заданных через указатели, деструктор вызывается неявно при использовании операции delete.

Имя деструктора начинается с тильды (~), непосредственно за которой следует имя класса. Свойства деструктора:

- не имеет аргументов и возвращаемого значения;
- не наследуется;
- не может быть объявлен как const или static (далее);
- может быть виртуальным (далее).

Если деструктор явным образом не определен, компилятор автоматически создает пустой деструктор.

Описывать в классе деструктор явным образом требуется в случае, когда объект содержит указатели на память, выделяемую динамически — иначе при уничтожении объекта память, на которую ссылались его поля-указатели, не будет помечена как свободная. Указатель на деструктор определить нельзя.

Деструктор для класса Person, в котором поле name реализуется как динамическая строка будет выглядеть так:

```
Person::~Person() {delete [] name;}
```

Без необходимости явно вызывать деструктор объекта не рекомендуется.

2.7. Определение методов класса.

Методы класса имеют неограниченный доступ ко всем элементам класса, независимо от спецификаторов доступа и порядка объявления методов в классе. Методы могут определяться как в классе, так и вне его. Определение метода внутри класса ничем не отличается от определения обычной функции. По умолчанию такой метод считается встроенной функцией (inline). Если метод определяется вне функции, то принадлежность метода классу указывается с помощью имени класса: Имя_класса::Имя_метода. В классе присутствует только прототип. Методы могут быть перегружены, могут принимать

аргументы по умолчанию. Метод (кроме статических методов) неявно получает в качестве аргумента указатель на тот объект, для которого он вызван. Этот указатель обозначается ключевым словом this и может быть использован в теле метода. В явном виде этот указатель применяется в основном для возвращения из метода указателя (return this;) или ссылки (return *this;) на вызвавший объект.

Пример:

```
Person&Old(Person&P)
{
if(P.GetAge()>60) return *this;
}
```

Запись *this представляет собой значение текущего объекта.

Методы могут быть перегружены.

Методы могут быть константными, т. е. не изменять значение полей класса. Константный метод обозначается с помощью слова const после списка аргументов метода. Для объекта-константы может быть вызван только константный метод. Для объекта-переменной может быть вызван и константный и неконстантный методы.

3. Постановка задачи

1. Определить пользовательский класс.
2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.
3. Определить в классе деструктор.
4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).
5. Написать демонстрационную программу, в которой продемонстрировать все три случая вызова конструктора-копирования, вызов конструктора с параметрами и конструктора без параметров.

4. Ход работы

Задача

Реализовать пользовательский класс TOBAP
наименование– string
количество – int
стоимость – float

1. Создадим пустой проект. Для этого требуется:
   1.1. Запустить MS Visual Studio:
   1.2. Выбрать команду File/New/Project
   1.3. В окне New Project выбрать Win Console 32 Application, в поле Name указать имя проекта (Lab2), в поле Location указать место положения проекта (личную папку), нажать кнопку Ok.
   1.4. В следующем окне выбрать кнопку Next.
   1.5. В диалоговом окне Additional Settings установить флажок Empty (Пустой проект) и нажать кнопку Finish.
   1.6. В результате выполненных действий получим пустой проект.
2. Добавим в проект файл Tovar.h, содержащий описание класса TOBAP. Для этого нужно:
   2.1. Вызвать контекстное меню проекта в панели Обозреватель решений (Solution Explorer), выбрать в нем пункт меню Add/ New Item.
   2.2. В диалоговом окне Add New Item – Lab2 выбрать Категорию Code, шаблон – Header File (.h), задать имя файла Tovar. В результате выполненных действий

получим пустой файл Tovar.h, в котором будет редактироваться текст программы.

2.3. Ввести следующий текст программы

//описание класса

```
#include <iostream>
#include <string>
using namespace std;
class Tovar
{
	//атрибуты
	string naimenovanie;
	int kolichestvo;
	double stoimost;
public:
	Tovar();//конструктор без параметров
	Tovar(string, int, double);//конструктор с параметрами
	Tovar(const Tovar&);//конструктор копирования
	~Tovar();//деструктор
	string get_naimenovanie();//селектор
	void set_naimenovanie(string);//модификатор
	int get_kolichestvo();//селектор
	void set_kolichestvo(int); //модификатор
	double get_stoimost();//селектор
	void set_stoimost(double); //модификатор
	void show();//просмотр атрибутов
};
```

3. Добавим в проект файл Tovar.cpp, содержащий описание методов класса ТОВАР. Для этого нужно:

3.1. Вызвать контекстное меню проекта в панели Обозреватель решений (Solution Explorer), выбрать в нем пункт меню Add/ New Item.

3.2. В диалоговом окне Add New Item – Lab2 выбрать Категорию Code, шаблон – C++File (.cpp), задать имя файла Tovar. В результате выполненных действий получим пустой файл Tovar.cpp, в котором будет редактироваться текст программы.

3.3. Ввести следующий текст программы:

```
#include "Tovar.h"
#include <iostream>
#include <string>
using namespace std;
//конструктор без параметров
Tovar::Tovar()
{
	naimenovanie="";
	kolichestvo=0;
	stoimost=0;
	cout<<"Constructor bez parametrov dlia objecta"<<this<<endl;
}
//конструктор с параметрами
Tovar::Tovar(string N, int K, double S)
{
	naimenovanie=N;
	kolichestvo=K;
	stoimost=S;
	cout<<"Constructor s parametrami dlia objecta"<<this<<endl;
}
//конструктор копирования
Tovar::Tovar(const Tovar &t)
{
	naimenovanie=t.naimenovanie;
	kolichestvo=t.kolichestvo;
	stoimost=t.stoimost;
```

```
	cout<<"Constructor copirovania dlia objecta"<<this<<endl;
}
//деструктор
Tovar::~Tovar()
{
	cout<<"Destructor dlia objecta"<<this<<endl;
}
//селекторы
string Tovar::get_naimenovanie()
{
	return naimenovanie;
}
int Tovar::get_kolichestvo()
{
	return kolichestvo;
}
double Tovar::get_stoimost()
{
	return stoimost;
}
//модификаторы
void Tovar::set_naimenovanie(string N)
{
	naimenovanie=N;
}
void Tovar::set_kolichestvo(int K)
{
	kolichestvo=K;
}
void Tovar::set_stoimost(double S)
{
	stoimost=S;
}
//метод для просмотра атрибутов
void Tovar::show()
{
	cout<<"naimenovanie :"<<naimenovanie<<endl;
	cout<<"kolichestvo :"<<kolichestvo<<endl;
	cout<<"stoimost :"<<stoimost<<endl;
}
```

4. Добавим в проект файл Lab2_main.cpp, содержащий основную программу. Для этого нужно:
   4.1. Вызвать контекстное меню проекта в панели Обозреватель решений (Solution Explorer), выбрать в нем пункт меню Add/ New Item.
   4.2. В диалоговом окне Add New Item – Lab2 выбрать Категорию Code, шаблон – C++File (.cpp), задать имя файла Lab2_main. В результате выполненных действий получим пустой файл Lab2_main.cpp, в котором будет редактироваться текст программы.
   4.3. Ввести следующий текст программы:

```
#include "Tovar.h"
#include <iostream>
#include <string>
using namespace std;
//функция для возврата объекта как результата
Tovar make_tovar()
{
	string s;
	int i;
	double d;
	cout<<"Vvedite naimenovanie: ";
	cin>>s;
	cout<<"Vvedite kolichestvo: ";
```

```
        cin>>i;
        cout<<"Vvedite stoimost: ";
        cin>>d;
        Tovar t(s,i,d);
        return t;
}
//функция для передачи объекта как параметра
void print_tovar(Tovar t)
{
        t.show();
}
void main()
{
        //конструктор без параметров
        Tovar t1;
        t1.show();
        //коструктор с параметрами
        Tovar t2("Computer", 1,15000);
        t2.show();
        //конструктор копирования
        Tovar t3=t2;
        t3.set_naimenovanie("Telephon");
        t3.set_kolichestvo(2);
        t3.set_stoimost(5000.0);
        //конструктор копирования
        print_tovar(t3);
        //конструктор копирования
        t1=make_tovar();
        t1.show();
}
```

5. Выполнить компиляцию программы, используя команду Build / Build Solution или функциональную клавишу F7. Исправить имеющиеся синтаксические ошибки и снова запустить программу на компиляцию.
6. Запустить программу на выполнение, используя команду Debug/ Start Without Debugging или комбинацию функциональных клавиш Ctrl+F5.
7. Изучить полученные результаты, сделать выводы.

5. Варианты

№	Задание
1	Пользовательский класс СТУДЕНТ ФИО – string Группа – string Средний балл – float
2	Пользовательский класс КНИГА Название – string Автор – string Год издания – int
3	Пользовательский класс МАРШРУТ Пункт отправления– string Пункт назначения – string Время в пути – float

4	Пользовательский класс ЧИСЛО Мантисса – foat Порядок – int Строковое представление – string
5	Пользовательский класс УРАВНЕНИЕ Коэффициент A – double Коэффициент B – double Коэффициент C – double
6	Пользовательский класс СТРАНА Столица – string Количество жителей – int Площадь – double
7	Пользовательский класс КОМПЬЮТЕР Процессор – string Объем ОП – int Объем ЖД – int
8	Пользовательский класс КВАРТИРА Адрес – string Площадь – double Количество комнат – int
9	Пользовательский класс СОТРУДНИК ФИО – string Должность – string Зарплата – double
10	Пользовательский класс ЭКЗАМЕН ФИО студента – string Предмет – string Оценка – int
11	Пользовательский класс АВТОМОБИЛЬ Марка – string Модель – string Стоимость– int
12	Пользовательский класс КВИТАНЦИЯ Номер – int Дата – string Сумма – double
13	Пользовательский класс АБИТУРИЕНТ ФИО абитуриента– string Специальность – string Балл ЕГЭ– int

14	Пользовательский класс БАНКОМАТ Идентификационный номер – int Остаток денег в банкомате – double Максимальная сумма, которую может снять клиент – double
15	Пользовательский класс ЗАРПЛАТА ФИО – string Оклад – double Премия (% от оклада) – int

6. Контрольные вопросы

1. Для чего нужен конструктор?
2. Сколько типов конструкторов существует в C++?
3. Для чего используется деструктор? В каких случаях деструктор описывается явно?
4. Для чего используется конструктор без параметров? Конструктор с параметрами? Конструктор копирования?
5. В каких случаях вызывается конструктор копирования?
6. Перечислить свойства конструкторов.
7. Перечислить свойства деструкторов.
8. К каким атрибутам имеют доступ методы класса?
9. Что представляет собой указатель this?
10. Какая разница между методами определенными внутри класса и вне класса?
11. Какое значение возвращает конструктор?
12. Какие методы создаются по умолчанию?
13. Какое значение возвращает деструктор?
14. Дано описание класса

```
class Student
{
string name;
int group;
public:
student(string, int);
student(const student&)
~student();
};
```

Какой метод отсутствует в описании класса?

15. Какой метод будет вызван при выполнении следующих операторов:

```
student*s;
s=new student;
```

16. Какой метод будет вызван при выполнении следующих операторов:

```
student s(“Ivanov”,20);
```

17. Какие методы будут вызваны при выполнении следующих операторов:

```
student s1(“Ivanov”,20);
student s2=s1;
```

18. Какие методы будут вызваны при выполнении следующих операторов:

```
student s1(“Ivanov”,20);
student s2;
s2=s1;
```

19. Какой конструктор будет использоваться при передаче параметра в функцию print():

```
void print(student a)
{a.show();}
```

20. Класс описан следующим образом:

```
class Student
{
string name;
int age;
public:
void set_name(string);
void set_age(int );
…..
};
```

Student p;

Каким образом можно присвоить новое значение атрибуту name объекта p?

7. Содержание отчета

1) Постановка задачи (общая и конкретного варианта).
2) Описание класса.
3) Определение компонентных функций.
4) Определение функций make_() и print_().
5) Объяснение результатов работы программы.