ЛАБОРАТОРНОЙ РАБОТЫ № 2

По дисциплине: Объектно-ориентированное программирование

Тема работы: **Создание и уничтожение объектов с использованием конструкторов и деструкторов. Управление доступом к элементам класса.**

Цель работы: Научить создавать объекты посредством использования конструктора, перегружать методы, управлять доступом к экземплярам класса.

Количество часов: 2ч.

Содержание работы (Задание, Задачи):

1. Для классов, описанных в лабораторной работе №1 сформировать заголовочные файлы. Продемонстрировать отделение интерфейса от реализации. (см. Пример 1)
2. Для классов, описанных в лабораторной работе №1 предусмотреть следующие конструкторы:
    a) неинициализирующий конструктор без параметров; инициализирующий параметрический конструктор;
    b) инициализирующий конструктор без параметров;
    c) инициализирующий конструктор, предполагающий значения по умолчанию для всех полей;
    d) инициализирующий конструктор без параметров со списком инициализации; инициализирующий параметрический конструктор.
3. Для класса c) предусмотреть деструктор. Создать локальные и глобальные объекты класса. Продемонстрировать работу деструктора.
4. Для класса d) прописать методы изменения полей (set-, get-). Продемонстрировать их работу.

Методические указания по выполнению:

I. Конструктор

Конструктор – метод класса, который автоматически вызывается при выделении памяти под объект. Конструктор имеет то же имя, что и класс, может иметь аргументы, но не возвращает значения, может быть параметрически перегружен.

Конструктор определяет операции, которые необходимо выполнить при создании объекта. Традиционно такими операциями являются инициализация полей класса и выделение памяти под динамические поля, если такие в классе объявлены. Явный вызов конструктора не возможен.

Как и любая другая функция с параметрами, конструктор может быть *переопределен (паметрически перегружен)*. Поэтому класс может иметь несколько конструкторов, позволяющих использовать разные способы инициализации полей объектов.

- **Конструктор с параметрами.**
  - инициализирующий конструктор
    Num(int an){n=an;}
  - инициализирующий конструктор, с параметрами заданными по умолчанию
    Num(int an=10){ n=an; }
  - инициализация полей фиксированного и ссылочного типа посредством списка инициализации

```
class Num{
        public:
                const int n;                    //константное поле
                int &c;                         //ссылочное поле
                Num(int an, int &ac):n(an),c(ac){ }     /* инициализация полей фиксированного и ссылочного типа посредством списка инициализации */
};

void main()
{
        int k = 13;
```

```
    Num bb(10,k);  // инициализируемый объект
}
```

- **Конструктор без параметров.** Если в классе не объявлены конструктор и деструктор, то компилятор автоматически выполняет построение «пустых» (без параметров и операторов) конструктора и деструктора. Если же хотя бы один конструктор в классе объявлен, то автоматический пустой конструктор уже не создается. Это значит, что, при наличии в классе только конструкторов, требующих задания параметров, создание неинициализированных объектов в данном классе будет недоступно!

  Таким образом, создание объектов без указания аргументов требует, чтобы в классе был задан один из конструкторов, которые могут быть вызваны без указания аргументов, а именно:

  - неинициализирующий конструктор без параметров («пустой»), например:
    Num(){}
  - инициализирующий конструктор без параметров, например:
    Num(){n=0;} или со списком инициализации *Num():n(0){}*
  - инициализирующий конструктор, предполагающий значения по умолчанию для всех полей, например:
    Num(int an=0){ n=an; }

- **Конструктор копирования.** Копирующие конструкторы могут определяться в классе явно, но могут использоваться и копирующие конструкторы, определенные по умолчанию.

Ниже приведен пример программы, демонстрирующей создание объектов с использованием конструкторов, разбитой на несколько файлов.

При построении программы каждое определение класса обычно помещается в заголовочный файл, а определения элементов функций этого класса помещаются в файлы исходного кода с тем же базовым именем. Заголовочные файлы включаются (посредством #include) во все файлы, использующие этот класс, а исходный файл с определениями элементов-функций компилируется и компонуется с файлом, содержащим главную программу.

Пример 1. Использование конструктора

```
//point.h
//объявление класса Point
//элементы-функции определены в Point.cpp
//данный препроцессорный код
//не допускает многократных включений заголовочного файла
#ifndef point_H
#define point_H

class Point{
public:
    Point(int, int);        //конструктор с параметрами
    Point(){}               //неинициализирующий конструктор, "пустой"
    void setXY(int, int);   //явный инициализирующий метод
    void print(void);
private:
    int x;
    int y;
};

#endif;
```

```
//point.cpp
#include <iostream>
#include "point.h"
using namespace std;

Point::Point(int ax, int ay){
    x = ax; y = ay;
}

void Point::setXY(int ax, int ay){
    x = ax;
    y = ay;
}

void Point::print(void){
    cout<<"coordinates ("<<x<<" : "<<y<<")"<<endl;
}
```

```
//person.h
#ifndef person_H
#define person_H

#include <iostream>

class Person{
public:
    Person(char *, string, int);  /* прототип конструктора с
                                     параметрами, заданными  по умолчанию */
    ~Person(){}                  // деструктор
    void print(void);
    void setCity(string);
private:
    char name[15];
    string city;
    int year;
};

#endif;
```

```
//person.cpp
#include <iostream>
#include <string.h>
#include "person.h"
using namespace std;

Person::Person(char *aname = "anonym", string acity = "Simf", int ayear = 1990){  /* тело конструктора с
                                                                параметрами, заданными по умолчанию */
    strcpy(name,aname);
    city = acity;
    year = ayear;
}

void Person::print(void){
    cout<<"Name: "<<name<<endl<<"City: "<<city<<endl<<"Year: "<<year<<endl;
}
void Person::setCity(string acity){
    city = acity;
}
```

```
//программа-тестер для классов Point и Person
#include "person.h"
#include "point.h"
#include <iostream>
#include <string>
#include <conio.h>
using namespace std;

void main()
{
    Point A, B(2,3);
    A.print();
    A.setXY(1,1);
    A.print();
    B.print();

    Person C;                       //объект с полями по умолчанию
    Person E("Vasya");              //объект с заданным именем
    Person R("Paul","Yalta");       //объект с заданными именем и городом
    Person D("Max","Kiev",1989);    //объект с заданными именем,городом,годом
    Person K = D;                   //конструктор копирования // задается по умолчанию

    C.print();  E.print();  R.print();  D.print();  K.print();

    _getch();
}
```

Помещайте объявление класса в заголовочный файл, который должен быть включен любым клиентом, желающим использовать этот класс. Это - открытый интерфейс класса. Помещайте определения элементов-функций этого класса в отдельный исходный файл. Это – реализация класса.

II. Деструктор

При освобождении объектом памяти автоматически вызывается другой специальный метод класса – деструктор. Имя деструктора по аналогии с именем конструктора, совпадает с именем класса, но перед ним стоит символ «~» («тильда»).

Деструктор определяет операции, которые необходимо выполнить при уничтожении объекта. Обычно он используется для освобождения памяти, выделенной под динамические поля объекта данного класса конструктором. Деструктор не возвращает значения и не имеет параметров. Класс может иметь только один деструктор или не иметь ни одного. В отличие от конструктора деструктор может вызываться явно.

Момент уничтожения объекта, а, следовательно, и автоматического вызова деструктора определяется типом памяти, выбранным для размещения объекта: локальная, глобальная, внешняя и т. д. Если программа завершается с использованием функции *exit*, то вызываются деструкторы только глобальных объектов. При аварийном завершении программы, использующей объекты некоторого класса, функцией *abort* деструкторы объектов не вызываются.

III. Управление доступом к элементам класса

Элементы класса принято объявлять закрытыми (*private*), однако это не означает что клиенты, использующие класс, не смогут производить изменения в этих данных. Для этого предусмотрены функции *доступа* ***set-, get-функции.*** Наличие подобных функций не нарушает концепции закрытых данных, так как ***set-, get-функции*** не обязаны устанавливать предлагаемые значения (***set-*** может предусматривать проверку целостности данных) и возвращать данные в чистом виде.

Пример 2. Использование set-, get-функций

```
#include <locale.h>
```

```
#include <string.h>
#include <iostream>
using namespace std;

class child
{
 private:    char name[20];    int  age;
 public:
     void print(void)
     {              cout<<" Имя: "<<name;      cout<<" Возраст : "<<age<< endl;   }

    child(char *Name,int Age):age(Age)
     {        strcpy(name,Name);   }

    void setChild(char*Name,int Age){
          strcpy(name,Name);
          age = (Age < 0)  ? 0 : Age;  }
    void setName(char *Name){
          strcpy(name,Name);   }
    void setAge(int Age){
          age = (Age < 0)  ? 0 : Age; }

    int  getAge(){
          return age; }
};

void main()
{
```

```
    setlocale(0,"russian");
    child aa("Мария",6);
    aa.print();    // выводит:   Имя: Мария   Возраст: 6
    child bb=aa;   // вызывает копирующий конструктор
    bb.print();    // выводит:   Имя: Мария   Возраст: 6
    system("pause");
}
```

Пособия и инструменты:

Visual Studio 2008

Вопросы для защиты лабораторной работы:

1. Как называется множество открытых элементов-функций класса?
2. Говорят, что реализация класса скрыта от его клиентов или …
3. Какую функцию следует использовать для присваивания значений закрытым элементам класса?
4. Сформулируйте особенности конструкторов и деструкторов классов C++. Что такое неинициализирующий конструктор и чем он отличается от конструктора без параметров?
5. Когда использование конструктора, вызываемого без аргументов, необходимо?
6. Найдите ошибку в каждом из следующих фрагментах. Объясните, как ее необходимо исправить.
   a) Предположим, что в классе *Time* объявлен следующий прототип:
      void ~Time(int);
   b) Предположим, что в классе *Employee* объявлен следующий прототип:
      *int Employee(char *, int);*

Литература:

1. Скляров В.А. Язык C++ и объектно-ориентированное программирование. – М.: Высшая школа., 1997.
2. Иванова Г.С., Ничушкина Т.Н., Пугачев Е.К. Объектно-ориентированное программирование: Учебник для ВУЗов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2007.

3. Березин Б.И., Березин С.Б. Начальный курс С и С++. – М.: «Диалог-МИФИ», 1997.
4. Вайнер Р., Пинсон Л. С++ изнутри: Пер. с англ. – Киев: «ДиаСофт», 1993.
5. Дейтел Х., Дейтел П. Как программировать на С++: пер. с англ./ – М.: ЗАО «Издательство БИНОМ», 2005 г. – 1024 с.: ил.