Практическое занятие № 13 «Классы в C++»

Учебные цели:

- получение умений и навыков создания классов в C++.

Воспитательные цели:

- формировать диалектико-материалистическое мировоззрение;
- формировать навыки самостоятельности и дисциплинированности;
- стимулировать активную познавательную деятельность обучаемых, способствовать формированию у них творческого мышления.

Категория слушателей: 2,3 курс.
Время: 90 мин.
Место проведения: компьютерный класс.
Материально-техническое обеспечение:
1) персональный компьютер *IBM PC* с операционной системой Windows XP; 2) среда разработки приложений *Visual C++.NET.*

ПЛАН ПРАКТИЧЕСКОГО ЗАНЯТИЯ

Учебные вопросы	Время, мин
Вступительная часть	5
1. Создание классов в C++	15
2. Выполнение индивидуального задания.	65
Заключительная часть	5

Элементы теории

Абстракция данных есть метод разработки программ с представлением в ней понятий из прикладной области как пользовательских (user-defined) типов данных. Интерфейс типа данных (спецификация) отделяется от его реализации, что

− облегчает понимание программ, позволяя прямо описывать понятие в естественных для него терминах;

− изменять реализацию типа данных, не задевая пользователей.

C++ обеспечивает абстракцию данных, поддерживая следующие понятия и возможности:

− конструктор пользовательских типов `class`;

− средства управления доступом (`public`, `private`);

− абстрактные классы;

− гарантированная инициализация и очистка объектов;

− пользовательские преобразования типов объектов;

− параметризованные (родовые) функции и типы данных;

− обработка исключительных ситуаций.

С точки зрения языка программирования класс объектов можно рассматривать как тип данного, а отдельный объект − как данное этого типа. Определение программистом собственных классов объектов для конкретного набора задач должно позволить описывать отдельные задачи в терминах самого класса задач (при соответствующем выборе имен типов и имен объектов, их параметров и выполняемых действий).

Основные идеи объектно-ориентированного подхода опираются на следующие положения:

- программа представляет собой модель некоторого реального процесса, части реального мира.
- модель реального мира или его части может быть описана как совокупность взаимодействующих между собой объектов.
- объект описывается набором параметров, значения которых определяют состояние объекта, и набором операций (действий), которые может выполнять объект.
- взаимодействие между объектами осуществляется посылкой специальных сообщений от одного объекта к другому. Сообщение, полученное объектом, может потребовать выполнения определенных действий, например, изменения состояния объекта.
- объекты, описанные одним и тем же набором параметров и способные выполнять один и тот же набор действий, представляют собой класс однотипных объектов.

Таким образом, объектно-ориентированный подход предполагает, что при разработке программы должны быть определены классы используемых в программе объектов и построены их описания, затем созданы экземпляры необходимых объектов и определено взаимодействие между ними.

Классы объектов часто удобно строить так, чтобы они образовывали иерархическую структуру. Например, класс «Студент», описывающий абстрактного студента, может служить основой для построения классов «Студент 1 курса», «Студент 2 курса» и т.д., которые обладают всеми свойствами студента вообще и некоторыми дополнительными свойствами, характеризующими студента конкретного курса. В таких иерархических структурах один класс может рассматриваться как базовый для других, производных от него классов. Объект производного класса обладает всеми свойствами базового класса и некоторыми собственными свойствами, он может реагировать на те же типы сообщений от других объектов, что и объект базового класса и на сообщения, имеющие смысл только для производного класса. Обычно говорят, что объект производного класса наследует все свойства своего базового класса.

Одним из наиболее важных понятий C++ является класс. Класс представляет собой механизм для создания новых типов. Новый тип создается путем объявления класса. Класс − коллекция переменных (часто различных типов), скомбинированная с набором связанных функций.

Синтаксис описания класса похож на синтаксис описания структуры.

```
class имя_класса
{
[private:]
  закрытые элементы – члены класса
public:
  открытые элементы - члены класса
};
```

Имя_класса с этого момента становится новым именем типа данных, которое используется для объявления объектов класса.

Члены класса – это переменные-члены (поля) и функции-члены (методы) этого класса, иными словами членами класса могут быть как переменные, так и функции. Функции и переменные, объявленные внутри объявления класса, становятся членами этого класса. Функции-члены класса будем называть методами этого класса.

По умолчанию, все функции и переменные, объявленные в классе, становятся закрытыми (***private***). Т.е. они доступны только из других членов этого класса. Для объявления открытых членов класса используется ключевое слово ***public***. Все функции-методы и переменные, объявленные после слова ***public***, доступны и для других членов класса, и для любой другой части программы, в которой содержится класс.

Поля класса:

- могут иметь любой тип, кроме типа этого же класса (но могут быть указателями или ссылками на этот класс);
- могут быть описаны с модификатором ***const***, при этом они инициализируются только один раз (с помощью конструктора) и не могут изменяться;
- могут быть описаны с модификатором ***static***, при этом они создаются для всех объектов класса в единственном экземпляре, то есть не дублируются.

Инициализация полей при описании *не допускается*.

Классы могут быть *глобальными* (объявленными вне любого блока) и *локальными* (объявленными внутри блока, например, функции или другого класса).

Перечислим некоторые особенности локального класса:

- внутри локального класса можно использовать типы, статические (*static*) и внешние (*extern*) переменные, внешние функции и элементы перечислений из области, в которой он описан; запрещается использовать автоматические переменные из этой области;
- локальный класс не может иметь статических элементов;
- методы этого класса могут быть описаны только внутри класса;
- если один класс вложен в другой класс, они не имеют каких-либо особых прав доступа к элементам друг друга и могут обращаться к ним только по общим правилам.

В качестве примера создадим класс окружностей с центром в начале координат. Для этого требуется описать его поля и методы.

Пример 1.

```
#include <iostream>  // для объектов cin и cout
#include "math.h"          // для функции sqrt()
class Circle             // класс окружностей с центром
                         // в начале координат
 { double x, y;      // переменные-члены (поля)
                     // (x, y) – координаты точки,
                     // принадлежащей окружности
 public:
 void Set_x_y (double x1, double y1); // метод доступа –
                           // инициирует поля класса x и y
  double Get_Radius () {return sqrt(x*x+y*y);}; // метод
                          // доступа – возвращает значение
                          // радиуса окружности
  double Length_Circle( double r);     // метод доступа –
                            // возвращает длину окружности
   double Square_Circle(); // метод доступа –
                           // возвращает площадь окружности
 };
```

В C++ для создания класса традиционно принято использовать ключевое слово ***class***. Хотя методы *Set_x_y (), Length_Circle()* и S*quare_Circle()* объявлены в классе *Circle* , они еще не определены. Для определения метода − члена класса нужно связать имя метода с именем класса. Это достигается путем написания имени метода вслед за именем класса с двумя двоеточиями. Два двоеточия называют операцией расширения области видимости.

```
 void Circle :: Set_x_y (double x1, double y1)
 {
   x=x1;
   y=y1;
 }
double Circle :: Length_Circle( double r)
{
return 2*3.1415*r;

}
double Circle :: Square_Circle()
{
const double PI=3.1415;
double S;
S=PI*(X*X+Y*Y);
return S;     }
```

В теле функции *main()* создадим две окружности *A* и *B*, введя координаты соответствующих точек окружностей, определим их радиус, длину и площадь.

```
int main ()
{
using namespace std;
Circle A, B;  // создание объектов A и B класса Circle
```

```
double x1, y1;
cout << "Vvedite koordinaty tochki okrugnosti A"<<endl;
cout << "Vvedite X1=";
cin>>x1;
cout << "Vvedite Y1=";
cin>>y1;
A.Set_x_y(x1,y1);
cout << "Vvedite koordinaty tochki okrugnosti B"<<endl;
cout << "Vvedite X1=";
cin>>x1;
cout << "Vvedite Y1=";
cin>>y1;
B.Set_x_y(x1,y1);
cout<<"Ra="<<A.Get_Radius()<<"\t\t";
cout<<"Rb="<<B.Get_Radius()<<endl;
cout<<"La="<<A.Length_Circle(A.Get_Radius())<<"\t";
cout<<"Lb="<<B.Length_Circle(B.Get_Radius())<<endl;
cout<<"Sa="<<A.Square_Circle()<<"\t";
cout<<"Sb="<<B.Square_Circle()<<endl;
return 0;
}
```

Результаты работы программы представлены на рисунке 1.

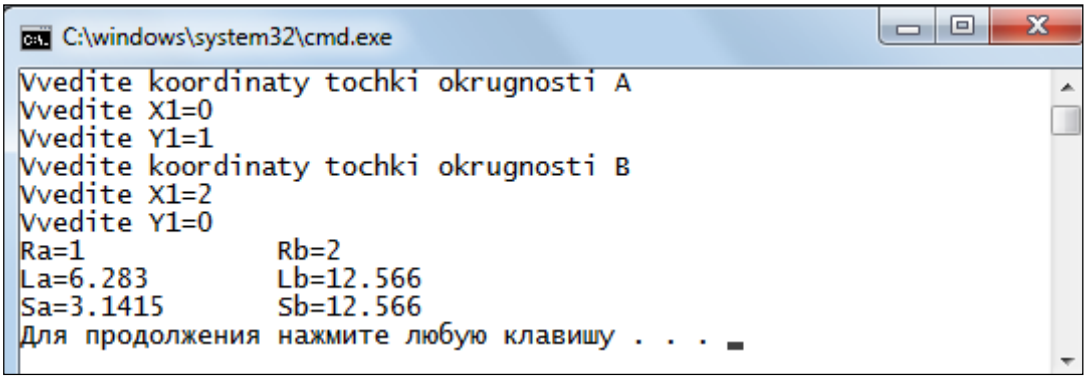

Рис. 1. Результаты работы программы нахождения радиуса, длины и площади окружности

В каждом классе есть хотя бы один метод, имя которого совпадает с именем класса. Он называется *конструктором* и вызывается автоматически при создании объекта класса. Конструктор предназначен для инициализации объекта. Автоматический вызов конструктора позволяет избежать ошибок, связанных с использованием неинициализированных переменных.

Описание *конструктора*:

```
идентификатор_конструктора (список параметров)
{тело конструктора}
```

Идентификатор конструктора должен совпадать с именем класса. У конструктора не указывается тип возвращаемого значения (даже void). Конструктор может иметь значения параметров по умолчанию.

Правила использования конструкторов:

− если конструктор не вызывается явно, то он вызывается автоматически при создании объекта с использованием значений параметров по умолчанию;

− если конструктор не описан явно, он генерируется транслятором автоматически.

Вызывать конструктор можно так:

имя_класса имя_объекта = имя_конструктора (список фактических параметров);

имя_конструктора имя_объекта (список фактических параметров);

Рассмотрим пример использования конструктора для инициализации объектов класса *Circle* и установки по умолчанию значений координат точки окружности x1=1, y1=1.

Пример 2.

```
class Circle
{ double x, y;
 public:
 Circle(double x1=1,double y1=1); // конструктор
 void Set_x_y (double x1, double y1);
 double Get_Radius () {return sqrt(x*x+y*y);};
 double Length_Circle( double r);
 double Square_Circle(); };
// конструктор класса Circle
 Circle:: Circle(double x1, double y1)
 {
 x=x1;
 y=y1;
 }
 void Circle :: Set_x_y (double x1, double y1)
 {
 ...   // определение метода Set_x_y ()
 }
double Circle :: Length_Circle( double r)
{
...    // определение метода Length_Circle ()
}
double Circle :: Square_Circle()
{
...          // определение метода Square_Circle ()
}
int main ()
{
using namespace std;
Circle A, B(2,0); // создаем окружности A(1,1) и B(2,0)
cout<<"Ra="<<A.Get_Radius()<<"\t";
cout<<"Rb="<<B.Get_Radius()<<endl;
cout<<"La="<<A.Length_Circle(A.Get_Radius())<<"\t";
cout<<"Lb="<<B.Length_Circle(B.Get_Radius())<<endl;
cout<<"Sa="<<A.Square_Circle()<<"\t";
cout<<"Sb="<<B.Square_Circle()<<endl;
return 0;
```

```
}
```

Результаты работы программы представлены на рисунке 2.

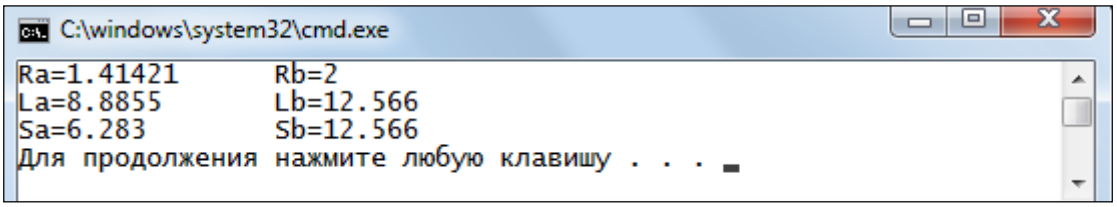

Рис. 2. Результаты работы программы с использованием конструктора для инициализации объектов класса *Circle*

Задание 1. а) Создать массив объектов (не менее 3), содержащий сведения о пациентах городской больницы. Структура записи (класса): фамилия и инициалы пациента, пол, возраст, место проживания (город), диагноз, массив дат и температур пациента, количество дней наблюдения за пациентом, возраст, номер палаты, отделение. Количество записей произвольное (не более 30).

б) Написать программу, выдающую на экран:

- список пациентов, фамилии которых начинаются с введенной строки символов. Кроме указания фамилии и инициалов пациентов, вывести их пол, данные наблюдения за их температурой и среднее значение температуры за время наблюдения.

Листинг 1.

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

struct d_tem
{
	char data[11];
	float temperatura;
};

class C_pacient
{
	public:

	char* GetFam  (void)
	{	return cFam_in;
	};

	void SetFam (int i)
	{char newline;
	cout << "Vvedite FIO "<<i<<"-ogo pacienta: ";
```

```
cin.get(cFam_in, 29, '\n');
cin.get(newline);
};

char* GetPol  (void)
{     return cpol;
};

void SetPol(int i)
{char newline;
cout << "Vvedite pol "<<i<<"-ogo pacienta: ";
cin.get(cpol, 5, '\n');
cin.get(newline);};

void SetdTem(int m)
{char newline;
for (int i=0; i<m; i++)
{ cout << "Vvedite "<<i+1<<"-yu datu: ";
cin.get(dTem[i].data, 11, '\n');
cin.get(newline);
cout << "Vvedite temperaturu: ";
cin>>dTem[i].temperatura;
cin.get(newline);  }
};
void GetdTem(int m)
{ for (int i=0; i<m; i++)
cout << dTem[i].data<<'\t'<<dTem[i].temperatura<<endl;
cout << endl;
};

void SetkDen(int m)
{
ckDen=m;
};

int GetkDen(void)
{return ckDen;};

void den(int m)
{ int k;
k=m%10;
switch (k)
{
case 1: cout <<" den' ";
      break;
case 2: cout<<" dnay ";
      break;
case 3: cout<<" dnay ";
      break;
case 4: cout<<" dnay ";
      break;
default : cout <<" dney "; };
};
```

```
        void Srtem (int m)
        {       float sum=0;
                for (int i=0; i<m; i++)
                sum+=dTem[i].temperatura;
                cout << "Sr. tem-ra ="<< sum/m <<" (za "<<m;
                den(m);
                cout<<" nabludeniay)"<<endl;
        };
private:

        char cFam_in[30];
        char cpol[5];
        d_tem dTem[10];
        int ckDen;
        int cvozrast;
        int cgorod;
        char cdiagnoz[200];
        int cnom_pal;
        char cotdelenie[150];
};
int main()
{
        C_pacient bolnye[30];
        int n,m;
        char newline;
        cout<<"Vvedite kol-vo pacientov n=";
        cin>>n;
        cin.get(newline);
        for (int i=0; i<n; i++)
                {cout<<i+1<<". ";
                bolnye[i].SetFam(i+1);
                bolnye[i].SetPol(i+1);
                cout<<"Vvedite kol-vo dney nabludeniay m=";
                cin>>m;
                cin.get(newline);
                bolnye[i].SetkDen(m);
                bolnye[i].SetdTem(m);
                }
        cout << endl;
        char str[30];
        int s=1;
        while (s)
        {
         cout<<"Vvedite 1 - dlay poiska pacientov"<<endl;
         cout<<"Vvedite 0 - dlay vyhoda iz programmy."<<endl;
         cin>>s;
         cin.get(newline);
        if (s==0) return 0;
        cout<<"Vvedite familiu pacienta Fam= ";
        cin.getline(str, 29);
        string find_fam(str);
        for (int i=0; i<n; i++)
        { string Fam(bolnye[i].GetFam());
```

```
        if (-1!=Fam.find(find_fam,0))
        {cout<<Fam<<"\t"<<bolnye[i].GetPol()<<"\t"<<" nabludenie:
"
             <<bolnye[i].GetkDen()<<" ";
         bolnye[i].den(bolnye[i].GetkDen());
         cout<<endl;
        bolnye[i].GetdTem(bolnye[i].GetkDen());
        bolnye[i].Srtem(bolnye[i].GetkDen());
        }
        }
        cout<<endl;
        }
}
```

Контрольные вопросы

1. Дать определение понятиям класс, объект.
2. Основные концепции объектно-ориентированного программирования.
3. Инкапсуляция, наследование, полиморфизм.
4. Объектно-ориентированный подход к разработке программ.
5. История и основы языка C++.
6. Примеры простых программ на языке C++.
7. Объектно-ориентированные средства языка C++.
8. Объекты, классы.
9. Инкапсуляция данных и методы доступа?