Урок 10. Объектно-ориентированное программирование в C++

Цель работы: приобрести навыки создания собственных классов и познакомиться с основными идеями объектно-ориентированного программирования.

Основным отличием языка C++ от C является наличие в нем средств объектно-ориентированного программирования (ООП).

Часто в литературе язык C++ определяют именно как язык объектно-ориентированного программирования. Для C++ базовые понятия ООП: это инкапсуляция, наследование и полиморфизм.

Класс — это структурированный тип, включающий в себя в качестве элементов типизированные данные и функции, применяемые по отношению к этим данным. Таким образом, инкапсуляция (объединение параметров и методов) заложена в составе элементов класса: типизированные данные — это параметры, а методы реализованы через функции.

Тип «класс» устанавливается для объектов. Принято говорить: однотипные объекты принадлежат одному классу.

Синтаксис объявления класса подобен синтаксису объявления структуры. Объявление начинается с ключевого слова class, за которым следует имя класса. В простейшем случае объявление класса имеет следующий формат:

```
class  имя
{
тип1  переменная1
тип2  переменная2
public:
функция1;
функция2;
};
```

Основное отличие класса от структур состоит в том, что все члены класса по умолчанию считаются закрытыми и доступ к ним могут получить только функции — члены этого же класса. Однако режим доступа к элементам класса может быть изменен путем его явного указания. Для этого перед элементами класса записывается соответствующий спецификатор доступа. Существуют три таких спецификатора:

- `private  (частный);`
- `public  (общедоступный);`
- `protected  (защищенный).`

Режим доступа private обозначает, что соответствующий элемент может использоваться только функциями данного класса. Этот режим доступа устанавливается по умолчанию.

Элементы с режимом доступа public доступны в других частях программы. О режиме protected будет сказано немного позже. Чаще всего режим доступа к данным (переменным) бывает private, а к функциям — public. Это отражено в приведенном выше формате объявления класса.

В качестве примера рассмотрим объявление класса обыкновенных дробей с именем Drob, в котором значение дроби определено через структуру двух целых чисел (числитель и знаменатель), а к методам работы с дробью отнесены ввод дроби — функ-

ция Vvod; вычисление наибольшего общего делителя числителя и знаменателя — функция NOD; сокращение дроби — функция Sokr; возведение дроби в целую степень — функция stepen — и вывод дроби на экран — функция Print. Объявление соответствующего класса выглядит так:

```
class  Drob
{
Frac  A;
public:
void  Vvod(void);
int  NOD(void);
void  Sokr(void);
void  Stepen(int  N);
void  Print (void) ;
};
```

Имеется в виду, что глобально по отношению к этому классу объявлена структура с именем Frac:

```
struct Frac
{
int P;
int Q;
};
```

Таким образом, пять методов реализованы пятью функциями, которые в объявлении класса представлены своими прототипами.

Описания функций — членов класса производится отдельно. При этом нужно указывать, к какому классу принадлежит данная функция. Для этого используется операция принадлежности, знак которой :: . Например:

```
void  Drob::Vvod(void)
{
cout << "Числитель?";   cin >> A.P;
cout << "Знаменатель?"; cin >>A.Q;
}
```

В основной части программы (основной функции) класс Drob будет поставлен в соответствие определенным переменным в качестве типа. Например:

```
Drob Y;
```

После этого переменная y воспринимается в программе как объект соответствующего класса. Для основной программы открыты только функции этого объекта. Следовательно, воздействовать на параметры объекта можно только через эти функции.

Аналогично элементам структуры обращение к элементам объекта производится с помощью составного имени (через точку). Например: Y.Vvod( ).

Пример 1.

```
//В программе объявлен класс простых дробей,
//описана переменная этого класса, выполнена
//обработка  переменной
#include <iostream>
#include <cmath>
#include <clocale>
```

```
using namespace std;
struct Frac
{
int P;
int Q;
};
Frac F;
//Объявление класса
class Drob
{
Frac A;
public:
void  Vvod(void);
int  NOD(void);
void  Sokr(void);
void  Stepen(int N);
void  Print(void);
};
//Описания функций — членов класса
void  Drob::Vvod(void)
{
cout << "Числитель?"; cin >> A.P;
cout << "Знаменатель?";  cin >> A.Q;
}
int Drob::NOD(void)
{
int M, N;
M = abs(A.P);
N = A.Q;
while(M != N)
{
if(M > N)
if(M % N != 0) M = M % N; else M = N;
else  if(N % M != 0) N = N % M; else N = M;
}
return M;
}
void  Drob::Sokr(void)
{
int X;
X = NOD( );
if(A.P != 0)
{
A.P = A.P/X;
A.Q = A.Q/X;
}
else A.Q = 1;
}
void Drob::Stepen(int N)
{
int  i;
F.P = F.Q = 1;
for(i = l; i <= N; i++)
{
F.P *= A.P;
```

```
F.Q *= A.Q;
}
}
void  Drob::Print(void)
{
cout << "\n" << A.P << "/" << A.Q << "\n";
}
//Основная функция
int main(void)
{
setlocale(LC_ALL,"RUS");
Drob  Y;  //Объявление  объекта
cout << "Вводите  дробь !" << "\n";
Y.Vvod( );
Y.Sokr( );
cout << "flpo6b  после  сокращения:"<<"\n" ;
Y.Print( );
Y.Stepen(2);
cout << "flpo6b,  возведенная  в квадрат:" << "\n";
cout << F.P << "/" << F.Q << "\n" ;
}
```

В результате выполнения этой программы на экране получим:

```
Вводите дробь!
Числитель? 6
Знаменатель? 15
Дробь после сокращения:
3/5
Дробь, возведенная  в квадрат:
9/25
```

Наследование — второе фундаментальное понятие ООП. Механизм наследования позволяет формировать иерархии классов.

Класс-наследник получает свойства класса-предка. В классе-наследнике могут быть объявлены новые дополнительные элементы. Элементы-данные должны иметь имена, отличные от имен предка. Элементы-функции могут быть новыми относительно предка, но могут и повторять имена функций своих предков. Здесь действует принцип «снизу вверх» при обращении к функции: функция потомка перекрывает одноименную функцию своего предка. Формат объявления класса-потомка следующий:

```
class имя_потомка: режим_доступа  имя_предка
{новые_элементы}
```

Для того чтобы элементы-данные класса-предка были доступны функциям класса-потомка, этим элементам должен быть поставлен в соответствие режим доступа protected (защищенный).

Пример 2.

```
//В программе объявлен исходный класс
//четырехугольников и классы-наследники
//параллелограммов, ромбов и квадратов,
#include <iostream>
#include <cmath>
#include <clocale>
```

```
using namespace std;
//Объявление базового класса четырехугольников
class  FourAngle
{
protected:
double x1, y1, x2, y2, x3, y3, x4, y4,
A, B, C, D, D1, D2,
Alpha, Beta, Gamma, Delta,
P, S;
public:
void  Init(void);
void Storony(void);
void Diagonali(void);
void Angles(void);
void Perimetr(void);
void Ploshad(void);
void PrintElements(void);
};
//Объявление класса параллелограммов —  наследника
//четырехугольников
class Parall: public  FourAngle
{
public:
void Storony(void);
void Perimetr(void);
void Ploshad(void);
};
//Объявление класса ромбов — наследника
//параллелограммов
class Romb: public  Parall
{
public:
void Storony(void);
void Perimetr(void);
};
//Объявление класса квадратов — наследника ромбов
class Kvadrat: public Romb
{
public:
void Angles(void);
void Ploshad(void);
};
//Описания функции — членов класса
void  FourAngle::Init(void)
{
cout << "\n, Введите  координаты вершин: \n";
cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3 >> x4 >> y4;
}
void  FourAngle::Storony(void)
{
A = sqrt((x2 - x1)*(x2 - x1) + (y2 - y1)*(y2 - y1));
B = sqrt((x3 - x2)*(x3 - x2) + (y3 - y2)*(y3 - y2));
C = sqrt((x4 - x3)*(x4 - x3) + (y4 - y3)*(y4 - y3));
D = sqrt((x4 - x1)*(x4 - x1) + (y4 - y1)*(y4 - y1));
}
```

```
void  FourAngle::Diagonali(void)
{
D1 = sqrt((x1 - x3)*(x1 - x3) + (y1 - y3)*(y1 - y3));
D2 = sqrt((x2 - x4)*(x2 - x4) + (y2 - y4)*(y2 - y4));
}
//Функция Ugol не является членом  какого-либо
//класса. Она выполняет вспомогательную роль для
//функции Angles. Эта функция может независимым
//образом использоваться и в основной программе для
//определения углов треугольника, заданного длинами
//сторон
double Ugol(double Aa, double Bb, double Cc)
{
double VspCos, VspSin, Pi;
Pi = 4*atan(1.0);
VspCos = (Aa*Aa + Bb*Bb - Cc*Cc)/2/Aa/Bb;
VspSin = sqrt(1 - VspCos*VspCos);
if(abs(VspCos) > 1e-7)
return  (atan(VspSin/VspCos) + Pi*(VspCos<0))/Pi/180;
else return 90.0;
}
void  FourAngle::Angles(void)
{
Alpha = Ugol(D, A, D2);  Beta = Ugol(A, B, D1);
Gamma = Ugol(B, C, D2);  Delta = Ugol(C, D, D1);
}
void  FourAngle::Perimetr(void)
{
P = A + B + C + D;
}
void  FourAngle::Ploshad(void)
{
double Per1, Per2;
Per1 = (A + D + D2)/2;
Per2 = (B + C + D1)/2;
S = sqrt(Per1*(Per1 - A)*(Per1 - D)*(Per1 - D2)) + sqrt(Per2*(Per2 -
B)*(Per2 - C)*(Per2 – D1));
}
void  FourAngle::PrintElements(void)
{
cout << "Стороны:\n" << A << "  " << B <<"  " << C << " " << D << "\n";
cout << "Углы:\n" <<Alpha <<"  "<<Beta << "  "<< Gamma << " " << Delta
<< "\n";
cout << "Периметр:\n" << P << "\n";
cout << "Площадь:\n" << S << "\n";
cout << "Диагонали:\n"<< D1 <<"  " << D2 << "\n";
}
void  Parall::Storony(void)
{
A = sqrt((x2 – x1)*(x2 – x1) + (y2 – y1)*(y2 – y1));
B = sqrt((x3 - x2)*(x3 - x2) + (y3 - y2)*(y3 - y2));
C = A; D = B;
}
void  Parall::Perimetr(void)
{
P = 2*(A + B);
```

```
}
void  Parall::Ploshad(void)
{
double Per;
Per = (A + D + D2)/2;
S = 2*sqrt(Per*(Per - A)*(Per - D)*(Per - D2));
}
void  Romb::Storony(void)
{
A = B = C = D = sqrt((x2 – x1)*(x2 – x1) + (y2 – y1)*(y2 – y1));
}
void  Romb::Perimetr(void)
{
P = 4*A;
}
void  Kvadrat::Angles(void)
{
Alpha = Beta = Gamma = Delta = 90.0;
}
void  Kvadrat::Ploshad(void)
{
S = A*A;
}
// Основная функция. По координатам вершин квадрата
// вычисляет и выводит все его параметры
int  main(void)
{
setlocale(LC_ALL,"RUS");
Kvadrat obj; //Объявление  объекта класса «квадрат»
obj.Init( );
obj.Storony( );
obj.Diagonali( );
obj.Angles( );
obj.Perimetr( );
obj.Ploshad( );
obj.PrintElements( );
}
```

В результате выполнения теста на экране будет получено:

```
Введите координаты вершин:
0 0 1 1 2 0 1 - 1
Стороны:
1.414214 1.414214 1.414214 1.414214
Углы:
90 90 90 90
Периметр:
5.656854
Площадь:
2
Диагонали:
2 2
```

Конструкторы и деструкторы. Основное назначение конструктора — инициализация элементов-данных объекта и выделение динамической памяти под данные. Конст-

руктор срабатывает при выполнении оператора определения типа «класс для объекта». Деструктор освобождает выделенную конструктором память при удалении объекта. Области памяти, занятые данными базовых типов, таких, как int, float, double и т.п., выделяются и освобождаются системой автоматически и не нуждаются в помощи конструктора и деструктора. Именно поэтому в программах, рассмотренных в примерах 1 и 2, конструкторы и деструкторы не объявлялись (система все равно создает их автоматически).

Конструктор и деструктор объявляются как члены-функции класса. Имя конструктора совпадает с именем класса. Имя деструктора начинается с символа ~ (тильда), за которым следует имя класса.

Пример 3. Объявляется класс для строковых объектов. В этом примере конструктор с помощью оператора new резервирует блок памяти для указателя stringl. Освобождение занятой памяти выполняет деструктор с помощью оператора delete.

```
class  string_operation
{
char *string1;
int  string_len;
public:
string_operatoin(char*)  //Конструктор
{
string1= new  char[string_len];
}
~string_operation( )  //Деструктор
{
delete string1;
}
void input_data(char*);
void output_data(char*);
};
```

В основной программе явного обращения к конструктору и деструктору не требуется. Они выполняются автоматически.

Полиморфизм допускает использование функций с одним и тем же именем (а также операций) применительно к разным наборам аргументов и операндов, а также к разным их типам, в зависимости от контекста программы. В C++ полиморфизм реализован через механизм перегрузки.

Внутри класса допускается существование нескольких функций с одинаковым именем, но различающимися типами результатов и наборами формальных параметров. При обслуживании обращения к такой функции компилятор выбирает подходящий вариант в зависимости от количества и типов аргументов.

Пример 4. В следующей программе определяется перегруженная функция modul( ) класса absolute, которая возвращает абсолютное значение как целочисленного, так и вещественного аргумента.

В первом случае для этого используется библиотечная функция abs(), принимающая аргумент типа int, во втором случае — fabs( ), принимающая аргумент типа double.

```
#include <iostream>
#include <cmath>
#include <clocale>
```

```
using namespace std;

class absolute
{
public:
int  modul(int);
double  modul(double);
};
int absolute::modul(int X)
{
return(abs(X));
}
double absolute::modul(double X)
{
return(fabs(X));
}
int main( )
{
setlocale(LC_ALL,"RUS");
absolute number;
cout << "Абсолютное  значение числа –765 равно:" << number.modul(-765)
<< endl;
cout << "Абсолютное  значение числа –23.987 равно:"<<number.modul(-
23.987) << endl;
}
```

В результате работы программы получим:

```
Абсолютное значение числа –7 65 равно 7 65
Абсолютное значение числа –23.987 равно 23.987
```

Перегрузка операций. Полиморфизм в C++ реализуется не только через механизм перегрузки функций, но и через перегрузку операций. Применительно к объектам определенного класса могут быть определены специфические правила выполнения некоторой операции. При этом сохраняется возможность ее традиционного применения в другом контексте.

Для перегрузки операции применительно к классу в число членов класса должна быть включена специальная функция операции, которая определяет действия, выполняемые по этой операции, формат определения функции-операции:

```
тип_возвращаемого_значения  operator  знак_операции
(спецификации_параметров_операции)  {тело_функции операции}
```

Рассмотрим пример программы, в которой используются перегруженные операции.

Пример 5. Класс vector определяет трехмерный вектор в евклидовом пространстве. В этом классе будут использоваться перегруженные операции сложения (+) и присваивания (=) как операции с трехмерными векторами. Сумма двух векторов вычисляется как вектор, компоненты которого равны суммам соответствующих компонент слагаемых. Операция = выполняет покомпонентное присваивание векторов.

```
#include <iostream>
#include <clocale>

using namespace std;
```

```
class vector
{
int x, y, z; //Компоненты  вектора
public:
vector operator + (vector t);
vector operator = (vector t);
void  show(void);
void assign(int mx, int my, int mz);
}
//Перегрузка  операции +
vector vector::operator + (vector t)
{
vector temp;
temp.x = x + t.x;
temp.y = y + t.y;
temp.z = z + t.z;
}
//Перегрузка  операции =
vector vector::operator = (vector  t)
{
x = t.x;
y = t.y;
z = t.z;
return  *this; // Использован  this
}
void  vector::show(void)
{
cout << x << ",  ";
cout << y << ",  ";
cout << x << \n;
}
void vector::assign(int mx, int my, int mz)
{
x = mx; y = my; z = mz;
}
//Основная  программа
int main(void)
{
setlocale(LC_ALL,"RUS");
vector a, b, c;
a.assign (1, 2, 3);
b.assign(10, 10, 10);
a.show( );
b.show( );
c = a + b; //Работают перегруженные  операции  +,  =
c.show( );
c = a + b + c;
c.show;
c = b = a;
c.show;
b.show;
}
```

В результате выполнения программы на экране получим:

```
1,    2,    3
10,   10,   10
11,   12,   13
22,   24,   26
1,    2,    3
1,    2,    3
```

Здесь естествен вопрос: почему бинарные операции-функции + и = имеют в описании только по одному аргументу? Дело в том, что другой аргумент всегда передается неявно с использованием this-указателя. Оператор temp.x = x + t.x; аналогичен строке temp.x = this -> x + t.x, т.е. x ссылается на this -> x. Здесь this ассоциируется с объектом, предшествующим знаку операции. Объект справа от знака операции передается как параметр функции.

Если аналогичным образом определять унарные операции как функции-члены класса, то для них не требуется указания параметра. Объект, к которому относится операция, передается в функцию неявно через указатель this. Например, для класса vector можно добавить объявление унарной постфиксной операции ++:

```
vector operator ++ (void);
```

Описание этой функции будет следующим:

```
vector  vector::operator ++ (void)
{
x++;
y++;
x++;
return  *this;
}
```

Упражнения

1. Определить класс Line, содержащий в качестве полей данных координаты начала и конца линии, а также содержащий методы для чтения и установки координат.
2. Определить класс DAY, содержащий в себе перечислимый тип, определяющий день недели (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday), а также методы для установки, чтения и показа на экране информации о том, какой день недели находится в данном объекте. При вызове из объекта метода показа на экран должны выводиться название дня недели и справка о том, рабочий он или выходной.
3. Опишите любой базовый и производный от него классы так, чтобы при срабатывании в них конструкторов и деструкторов на экран выдавались соответствующие надписи типа «Сработал такой-то метод из такого-то класса».
4. Определите класс integer, хранящий в поле данных целое число. Перегрузите одноместную операцию ! для этого класса, выводящую в зависимости от знака (+ или -) этого числа результат 0 (отрицательный) или 1 (положительный). Перегрузите также двухместную операцию + так, чтобы при сложении объекта с целым числом значения поля данных увеличивалось на это число, а при сложении объекта с другим объектом этого же класса поле данных уменьшалось.

Литература

Семакин И.Г., Шестаков А.П. Основы программирования: Учебник.- М.: Мастерство, 2002.- 432 с.