20. КЛАССЫ И ШАБЛОНЫ

Шаблон семейства классов определяет способ построения отдельных классов подобно тому, как класс определяет правила построения и формат отдельных объектов. Шаблон семейства классов определяется так:

template < список параметров шаблона > определение класса

В определении класса, входящего в шаблон, особую роль играет имя класса. Оно является не именем отдельного класса, а параметризованным именем семейства классов.

Определение шаблона может быть только глобальным.

Определим класс Vector, в число данных которого входит одномерный массив. Независимо от типа элементов этого массива в классе должны быть определены одни и те же базовые функции, например, функция умножения каждого компонента вектора на число и тому подобное. Если тип элементов вектора задать в качестве параметра шаблона класса, то система будет формировать вектор нужного типа и соответствующий класс при каждом определении конкретного объекта.

```
// Файл vec.cpp
template < class T >          // T – параметр шаблона;
class Vector{ public:

T* pv;                        // одномерный массив;
int sz;  public:              // размер массива.
int size(){
return sz;  }


Vector<T>(int);               // Конструктор
~Vector(){  delete
[]pv; } void
umn(T n){
for (int i = 0; i < sz; i++ ) pv[i] *= n;}
};
template < class T >
Vector < T >::Vector(int n){// Конструктор
pv = new T[n]; sz = n; }
// Конец файла vec.cpp
```

Когда шаблон введен, появляется возможность определить конкретные объекты конкретных классов, каждый из которых параметрически порожден из шаблона. Формат определения объекта одного из классов, порожденного шаблоном, следующий:

имя параметризованного класса < фактические параметры шаблона > имя объекта (параметры конструктора).

В нашем случае определить вектор, имеющий 100 компонент типа double, можно так:

```
Vector < double > array( 100 );
```

Программа, использующая шаблон Vector, может выглядеть так:

```
#include <iostream>
#include <iomanip>          //Для использования манипулятора setw()
#include "vec.cpp"
using namespace std;
void main(){ Vector
< int > x(5);
Vector < double > c(5); for
(int i = 0; i < 5; i++){
x.pv[i] = i;
c.pv[i] = i * i; } cout <<
setw(3) << "x:" << '\n';
for (int i = 0; i < 5; i++) cout << setw(6) << x.pv[i];
cout << '\n'; x.umn(2); for (int i = 0; i < 5; i++)
cout << setw(6) << x.pv[i]; cout << '\n'; cout <<
setw(3) << "c:" << '\n';
for (int i = 0; i < 5; i++) cout << setw(6) << c.pv[i];
cout << '\n'; c.umn(0.5); for (int i = 0; i < 5; i++)
cout << setw(6) << c.pv[i]; cout << '\n';
}
```

Результат:

```
x:
    0     1     2     3     4
0     2     4     6     8 c:
    0     1     4     9    16
```

0	0.5	2	4.5	8

В списке параметров шаблона могут присутствовать формальные переменные, не определяющие тип. Точнее, это параметры, для которых тип фиксирован:

```
template < class T, int size = 64 >
class row { T *
data; int length;
public: row(){
length = size; data
= new T[ size];
}
~row(){  delete
T[] data;  }
…
};
```

В качестве фактического аргумента для параметра size взята константа. В общем случае может быть использовано константное выражение, однако использовать выражения, содержащие переменные, нельзя.

Отметим, что C++ всегда предоставляется вместе со стандартной библиотекой, которую можно (и нужно) использовать при написании программ. Стандартная библиотека C++ представляет собой большой набор функций и классов для решения различных задач. Например, многие важные вещи, в частности ввод/вывод, находятся в стандартной библиотеке.

Частью стандартной библиотеки C++ является библиотека стандартных шаблонов (Standard Template Library, STL). Функциональные особенности стандартной библиотеки объявляются внутри пространства имен std. Стандартная библиотека шаблонов – подмножество стандартной библиотеки C++ и содержит контейнеры, алгоритмы, итераторы, объекты-функции и т. д.

Очень полезным и популярным шаблонным классом из STL является шаблон std::vector.

std::vector (или просто «вектор») – это динамический массив, который может сам управлять выделенной себе памятью. Это означает, что можно создавать массивы, длина которых задаётся во время выполнения, без использования операций new и delete. std::vector находится в заголовочном файле vector.

Рассмотрим пример с использованием STL.

```
#include <iostream> #include
<vector>
using namespace std;

void main(){ int
n;
cout << "Задайте длину вектора:\n";
cin >> n;  vector < int >
x(n); vector < char >
c(n); for (int i = 0; i < n;
i++){
x[i] = i;  c[i]
= 'A' + i;
} for (int i = 0; i < n; i++) cout <<
"  " << x[i] << "  " << c[i]; cout <<
"\n"; }
```

Результат:

```
0  A  1  B  2  C  3  D  4  E
```