## Шаблонный класс vector

Шаблонный класс vector похож на класс string в том, что он является динамическим массивом. Установить размер объекта vector можно во время выполнения, и можно добавлять новые данные в конец или вставлять их в середину, В основном

vector представляет собой альтернативу применению операции new для создания динамического массива. На самом деле класс vector использует операции new и delete для управления памятью, но делает это автоматически.

На данный момент мы не планируем глубоко погружаться в то, что собой представляет шаблонный класса. Вместо этого мы рассмотрим несколько базовых практических вопросов.

* Во-первых, чтобы можно было работать с объектом vector,

понадобится включить заголовочный файл vector.

* Во-вторых, идентификатор vector является частью пространства имен std, поэтому придется использовать директиву using, объявление using или запись std: :vector.
* В-третьих, шаблоны применяют другой синтаксис для указания типа сохраненных данных.
* В-четвертых, класс vector использует отличающийся синтаксис для указания количества элементов.

Ниже показаны некоторые примеры:

*#include <vector>*

*using namespace std;*

*vector<int> vi; // создание массива int нулевого размера*

*int n;*

*cin >> n;*

*vector<double> vd(n); // создание массива из п элементов double*

На основе этого кода можно сказать, что ѵі — это объект типа vector<int>, a vd — объект типа vector<double>. Поскольку объекты vector изменяют свои размеры автоматически при вставке или добавлении значений к ним, вполне нормально

для ѵі начать с размера 0. Но чтобы изменение размера работало, необходимо применять разнообразные методы, входящие в состав пакета vector.

**В общем,** следующее объявление создает объект vector по имени vt, который может хранить *количество\_элементов* элементов типа *имяТипа:*

*ѵесtor<имяТипа> vt (количество\_элементов) ;*

Параметр количество\_элементов может быть целочисленной константой или целочисленной переменной.

# Шаблонный класс array (C++11)

Класс vector обладает большими возможностями, чем встроенный тип массива, но достигается это ценой некоторого снижения эффективности. Если все, что требуется — это массив фиксированного размера, может быть выгоднее использовать встроенный тип. Однако при этом снижается степень удобства и безопасности. С++11 реагирует на эту ситуацию добавлением шаблонного класса array, который является частью пространства имен std. Подобно встроенному типу, объект array имеет фиксированный размер и использует стек (или распределение в статической памяти)

вместо свободного хранилища, поэтому он характеризуется эффективностью встроенных массивов. К этому добавляется удобство и безопасность. Для создания объекта array должен быть включен заголовочный файл array. Используемый синтаксис несколько отличается от такового для vector:

*#include <array>*

*using namespace std;*

*array<int, 5> ai; // создание объекта array из пяти элементов int*

*array<double, 4> ad = {1.2, 2.1, 3.43. 4.3};*

**В общем** случае следующее объявление создает объект array по имени агг, который может хранить *количество\_элементов* элементов типа *имяТипа:*

*array<имяТипа, количество\_элементов> агг;*

В отличие от vector, количество\_элементов не может быть переменной.

В С++11 можно применять списковую инициализацию для объектов vector и array. Тем не менее, это не доступно для объектов vector в С++98.

Сравнение массивов, объектов vector и объектов array

Проще всего понять сходства и различия между массивами, объектами vector и объектами array, рассмотрев пример (листинг 4.24), в котором используются все три подхода.

**Листинг 4.24. choices. срр**

*// choices.срр -- вариации массивов*

*#include <iostream>*

*#include <vector> // STL C++98*

*#include <array> // C+ + 11*

*int main ()*

*{*

*using namespace std;*

*// С, исходный C++*

*double al[4] = {1.2, 2.4, 3.6, 4.8};*

*// C++98 STL*

*vector<double> a2(4); // создание объекта vector с четырьмя элементами*

*// Простой способ инициализации в С98 отсутствует*

*а2[0] = 1.0/3.0;*

*' а2[1] = 1.0/5.0;*

*а2[2] = 1.0/7.0;*

*а2[3] = 1.0/9.0;*

*// С++11 -- создание и инициализация объекта array*

*array<double, 4> аЗ = {3.14, 2.72, 1.62, 1.41};*

*array<double, 4> а4;*

*а4 = аЗ; // допускается для объектов array одного и того же размера*

*// Использование нотации массивов*

*cout « "al[2]: " « al[2] « " at " « &al[2] << endl;*

*cout « na2[2]: " « a2[2] « " at " « &a2[2] << endl;*

*cout « Ma3[2]: " « a3[2] « " at " « &a3[2] « endl;*

*cout « na4[2]: " « a4[2] « " at " « &a4[2] « endl;*

*// Преднамеренная ошибка*

*al[-2] = 20.2;*

*cout « "al[-2]: " « al[-2] <<" at " « &al[-2] « endl;*

*cout « Ma3[2]: " « a3[2] << " at " « &a3[2] « endl;*

*cout « "a4[2]: " « a4[2] « " at " « &a4[2] « endl;*

*return 0;*

*}*

**Ниже показан пример вывода:**

*al[2]: 3.6 at 0x28cce8*

*a2[2]: 0.142857 at 0xca0328*

*a3[2] : 1.62 at 0x28ccc8*

*a4 [2] : 1.62 at 0x28cca8*

*al[-2]: 20.2 at 0x28ccc8*

*a3[2] : 20.2 at 0x28ccc8*

*a4[2]: 1.62 at 0x28cca8*

***Замечания по программе***

Во-первых, обратите внимание, что независимо от применяемого подхода —встроенного массива, объекта vector или объекта array — мы можем использовать стандартную нотацию массивов для доступа к индивидуальным членам. Во-вторых, по адресам легко заметить, что объекты array находятся в той же самой области памяти (в данном случае — в стеке), что и встроенный массив, тогда как объект vector, хранится в другой области (в свободном хранилище, или куче). В-третьих, в коде показано, что один объект array можно присвоить другому объекту array. В случае встроенных объектов понадобится поэлементно копировать данные. Далее обратите особое внимание на следующую строку:

al[-2] = 20.2;

Что означает индекс -2? Вспомните, что эта запись транслируется в такой код:

*\*(al-2) = 20.2*;

Выразить словами это можно так: посмотреть, на что указывает аі, переместиться на два элемента double назад и поместить туда значение 20.2. То есть сохранить информацию в позиции за пределами массива. В этом конкретном случае данной

позицией оказывается объект array по имени аЗ. Другой компилятор поместит 20.2 в а4, а прочие могут предпринять еще какие-нибудь неверные действия.

Это пример небезопасного поведения встроенных массивов. Защищают ли объекты vector и array от такого поведения? Да, они могут, если вы им позволите. То есть вы по-прежнему можете писать небезопасный код вроде такого:

*а2[-2] = .5; // по-прежнему разрешено*

*аЗ[200] = 1.4;*

Однако существуют альтернативы. Одна из них предполагает применение функции-члена at (). Точно так же, как вы можете использовать функцию-член getline () с объектом сіп, вы можете применять функцию-член at() с объектами vector и

array: а2.at (1) = 2 . 3; // присваивает а2[1] значение 2.3 Отличие между использованием нотации с квадратными скобками и вызовом функции-члена at () состоит в том, что в случае at () указание недопустимого индекса во время выполнения по умолчанию приводит к аварийному завершению программы. За счет такой дополнительной проверки увеличивается время выполнения; именно поэтому в C++ доступен на выбор один из этих вариантов. Более того, классы vector и array предлагают способы использования объектов, которые снижают вероятность появления непредвиденных ошибок диапазона. Например, эти классы имеют функции-члены begin () и end (), позволяющие установить границы диапазона без случайного выхода за их пределы.