Векторы

- *Векторы представляют собой динамические массивы.*
- Класс *vector* поддерживает динамический массив, который при необходимости может увеличивать свой размер.В C++ размер массива фиксируется во время компиляции. И хотя это самый эффективный способ реализации массивов, он в то же время является и самым ограничивающим, поскольку размер массива нельзя изменять во время выполнения программы. Эта проблема решается с помощью вектора, который по мере необходимости обеспечивает выделение дополнительного объема памяти. Несмотря на то что вектор — это динамический массив, тем не менее, для доступа к его элементам можно использовать стандартное обозначение индексации массивов.

template <class T, class Allocator = allocator<T> >

class vector

Здесь *T*— тип сохраняемых данных, а элемент *Allocator* означает распределитель памяти, который по умолчанию использует стандартный распределитель. Класс *vector* имеет следующие конструкторы.

explicit vector(const Allocator &a = Allocator());

explicit vector(size_type num, const T &val = T(), const Allocator &a = Allocator());

vector(const vector<T, Allocator> &ob);

template <class InIter> vector(InIter start, InIter end, const Allocator &a = Allocator());

Первая форма конструктора предназначена для создания пустого вектора. Вторая создает вектор, который содержит *num* элементов со значением *val*, причем значение *val* может быть установлено по умолчанию. Третья форма позволяет создать вектор, который содержит те же элементы, что и заданный вектор *ob*. Четвертая предназначена для создания вектора, который содержит элементы в диапазоне, заданном параметрами-итераторами *start* и *end*.

Ради достижения максимальной гибкости и переносимости любой объект, который предназначен для хранения в векторе, должен определяться конструктором по умолчанию. Кроме того, он должен определять операции "<" и "==" Некоторые компиляторы могут потребовать определения и других операторов сравнения. Все встроенные типы автоматически удовлетворяют этим требованиям.

- vector<int> iv; /* Создание вектора нулевой длины для хранения int-значений. */
- vector<char> cv(5); /* Создание 5-элементного вектора для хранения char-значений. */
- vector<char> cv(5, 'x'); /* Инициализация 5-элементного char-вектора. */
- vector<int> iv2(iv); /* Создание int-вектора на основе int-вектора iv. */

Для класса vector определены следующие операторы сравнения:

==, <, <=, !=, > и >=

Для вектора также определен оператор индексации "[]", который позволяет получить доступ к элементам вектора с помощью стандартной записи с использованием индексов. Функции-члены, определенные в классе *vector*, перечислены в табл. Самыми важными являются

size(), *begin()*, *end()*, *push_back()*, *insert()* и *erase()*.

Функция *size()* возвращает текущий размер вектора, она позволяет определить размер вектора во время выполнения программы.

Таблица 21.2. Функции-члены, определенные в классе `vector`

Функция-член	Описание
`template <class InIter> void assign(InIter start, InIter end);`	Помещает в вектор последовательность, определяемую параметрами *start* и *end*
`void assign(size_type num, const T &val);`	Помещает в вектор *num* элементов со значением *val*
`reference at(size_type i); const_reference at(size_type i) const;`	Возвращает ссылку на элемент, заданный параметром *i*
`reference back(); const_reference back() const;`	Возвращает ссылку на последний элемент в векторе
`iterator begin(); const_iterator begin() const;`	Возвращает итератор для первого элемента в векторе
`size_type capacity() const;`	Возвращает текущую емкость вектора, или количество элементов, которое может храниться в векторе до того, как возникнет необходимость в выделении дополнительной памяти

`void clear();`	Удаляет все элементы из вектора
`bool empty() const;`	Возвращает истинное значение, если используемый вектор пуст, и ложное значение в противном случае
`const_iterator end() const;` `iterator end();`	Возвращает итератор для конца вектора
`iterator erase(iterator i);`	Удаляет элемент, адресуемый итератором `i`; возвращает итератор для элемента, расположенного после удаленного
`iterator erase(iterator start,` `iterator end);`	Удаляет элементы в диапазоне, задаваемом параметрами `start` и `end`; возвращает итератор для элемента, расположенного за последним удаленным элементом
`reference front();` `const_reference front() const;`	Возвращает ссылку на первый элемент в векторе
`allocator_type get_allocator()` `const;`	Возвращает распределитель памяти вектора

`iterator insert(`
`iterator i,`
`const T &val = T());`
	Вставляет значение `val` непосредственно перед элементом, заданным параметром `i`; возвращает итератор для этого элемента

Окончание табл. 21.2

Функция-член	Описание
`void insert(iterator i, size_type num, const T &val);`	Вставляет `num` копий значения `val` непосредственно перед элементом, заданным параметром `i`
`template <class InIter> void insert( iterator i, InIter start, InIter end);`	Вставляет в вектор последовательность элементов, определяемую параметрами `start` и `end`, непосредственно перед элементом, заданным параметром `i`
`size_type max_size() const;`	Возвращает максимальное число элементов, которое может содержать вектор
`reference operator[](size_type i) const;` `const_reference operator[](size_type i) const;`	Возвращает ссылку на элемент, заданный параметром `i`
`void pop_back();`	Удаляет последний элемент в векторе
`void push_back(const T &val);`	Добавляет в конец вектора элемент, значение которого задано параметром `val`

`reverse_iterator rbegin();` `const_reverse_iterator rbegin() const;`	Возвращает реверсивный итератор для конца вектора
`reverse_iterator rend();` `const_reverse_iterator rend() const;`	Возвращает реверсивный итератор для начала вектора
`void reserve(size_type num);`	Устанавливает емкость вектора равной значению не менее заданного *num*
`void resize(size_type num, T val = T());`	Устанавливает размер вектора равным значению, заданному параметром *num*. Если вектор для этого нужно удлинить, то в его конец добавляются элементы со значением, заданным параметром *val*
`size_type size() const;`	Возвращает текущее количество элементов в векторе
`void swap(` `deque<T, Allocator> &ob);`	Выполняет обмен элементами вызывающего вектора и вектора *ob*

- Функция *begin()* возвращает итератор, который указывает на начало вектора. Функция *end()* возвращает итератор, который указывает на конец вектора. Как уже разъяснялось, итераторы подобны указателям, и с помощью функций *begin()* и *end()* можно получить итераторы для начала и конца вектора соответственно.
- Функция *push_back()* помещает заданное значение в конец вектора. При необходимости длина вектора увеличивается так, чтобы он мог принять новый элемент. С помощью функции *insert()* можно добавлять элементы в середину вектора. Кроме того, вектор можно инициализировать. В любом случае, если вектор содержит элементы, то для доступа к ним и их модификации можно использовать средство индексации массивов. А с помощью функции *erase()* можно удалять элементы из вектора.

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{  vector<int> v; // создание вектора нулевой длины
  unsigned int i;
  // Отображаем исходный размер вектора v.
  cout << "Размер = " << v.size() << endl;
  /* Помещаем значения в конец вектора, и размер вектора будет
     по необходимости увеличиваться. */
  for(i=0; i<10; i++) v.push_back(i);
  // Отображаем текущий размер вектора v.
  cout << "Текущее содержимое:\n";
  cout << "Новый размер = " << v.size() << endl;
  // Отображаем содержимое вектора.
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;
```

```
/* Помещаем в конец вектора новые значения, и размер вектора
     будет по необходимости увеличиваться. */
for(i=0; i<10; i++ ) v.push_back(i+10);
  // Отображаем текущий размер вектора v.
  cout << "Новый размер = " << v.size() << endl;
  // Отображаем содержимое вектора.
  cout << "Текущее содержимое:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;
  // Изменяем содержимое вектора.
  for(i=0; i<v.size(); i++) v[i] = v[i] + v[i];
  // Отображаем содержимое вектора.
  cout << "Содержимое удвоено:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;
  return 0;
}
```

Результаты выполнения этой программы таковы.

```
Размер = 0
Текущее содержимое:
Новый размер = 10
0 1 2 3 4 5 6 7 8 9
Новый размер = 20
Текущее содержимое:
0 1 2 3 4  5  6   7  8   9  10 11 12 13 14 15 16 17 18 19
Содержимое удвоено:
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38
```

```
// Доступ к вектору с помощью итератора.
#include <iostream>
#include <vector>
using namespace std;
int main()
{
  vector<char> v; // создание массива нулевой длины
  int i;
  // Помещаем значения в вектор.
  for(i=0; i<10; i++) v.push_back('A' + i);
  /* Получаем доступ к содержимому вектора с помощью индекса. */
  for(i=0; i<10; i++) cout << v[i] << " ";
  cout << endl;
  /* Получаем доступ к содержимому вектора с помощью итератора. */
  vector<char>:: iterator p = v.begin();
  while(p != v.end()) {
    cout << *p << " ";
    p++;
  }
  return 0;
}
```

Вот как выглядят результаты выполнения этой программы.

A B C D E F G H I J

A B C D E F G H I J

В этой программе сначала создается вектор нулевой длины. Затем с помощью функции *push_back()* в конец вектора помещаются символы, в результате чего размер вектора соответствующим образом увеличивается.

Обратите внимание на то, как объявляется итератор *p*. Тип этого итератора определяется контейнерными классами. В нашей программе итератор *p* инициализируется таким образом, чтобы он указывал на начало вектора (с помощью функции-члена *begin()*). Итератор, который возвращает эта функция, можно затем использовать для поэлементного доступа к вектору, инкрементируя его соответствующим образом. Этот процесс аналогичен тому, как можно использовать указатель для доступа к элементам массива. Чтобы определить, когда будет достигнут конец вектора, используется функция-член *end()*. Эта функция возвращает итератор, установленный за последним элементом вектора. Поэтому, если значение *p* равно *v.end()*, значит, конец вектора достигнут.

```
// Демонстрация вставки элементов в вектор и удаления их из него.
#include <iostream>
#include <vector>
using namespace std;
int main()
{  vector<char> v;
  unsigned int i;
  for(i=0; i<10; i++) v.push_back('A' + i);
  // Отображаем исходное содержимое вектора.
  cout << "Размер = " << v.size() << endl;
  cout << "Исходное содержимое вектора:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl << endl;
  vector<char>:: iterator p = v.begin();
  p += 2; // указатель на 3-й элемент вектора
  // Вставляем 10 символов 'X' в вектор v.
  v.insert(p, 10, 'X');
  /* Отображаем содержимое вектора после вставки символов. */
  cout << "Размер вектора после вставки = " << v.size()<< endl;
  cout << "Содержимое вектора после вставки:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl << endl;
```

```
  // Удаление вставленных элементов.
  p = v.begin();
  p += 2; // указатель на 3-й элемент вектора
  v.erase(p, p+10); // Удаляем 10 элементов подряд.
  /* Отображаем содержимое вектора после удаления символов. */
  cout << "Размер вектора после удаления символов = "<< v.size() << endl;
  cout << "Содержимое вектора после удаления символов:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;
  return 0;}
```

При выполнении эта программа генерирует следующие результаты.

```
Размер = 10
Исходное содержимое вектора:
A B C D E F G H I J
Размер вектора после вставки = 20
Содержимое вектора после вставки:
A B X X X X X X X X X X C D E F G H I J
Размер вектора после удаления символов = 10
Содержимое вектора после удаления символов:
A B C D E F G H I J
```

```
// Хранение в векторе объектов класса.
#include <iostream>
#include <vector>
using namespace std;
class three_d {
    int x, y, z;
  public:
    three_d() { x = y = z = 0; }
    three_d(int a, int b, int c) { x = a; y = b; z = c; }
    three_d &operator+(int a) {
      x += a;
      y += a;
      z += a;
      return *this;
    }
       friend ostream &operator<<(ostream &stream, three_d obj);
    friend bool operator<(three_d a, three_d b);
    friend bool operator==(three_d a, three_d b);
};
```

```
/* Отображаем координаты X, Y, Z с помощью
    оператора вывода для класса three_d. */
ostream &operator<<(ostream &stream, three_d obj)
{
  stream << obj.x << ", ";
  stream << obj.y << ", ";
  stream << obj.z << "\n";
  return stream;
}
bool operator<(three_d a, three_d b)
{  return (a.x + a.y + a.z) < (b.x + b.y + b.z);}
bool operator==(three_d a, three_d b)
{  return (a.x + a.y + a.z) == (b.x + b.y + b.z);}
```

```
int main()
{  vector<three_d> v;
  unsigned int i;
  // Добавляем в вектор объекты.
  for(i=0; i<10; i++)
    v.push_back(three_d(i, i+2, i-3));
// Отображаем содержимое вектора.
  for(i=0; i<v.size(); i++)
    cout << v[i];
  cout << endl;
  // Модифицируем объекты в векторе.
  for(i=0; i<v.size(); i++) v [i] = v[i] + 10;
  // Отображаем содержимое модифицированного вектора.
  for(i=0; i<v.size(); i++) cout << v[i];
  return 0;}
```

Эта программа генерирует такие результаты.
0, 2, -3
1, 3, -2
2, 4, -1
3, 5, 0
5, 7, 2
6, 8, 3
7, 9, 4
8, 10, 5
9, 11, 6
10, 12, 7
11, 13, 8
12, 14, 9
13, 15, 10
14, 16, 11
15, 17, 12
16, 18, 13
17, 19, 14
18, 20, 15
19, 21, 16

пользе итераторов

Частично сила библиотеки STL обусловлена тем, что многие ее функции используют итераторы. Этот факт позволяет выполнять операции с двумя контейнерами одновременно. Рассмотрим, например, такой формат векторной функции *insert()*.

template <class InIter>

void insert(iterator i, InIter start, InIter end);

Эта функция вставляет исходную последовательность, определенную параметрами *start* и *end*, в приемную последовательность, начиная с позиции *i*. При этом нет никаких требований, чтобы итератор *i* относился к тому же вектору, с которым связаны итераторы *start* и *end*. Таким образом, используя эту версию функции *insert()*, можно один вектор вставить в другой.

```
// Вставляем один вектор в другой.
#include <iostream>
#include <vector>
using namespace std;
int main()
{  vector<char> v, v2;
  unsigned int i;
  for(i=0; i<10; i++) v.push_back('A' + i);
  // Отображаем исходное содержимое вектора.
  cout << "Исходное содержимое вектора:\n";
  for(i=0; i<v.size(); i++)
    cout << v[i] << " ";
  cout << endl << endl;
  // Инициализируем второй вектор.
  char str[] = "-STL — это сила!-";
  for(i=0; str[i]; i++) v2 .push_back (str [i]);
```

```
  /* Получаем итераторы для середины вектора v, а
     также начала и конца вектора v2. */
     vector<char>::iterator p = v.begin()+5;
  vector<char>::iterator p2start = v2.begin();
  vector<char>::iterator p2end = v2.end();
  // Вставляем вектор v2 в вектор v.
  v.insert(p, p2start, p2end);
  // Отображаем результат вставки.
  cout << "Содержимое вектора v после вставки:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  return 0;
}
```

При выполнении эта программа генерирует следующие результаты.

Исходное содержимое вектора:

A B C D E F G H I J

Содержимое вектора v после вставки:

A B C D E - S T L -- это сила! - F G H I J

Как видите, содержимое вектора *v2* вставлено в середину вектора *v*.