Шаблон vector

Вектор в библиотеке STL — это удобный массив (вернее, структура, аналогичная массиву), размер которого может изменяться динамически. Вектор целых чисел `A` задается таким образом:

```
vector <int> A;
```

Аналогично можно задавать векторы любого другого типа.

Вектор поддерживает операцию обращения по индексу `A[i]`, как и обычный массив, но в отличии от обычного массива размер вектора может меняться. У вектора есть метод `size`, возвращающий количество элементов в векторе в настоящий момент.

Есть несколько способов создать вектор (конструкторы вектора): `vector(B)`, где `B` — вектор такого же типа. Создает копию вектора `B` `vector(n)`, где `n` — целое число. Создает вектор из `n` элементов. `vector(n, val)` — создает вектор из `n` элементов, заполненных значением `val`.

Например:

```
vector <int> A(10, 179);
vector <int> B(A);
```

создаст вектор `A` из 10 элементов равных 179, а затем вектор `B`, являющийся копией вектора `A`.

Конструкторы можно вызывать явно для создания объектов, например, `vector<int>(10, 179)` создаст объект типа `vector <int>` и вызовет для вновь создаваемого объекта конструктор.

Для изменения вектора можно использовать метод `resize`. Первый его параметр — новый размер вектора, второй параметр (необязательный, имеет смысл только при увеличении размера вектора) — значение, которым заполняются вновь созданные элементы.

Для добавления нового элемента в конец вектора можно использовать метод `push_back(val)`. Для удаления последнего элемента из вектора можно использовать метод `pop_back()`.

Из вектора `A` можно удалить элемент с индексом `i` при помощи `A.erase(A.begin() + i)`. Для удаления элементов с `i`-го (включая) до `j`-го (не включая) можно использовать `A.erase(A.begin() + i, A.begin() + j)`. Также можно использовать итератор `end()`, возвращающий элемент на конец вектора (фиктивный элемент, следующий за последним). Например, удалить элементы с `i`-го до конца можно при помощи `A.erase(A.begin() + i, A.end())`, а удалить `k` последних элемента вектора можно при помощи `A.erase(A.end() - k,`

`A.end()`. Удаление требует сдвига элементов, поэтому выполняется за линейное время.

Вставить значения в вектор можно при помощи метода `insert`. Например, вставить элемент `val` перед `i`-м элементов вектора `A` можно при помощи `A.insert(A.begin() + i, val)`. Если же передать в качестве указателя два итератора, то можно вставить весь фрагмент между итераторами. Например, `A.insert(A.begin(), B.begin(), B.end())` вставляет в начало вектора `A` все содержимое вектора `B`.

Более подробно обо всех методах работы с векторами можно прочесть на cppreference.com.

Операции со строками в STL

В этом листке мы снова будем работать со строками, активно используя стандартную библиотеку языка C++ STL (Standard template library).

Для прочтения рекомендуется следующий раздел сайта www.cppreference.com.

Считывание строк

Напомним, что строки можно считывать двумя способами: до пробельного символа (пробела, конца строки, символа табуляции) при помощи конструкции `cin >> S`, и до конца строки при помощи функции `getline(cin, S)`.

Арифметические операторы

Со строками можно выполнять следующие арифметические операции:
`=` - присваивание значения.
`+=` - добавление в конец строки другой строки или символа.
`+` - конкатенация двух строк, конкатенация строки и символа.
`==`, `!=` - посимвольное сравнение.
`<`, `>`, `<=`, `>=` - лексикографическое сравнение.

Подробней об операторах для строк читайте здесь.

Конструкторы

Строки можно создавать с использованием следующих конструкторов:
`string()` - конструктор по умолчанию (без параметров) создает пустую строку.
`string(string & S)` - копия строки `S`
`string(int n, char c)` - повторение символа `c` заданное число `n` раз.
`string(char c)` - строка из одного символа `c`.
`string(string & S, int start, int len)` - строка, содержащая не более, чем `len` символов данной строки `S`, начиная с символа номер `start`.

Конструкторы можно вызывать явно, например, так:

```
S += string(10, 'z');
```

В этом примере явно вызывается конструктор `string` для создания строки, состоящей из 10 символов `'z'`.

Неявно конструктор вызывается при объявлении строки с указанием дополнительных параметров. Например, так:

```
string S(10, 'z');
```

Подробней о конструкторах для строк читайте здесь.

Методы для строк

Методом называется функция, которая применяется к объекту, например, строке. При вызове метода его имя пишется после идентификатора объекта через точку, например, у объекта типа `string` есть метод `size`, возвращающий длину строки. Если `S` — это строка, то метод вызывается так: `S.size()`. Другой пример: метод `substr` возвращает подстроку заданной строки:

```
string S = "Hello, world!";
cout << S.substr(6, 5) << endl;
```

В данном случае будет выведен текст `world`.

Вот список полезных методов, применяемых к строкам:

append	добавляет строку или символы к строке
assign	присваивает строке значение строк символов или других строк C++
clear	удаляет все символы из строки
compare	сравнивает две строки
empty	возвращает true если в строке нет символов
erase	удаляет символы из строки
find	ищет символы в строке
find_first_not_of	находит первый символ, отличный от
find_first_of	находит первый символ схожий с
find_last_not_of	находит последний символ, отличный от
find_last_of	находит последний символ, схожий с
insert	вставляет символы в строку
length	возвращает длину строки

npos	специальное значение, означающее «не найдено» или «все оставшиеся символы»
push_back	добавляет символ в конец строки
replace	заменяет символы в строке
resize	меняет размер строки
rfind	находит последнее вхождение подстроки
size	возвращает количество символов в строке
substr	возвращает определённую подстроку
swap	меняет две строки содержимым

Для обозначения того, что запрашиваемая последовательность символов не найдена, многие методы (например, `find`) возвращают специальную константу `string::npos`.