**Лабораторная работа №18.**

**Динамические структуры данных. Вектор**

**Вектор** — это структура данных, которая уже является моделью динамического массива.

**Как создать вектор (vector) в C++**

Для создания вектора необходимо подключить библиотеку — <vector>, в ней хранится шаблон вектора.

|  |  |
| --- | --- |
|  | #include <vector> |

Чтобы объявить вектор, нужно пользоваться конструкцией ниже:

|  |  |
| --- | --- |
|  | vector < тип данных > <имя вектора>; |

* Вначале пишем слово vector.
* Далее в угольных скобках указываем тип, которым будем заполнять ячейки.
* В конце указываем имя вектора.

Вот пример:

|  |  |
| --- | --- |
|  | vector <string> ivector; |

В примере выше мы создали вектор строк.

заполнить вектор можно еще при инициализации. Делается это также просто, как и в массивах:

|  |  |
| --- | --- |
| 1 | vector <int> ivector = {0, 1, 2}; |

**Обращение к ячейке**

В векторе для обращения к ячейке используются индексы. Обычно мы их используем совместно с квадратными скобками [].

Но в C++ есть еще один способ это сделать благодаря функции — **at()**. В скобках мы должны указать индекс той ячейки, к которой нужно обратиться.

Пример:

|  |  |
| --- | --- |
| 1  2  3  4  5 | vector <int> ivector = {1, 2, 3};    ivector.at(1) = 5; // изменили значение второго элемента    cout << ivector.at(1); // вывели его на экран |

Давайте запустим эту программу:

fuctsiya\_at.cpp

**5**

**Process returned 0 (0x0) execution time : 0.010 s**

**Press any key to continue.**

**Размер вектора**

Указывать размер вектора можно по-разному. Можно это сделать еще при его инициализации, а можно в программе. Вот, например, способ указать длину вектора на старте:

|  |  |
| --- | --- |
|  | vector <int> vector\_first(5); |

Так в круглых скобках () после имени вектора указываем первоначальную длину. А вот второй способ:

|  |  |
| --- | --- |
| 1  2 | vector <int> vector\_second; // создали вектор  vector\_second.reserve(5); // указали число ячеек |

Во второй строке присутствует reserve, это функция, с помощью которой резервируется нужное количество ячеек.

В чем отличие первого и второго способа?

Создадим два вектора и по-разному укажем их количество ячеек.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | #include <iostream>  #include <vector> // подключили библиотеку    using namespace std;    int main() {  setlocale(0, "");    vector <int> vector\_first(3); // объявили  // два  vector <int> vector\_second; // вектора  vector\_second.reserve(3);    cout << "Значения первого вектора (с помощью скобок): ";    for (int i = 0; i < 3; i++) {  cout << vector\_first[i] << " ";  }    cout << "Значения второго вектора (с помощью reserve): " << endl;    for (int i = 0; i < 3; i++) {  cout << vector\_second[i] << " ";  }    system("pause");  return 0;  } |

Результат:

reserve.cpp

**Значения первого вектора (с помощью скобок): 0 0 0**

**Значения второго вектора (с помощью reserve): 17 0 0**

**Process returned 0 (0x0) execution time : 0.010 s**

**Press any key to continue.**

В первом случае были получены три нуля, а во втором: 17, 0, 0.

При использовании первого способа все ячейки автоматически заполнились нулями.

При объявлении чего-либо (массива, вектора, переменной и т.д) мы выделяем определенное количество ячеек памяти, в которых уже хранится ненужный для ПК *мусор*. В нашем случае этим мусором являются числа.

Поэтому, когда мы вывели второй вектор, в нем уже находились случайные числа — 17, 0, 0. Обычно они намного больше.

Нужно помнить! При использовании второго способа есть некоторый плюс — по времени. Так как для первого способа компилятор тратит время, чтобы заполнить все ячейки нулями.

**Сравнение векторов**

Чтобы сравнить два вектора, потребуется применить оператор ветвления [if](https://codelessons.ru/cplusplus/lessons/operator-if-else-in-cpp.html).

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | if (vec\_first == vec\_second) { // сравнили!  cout << "Они равны!";  }  else {  cout << "Они не равны";  } |

**Вектор векторов**

В задачах, где требуется запись значений в двумерный массив можно использовать вектора (массив векторов).

|  |  |
| --- | --- |
| 1 | vector < vector < тип данных > >; |

Чтобы указать количества векторов в векторе нам потребуется — метод resize().

|  |  |
| --- | --- |
| 1  2 | vector < vector <int> > vec;  vec.resize(10); // десять векторов |

Но есть еще одни способ добавления векторов в вектор. Для этого способа мы будем использовать функцию push\_back() .

|  |  |
| --- | --- |
| 1 | vec.push\_back(vector <int>()); |

* В аргументах функции push\_back() находится — имя контейнера который необходимо добавить. В нашем случае — vector.
* Далее идет тип контейнера — <тип>.

Для двумерного вектора также можно указать значения еще при инициализации:

|  |  |
| --- | --- |
| 1  2  3 | vector < vector <int> > ivector = {{1, 4, 7},  {2, 5, 8},  {3, 6, 9}}; |
|  |  |

{1, 4, 7} — это значения элементов первого массива (первого слоя). Такие блоки значений {1, 4, 7} должны разделяться запятыми.

**Методы для работы с вектором:**

— это функции, которые относится к STL контейнеру vector.

**1) size() и empty()**

Если нам требуется узнать длину вектора, понадобится функция — size(). Эта функция практически всегда используется вместе с циклом for.

|  |  |
| --- | --- |
| 1  2  3 | for (int i = 0; i < ivector.size(); i++) {  // ...  } |

Также, если нам требуется узнать пуст ли вектор, мы можем использовать функцию — empty().

* При отсутствии в ячейках какого-либо значения это функция возвратит — true.
* В противном случае результатом будет — false.

|  |  |
| --- | --- |
| 1  2  3 | if (ivector.empty()) {  // ...  } |

**2) push\_back() и pop\_back()**

Одни из самых востребованных функций контейнера vector это — push\_back() и pop\_back().

* push\_back() – добавляет ячейку в конец вектора.
* pop\_back()— удаляет ячейку в конце вектора.

Использование функции push\_back() без указания значения ячейки — невозможно. Так или иначе, придется это значение указать!

Пример:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | #include <iostream>  #include <vector>    using namespace std;    int main() {  setlocale(0, "");    vector <string> vec\_string;    vec\_string.push\_back("апельсин"); // добавляем  vec\_string.push\_back("груша"); // четыре  vec\_string.push\_back("яблока"); // элемента  vec\_string.push\_back("вишня"); // в конец вектора    for (int i = 0; i < vec\_string.size(); i++) {  cout << vec\_string[i] << " ";  }    cout << endl;    vec\_string.pop\_back(); // удаляем элемент    for (int i = 0; i < vec\_string.size(); i++) {  cout << vec\_string[i] << " ";  }  system("pause");  return 0;  } |

Результат:

push\_back\_and\_pop\_back.cpp

**апельсин груша яблоко вишня**

**апельсин груша яблоко**

**Process returned 0 (0x0) execution time : 0.010 s**

**Press any key to continue.**

Можно значение удаляемого элемента записать в переменную:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | vec\_string.push\_back("апельсин");  vec\_string.push\_back("груша");    string val = vec\_string.pop\_back(); // присвоили значение удаляемой ячейке    cout << val; |

Результат:

push\_back.cpp

**груша**

**Process returned 0 (0x0) execution time : 0.010 s**

**Press any key to continue.**

**3) insert()**

— позволяет добавить новый элемент в произвольное место вектора.

|  |
| --- |
| <имя вектора>.insert(<итератор>, <значение>); |

<итератор> — указатель на элемент контейнера:

* begin() — указывает на начала вектора.
* end() — указывает на конец вектора.

<значение> —значение добавляемой ячейки.

Если в <итератор> указали именно итератор, а не функции begin() и end(), то элемент будет добавлен именно после той ячейки, на которую указывает итератор.

Пример:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | int main() {  setlocale(0, "");    vector vec\_string(2);    vec\_string[0] = 2; // заполнили две  vec\_string[1] = 3; // ячейки    for (int i = 0; i < vec\_string.size(); i++) {  cout << vec\_string[i] << " ";  }    cout << endl;  vec\_string.insert(vec\_string.begin(), 1); // добавили элемент в начало    for (int i = 0; i < vec\_string.size(); i++) {  cout << vec\_string[i] << " ";  }    cout << endl;  vec\_string.insert(vec\_string.end(), 4); // добавили элемент в конец    for (int i = 0; i < vec\_string.size(); i++) {  cout << vec\_string[i] << " ";  }    system("pause");  return 0;  } |

Результат:

insert.cpp

**2 3**

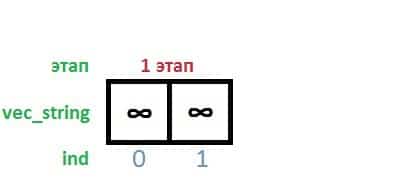
**1 2 3**

**1 2 3 4**

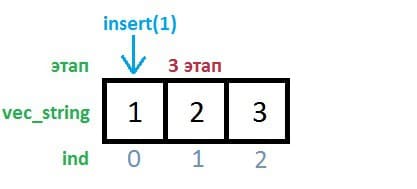
**Process returned 0 (0x0) execution time : 0.010 s**

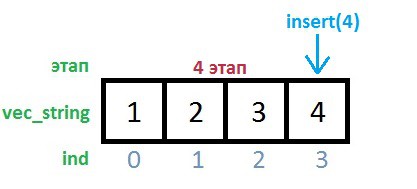
**Press any key to continue.**

На иллюстрациях ниже показано, как vec\_string изменялся в программе:









Вы должны знать! Если вы хотите, чтобы ваша программа работала как можно быстрее — вам нужно добавлять элементы именно в конец.

Так как при добавлении элемента в начало, с помощью той же функции insert(), в векторе происходит смещение всех ячеек вправо. К тому же, смещение идет по одной ячейке, а это у нас линейный поиск. При этом, чем больше вектор, тем медленнее будет происходить добавление элементов!

**4) front() и back()**

Позволяют просматривать первую и последнюю ячейки вектора:

front() и back().

|  |  |
| --- | --- |
| 1  2 | int front\_num = ivector.front();  int back\_num = ivector.back(); |

Варианты заданий

1. Создать вектор из символов. Удалить из вектора первый элемент, код которого меньше 48. Вставить символ % после каждой цифры.
2. Создать вектор из вещественных чисел. Вставить первый положительный элемент вектора после каждого отрицательного числа (если такого нет, оставить вектор без изменения).
3. Создать вектор из целых чисел. Вставить в вектор последний четный элемент после каждого нечетного элемента.
4. Создать вектор из целых чисел. Вставить в вектор число 11 после каждого элемента, равного 9.
5. Создать вектор из вещественных чисел. Удалить из вектора элемент перед каждым элементом со значением в интервале от 10 до 20.
6. Создать вектор из символов. Удалить из вектора элемент после каждого символа &.
7. Создать вектор из вещественных чисел. Вставить в вектор число 13.5 после первого элемента со значением большим 2.
8. Создать вектор из вещественных чисел. Определить среднее значение элементов вектора со значениями меньше либо равными 15. Удалить из вектора элементы, которые больше 25.
9. Создать вектор из целых чисел. Удалить из вектора первый элемент, больший числа 4. Вставить в вектор число 10 перед каждым числом, равным 15.
10. Создать вектор из вещественных чисел. Вставить в вектор первый отрицательный элемент перед каждым числом, равным 20 (если таких нет, оставить вектор без изменения).
11. Создать вектор из целых чисел. Удалить из вектора каждый элемент, кратный трем. Вставить в вектор число 88 после каждой пары равных рядом стоящих чисел.
12. Создать вектор из целых чисел. Вставить в вектор число 25 перед каждым элементом с положительным значением. Удалить из вектора все отрицательные числа.
13. Создать вектор из символов. Удалить из вектора элемент перед каждым символом ^. Определить количество символов \* в списке.
14. Создать вектор из вещественных чисел. Удалить из вектора элементы, у которых дробная часть больше 0,5.
15. Создать вектор из целых чисел. Вставить в вектор число 0 перед каждым числом от 2 до 7. Определить сумму чисел больших 7.
16. Создать вектор из строк. Удалить из вектора все строки, длина которых меньше 5 символов. После каждой оставшейся строки добавить её копию в верхнем регистре (используйте std::transform и std::toupper).
17. Создать вектор из структур, каждая из которых содержит имя и возраст человека. Отсортировать вектор по возрасту (по возрастанию). Удалить все элементы, возраст которых превышает 60 лет.
18. Создать вектор из целых чисел. Разделить вектор на два новых вектора: один содержащий только четные числа, а другой – только нечетные.
19. Создать вектор из вещественных чисел. Заменить каждый элемент вектора на его квадратный корень (используйте std::transform). Если какой-то элемент отрицательный, заменить его на 0.
20. Создать вектор из символов. Заменить каждую гласную букву на символ '\*'. (Учитывайте как строчные, так и прописные буквы).
21. Создать вектор из вещественных чисел. Найти минимальный и максимальный элементы вектора. Заменить все элементы, находящиеся между минимальным и максимальным (не включая их), на среднее арифметическое минимального и максимального.
22. Создать вектор из строк. Найти самую длинную строку в векторе. Удалить все строки, которые являются подстроками самой длинной строки.
23. Создать вектор из целых чисел. Добавить перед каждым элементом, который делится на 5 без остатка, среднее арифметическое всех элементов вектора.
24. Создать вектор из структур, содержащих название товара и его цену. Увеличить цену каждого товара на 10%, если его название начинается с буквы 'A' (регистр не важен).
25. Создать вектор из целых чисел. Сгруппировать все положительные числа в начале вектора, а все отрицательные – в конце (не меняя относительный порядок внутри групп).
26. Создать вектор из строк. Инвертировать каждую строку в векторе (использовать std::reverse).
27. Создать вектор из вещественных чисел. Найти второй по величине элемент в векторе.
28. Создать вектор из целых чисел. Удалить дубликаты из вектора (оставить только уникальные элементы, сохранив порядок).