**Цикл for и массивы**

На сегодняшнем занятии мы изучим цикл for, а также научимся работать с массивами чисел, которые позволяют хранить в себе последовательность.

**Сокращённая запись арифметических операций**

При написании программ часто требуется прибавить что-либо к текущему значению переменной. Например, при написании циклов while мы на каждом шаге прибавляли к счётчику единицу. Если название переменной длинное, то писать его несколько раз в одном арифметическом выражении неэкономно и неудобно. Чтобы избежать этого, была придумана сокращенная запись арифметических операций.

Например, мы хотим прибавить к переменной x единицу. Мы уже умеем записывать это как x = x + 1. То же самое можно написать в сокращенной записи как x += 1. Знак += означает, что к значению переменной, которая находится слева, будет прибавлен результат вычисления арифметического выражения, который находится справа. По аналогии с += существуют операции -=, \*=, /=. Они определены и для целых, и для вещественных чисел. Для целых чисел также определена операция %= для подсчёта остатка.

Поскольку задача прибавить или вычесть единицу возникает очень часто, то для её записи существует еще более короткая форма. Чтобы увеличить значение переменной x на единицу, можно написать x++. Если же мы хотим вычесть единицу, то писать нужно x−−. Именно в честь этой операции язык называется C++.

**Цикл for**

В языке C++ for является более короткой записью цикла while, которую удобно применять в некоторых случаях. Допустим, нам нужно вывести все числа от 1 до 100. Это несложно сделать с помощью цикла while:

int i;

i = 1;

while (i <= 100) {

    cout << i << " ";

    i++;

}

Однако намного проще сделать это с помощью цикла for:

int i;

for (i = 1; i <= 100; i++) {

    cout << i << " ";

}

После слова for в скобках пишутся три выражения:

1. Начальное значение счетчика (то, что выполняется непосредственно перед while).

2. Условие, при истинности которого выполняется цикл (абсолютно то же самое, что и в цикле while).

3. Изменение счетчика (то, что делается самой последней операцией в цикле while).

Выражения разделяются точкой с запятой. В принципе, в качестве выражений может выступать что угодно: первое из них выполнится один раз до начала цикла, второе будет проверяться на каждом шаге, а третье — выполняться каждый раз после того, как выполнены все действия в теле цикла.

**Векторы**

На прошлых занятиях мы уже научились обрабатывать последовательности, при этом не запоминая их целиком. Но такое возможно не во всех задачах. Например, на практических занятиях мы научились упорядочивать два и даже три числа по возрастанию. Если бы нас попросили написать программу, упорядочивающую сто тысяч чисел, то у нас на это ушло бы довольно много времени. Поэтому было бы здорово научиться хранить последовательности элементов и работать с ними в независимости от их количества.

В языке C++ есть несколько способов хранить последовательности чисел, и мы рассмотрим только один из них — vector. Он назван в честь математического термина «вектор», который обозначает занумерованную последовательность объектов.

Возвращаясь к идее «коробочек» для хранения чисел, вектор можно представить как выложенные в ряд коробочки, на каждой — одинаковое название и порядковый номер (он также называется индекс). Нумерация коробочек идёт с нуля и без пропусков.

Чтобы создать вектор, нам потребуется подключить библиотеку vector, написав в начале программы:

#include <vector>

Рассмотрим пример: задано число N — количество элементов последовательности, а затем N целых чисел Ai — это члены последовательности. Необходимо развернуть и вывести эту последовательность чисел.

int n;

cin >> n;

vector <int> a(n);

//считывание

for (int i = 0; i < n; i++) {

    cin >> a[i];

}

//обработка и вывод

for (int i = n - 1; i >= 0; i--) {

    cout << a[i] << " ";

}

Рассмотрим новый код подробнее. Строка vector <int> a(n); создаёт новый вектор. Сначала пишется слово vector, затем тип элементов вектора в угловых скобках (int или double), затем имя вектора и в круглых скобках — количество элементов в нём. Считать вектор целиком в C++ нельзя, поэтому мы будем считывать очередное число и класть его в элемент вектора с соответствующим номером.

Чтобы обратиться к отдельным элементам вектора, нужно указать номер в квадратных скобках после его имени. Внутри скобок допускается любое арифметическое выражение, а с отдельным элементом вектора можно обращаться как с обычной переменной. Если указать в скобках номер, который превышает размер вектора (или отрицательное число), то программа скомпилируется, но будет работать неправильно или вовсе сломается. Это связано с тем, что такое обращение попадает в область памяти, которая не относится к нашему вектору.

**Метод push\_back**

Рассмотрим еще одну похожую задачу: нужно считать последовательность и вывести задом наперед только положительные элементы. Задачу можно решить несколькими способами — например, считать всю последовательность и при выводе печатать только положительные элементы. Мы научимся решать задачу другим способом, запоминая только положительные элементы на этапе считывания данных.

int n;

cin >> n;

vector <int> a;

//считывание

for (int i = 0; i < n; i++) {

      int temp;

      cin >> temp;

      if (temp > 0) {

            a.push\_back(temp);

      }

}

//обработка и вывод

for (i = a.size() - 1; i >= 0; i--) {

      cout << a[i] << " ";

}

В этой программе при создании вектора мы не указывали количество элементов в нём — он создался пустым.

Добавление делается с помощью метода push\_back, при этом в скобках указано то, что и требуется добавить. Метод — это почти то же самое, что функция, применяемая к объекту. Сначала указывается имя объекта (в нашем случае это вектор a), затем ставится точка, пишется имя метода и в скобках указываются параметры. Аналогично push\_back используется и метод size, который не принимает параметров и возвращает количество элементов в векторе.

**Поиск минимума в последовательности**

Часто возникает задача найти минимум (или максимум) в последовательности из N чисел. Эта задача уже встречалась нам. Рассмотрим такую задачу: есть последовательность чисел, в ней нужно поменять местами то число, что стоит в начале последовательности, и самое маленькое число.

Большинство простых программ — это запись тех действий, которые произвёл бы человек для решения той же задачи. Что будет делать человек, если его попросят найти минимум в последовательности из одного числа? Очевидно, он назовёт это число. Для двух чисел задача тоже решается несложно — достаточно выбрать меньшее из них. Как эта задача решается больше чем для двух чисел? В каждый момент времени человек помнит наименьшее из уже названных чисел и сравнивает его со следующим в ряду. Если очередное число оказалось меньше, чем запомненное минимальное, то человек запоминает вместо старого числа новое. Таким образом, в любой момент человек может сказать, какое из уже названных чисел было минимальным. В нашей задаче нужно помнить не само число, а его номер в последовательности. После нахождения номера минимального числа, нужно просто поменять местами этот элемент с элементом с индексом 0.

int n;

cin >> n;

vector <int> a;

//считывание

for (int i = 0; i < n; i++) {

    int temp;

    cin >> temp;

    a.push\_back(temp);

}

//обработка

int num\_min = 0; //номер минимального элемента

for (int i = 0; i < n; i++) {

    if (a[i] < a[num\_min]) {

        num\_min = i;

    }

}

//обмен значений элементов a[0] и a[num\_min]

int temp;

temp = a[0];

a[0] = a[num\_min];

a[num\_min] = temp;

//вывод

for (auto now : a) {

    cout << now << " ";

}

В этом решении есть два интересных момента. Во-первых, это обмен значений двух переменных. Он делается через третью, вспомогательную переменную. Во-вторых, это ещё более короткая запись при выводе всех значений вектора. Цикл for (auto now : a) будет поочередно подставлять в переменную now все значения из вектора a. Здесь auto – это тип переменной, «автоматический». Поскольку вектор состоит из целых чисел, то переменная now автоматически будет определена как целое число.

**Сортировка массива**

Рассмотрим теперь задачу сортировки массива. В ней нужно расположить числа, записанные в массиве, в порядке неубывания (это то же самое, что порядок возрастания, только в нём могут быть одинаковые элементы). Для решения этой задачи воспользуемся решением предыдущей. Прошлая программа умела находить минимальный элемент и ставить его в начало массива. Таким образом, в начале массива появилась отсортированная часть, состоящая пока что из одного элемента. Если повторить эту операцию (поиск минимума и перестановка) для всех элементов, кроме начального, то у нас получится отсортированная часть массива, состоящая уже из двух элементов. Повторив эту операцию N раз, мы получим упорядоченный массив.

int n;

cin >> n;

vector <int> a;

//считывание

for (int i = 0; i < n; i++) {

    int temp;

    cin >> temp;

    a.push\_back(temp);

}

//обработка

for (int j = 0; j < n; j++) { //j – начиная с какого номера ищем наименьший

    int num\_min = j; //номер минимального элемента

    for (int i = j; i < n; i++) { //ищем только в еще не упорядоченной части

        if (a[i] < a[num\_min]) {

            num\_min = i;

        }

    }

    //обмен значений элементов a[j] и a[num\_min]

    int temp;

    temp = a[j];

    a[j] = a[num\_min];

    a[num\_min] = temp;

}

//вывод

for (auto now : a) {

    cout << now << " ";

}

Такой метод сортировки называется «сортировка выбором минимума».

Напоминаю вам что речь идёт о ситуациях в которых программа может вести себя непредсказуемо и выводить непредсказуемые результаты. Это связано с тем что в стандарте нет чёткого предписания как должен вести себя компилятор в подобной ситуации, в результате поведение программы может менятся от компилятора к компилятору или даже от запуска к запуску в одном компиляторе! Само понятие *неопрнеделённого поведения* свойственно именно языкам C/C++.

3. **Выход индекса за пределы массива/вектора.**

Поскольку в этом уроке речь идёт о массивах/векторах, то самое время затронуть вопрос выхода индекса массива\вектора за границы выделенной под него памяти, потому что вы неминуемо с этим столкнётесь ).

В языках C/C++ нет встроенного контроля над выходом индекса за дозволенные пределы - за этим обязан (вынужден) следить програмист. **Выход индекса за дозволенные границы может привести к падению программы**. Это не обязательно происходит при первом выходе за границы и может вовсе не случится, но при этом ваша программа может использовать (и даже изменять) данные в памяти не принадлежащей массиву/вектору. По понятным причинам, такая ситуация может привести к **непредсказуемому поведению программы**! Например, программа будет выдавать совсем не те результаты которые вы ожидаете.

Распространнённые ситуации, при которых происходит выход за границы массива.

1. Вы объявили массив (вектор) нулевой длины или недостаточной длины. Например,  исли объявить массив так

int arr[0];

то выход за пределы массива будет происходить по любому индексу, даже нулевому;

cout << arr[0];

В случае с вектором, часто путают два способа заполнения вектора. Если вы объявили вектор v*v* пустым

vector<int> v;

то к нему нельзя обращаться по индексу (!!!) до тех пор пока вы туда что либо не поместили (например методом push\_back)!!!  Если хотите заполнять массив динамически - объявляйте его пустым и заполняйте элементами по push\_back . Это уместно когда вам неизвестны размеры результирующего вектора (или потому что не известен размер исходных данных или вы хотите отфильтровать входные данные, как было показано на примерах в лекции). Если же вам известен размер вектора то лучше его объявить соотвествующего размера:

vector <int> v(n);

Тогда под элементы вектора сразу выделяется память и они доступны по индексу. Подробнее об этом написано в [этом](https://stepik.org/lesson/13025/step/1?discussion=3475217&reply=3477148&unit=4377) комментарии ниже.

2. Следует помнить, что заключительный элемент массива (вектора) из n*n* элементов имеет индекс n-1*n*−1. Нередко индекс n*n* включают в диапазон цикла for, а это уже выход за правую границу массива:

// правильный вариант, само значение n не входит в диапазон цикла

for (int i = 0; i < n; ++i) {

.....

}

3. В задачах часто встречается ситуация, когда вам необходимо сравнивать соседние элементы массива (обозначим его через arr*arr*). В этом случае, в теле цикла обычно сравнивают либо пару:

arr[i], arr[i+1]

либо пару

arr[i-1], arr[i]

В первом случае цикл for нужно закончить на индексе, на единицу меньше чем в обычном цикле

for (int i = 0; i < n - 1; ++i) {

.....

}

, во втором случае цикл for следует начать с единицы:

for (int i = 1; i < n; ++i) {

.....

}

В противном случае произойдёт выход за границы массива (на первом либо на последем шаге цикла соотвественно).

Если программа в результате всё таки *падает*, то в случае с массивом будет выдано характерное сообщение:

*Segmentation fault (core dumped)*

В случае с вектором - вместо такого сообщения иногда может быть выдана длинная таблица со странными значениями)).