Модуль 1 Перебор
Лекция 1.1 Перебор всех возможных строк из заданных символов

Первая часть нашего курса посвящена перебору. Мы разберём, как реализовывать рекурсивный перебор для решения разных задач. Начнём с совсем элементарной задачи.

> Нужно перебрать все строки длины 3, состоящие из строчных латинских букв ‘a’, ‘b’ и ‘c’. Каждая буква может встречаться в строке несколько раз или не встречаться совсем.

Можно убедиться, что всего таких строк — 27. Они все приведены на рис. 1.

```
aaa  baa  caa
aab  bab  cab
aac  bac  cac
aba  bba  cba
abb  bbb  cbb
abc  bbc  cbc
aca  bca  cca
acb  bcb  ccb
acc  bcc  ccc
```

Рис. 1. Все возможные строки из букв ‘a’, ‘b’ и ‘c’

Заметим, что удобно перебирать строки в определённом порядке. Сначала идут строки, начинающиеся с буквы ‘a’, затем — с буквы ‘b’, затем — с буквы ‘c’. Строки с одинаковой первой буквой упорядочены по второй букве, а если совпадают первые две буквы — по третьей букве. Такой порядок называется *лексикографическим*.

Дадим строгое определение. Рассмотрим две строки одинаковой длины s и t. Будем считать, что строка s меньше строки t, если первые k символов этих строк совпадают: $s[i] = t[i]$, $i = 0, 1, \ldots, k-1$ (у нас 0-индексация — символы в строках нумеруются с нуля), а следующий символ в строке s меньше, чем в строке t: $s[k] < t[k]$. В этом определении может быть $k = 0$, то есть совпадающих символов в начале может не быть.

Например, aba < baa. Эти строки различаются уже по первому символу, $k = 0$. Другой пример: baac < baca. Эти две строки имеют два общих символа в начале, но зато третий символ и первой строки меньше, чем у второй: a < c ($k = 2$).

Строки расположены в *лексикографическом порядке*, если каждая строка меньше следующей за ней в смысле данного определения. Если перебирать строки в лексикографическом порядке, значительно проще отследить, что мы ничего не пропустили и не перебрали одну и ту же строку два раза.

Рассмотрим программу на языке C++, реализующую перебор строк из трёх символов ‘a’, ‘b’, ‘c’ тремя вложенными циклами (см. рис. 2).

```
#include <iostream>
using namespace std;
...
for (char c1 = 'a'; c1 <= 'c'; c1++)
    for (char c2 = 'a'; c2 <= 'c'; c2++)
        for (char c3 = 'a'; c3 <= 'c'; c3++)
            cout << c1 << c2 << c3 << endl;
```

Рис. 2. Решение задачи о переборе строк из букв ‘a’, ‘b’ и ‘c’

В первом цикле переменная символьного типа c1 последовательно принимает значения от ‘a’ до ‘c’, то есть ‘a’, ‘b’, ‘c’.

for (**char** c1 = ‘a’; c1 <= ‘c’; c1++)

Это будет первый символ строки. Во втором цикле аналогично перебирается второй символ нашей строки c2.

for (**char** c2 = ‘a’; c2 <= ‘c’; c2++)

И в третьем цикле — третий символ c3.

for (**char** c3 = ‘a’; c3 <= ‘c’; c3++)

Затем выводятся все три символа и после них — перевод строки.

cout << c1 << c2 << c3 << endl;

Этот код позволяет вывести все нужные нам строки в лексикографическом порядке.

Рассмотрим более общую задачу.

> Вывести все строки длины n, состоящие из первых m букв латинского алфавита.

Предполагается, что числа n и m заданы. Эту задачу уже не так просто решить вложенными циклами, потому что значения n могут быть разными. Для большей общности будем искать последовательности не из букв, а из чисел от 1 до m.

> Вывести все последовательности длины n, состоящие из чисел от 1 до m. Каждое число может встречаться в последовательности несколько раз или не встречаться совсем.

Разберём программу, которая решает эту задачу (см. рис. 3).

Нам понадобится вектор a для хранения последовательности:

vector<**int**> a;

Вектор — это аналог массива с некоторыми дополнительными возможностями. Он расположен в пространстве имён std. Для использования векторов

```
vector<int> a;

void rec(int idx)
{
    if (idx == n)
    {
        out();
        return;
    }
    for (int i = 1; i <= m; i++)
    {
        a[idx] = i;
        rec(idx + 1);
    }
}

int main()
{
    ...
    a = vector<int>(n);
    rec(0);
    ...
}
```

Рис. 3. Решение задачи о переборе последовательностей

также нужно подключить заголовочный файл vector, то есть в начале программы требуются следующие две строчки:

#include<vector>

using namespace std;

Строка в функции main создаёт вектор целых чисел типа int заданного размера n (мы считаем, что уже прочитали это значение):

a = vector<**int**>(n);

Далее мы будем использовать вектор a как обычный массив. В нём будет содержаться последовательность, при этом по ходу выполнения программы последовательность будет меняться.

Основная часть решения — рекурсивная функция rec.

void rec(**int** idx)

Ей передаётся один целочисленный параметр idx — индекс в векторе a. Сначала функция rec вызывается от значения idx = 0.

rec(0);

В функции rec мы перебираем в цикле, какое число от 1 до m поставить в позицию idx в векторе.

for (**int** i = 1; i <= m; i++)

На каждой итерации цикла значение a[idx] присваивается i и происходит рекурсивный вызов функции rec.

a[idx] = i;

rec(idx + 1);

Мы переходим к следующему элементу вектора с индексом (idx + 1). Таким образом, функция rec для каждой позиции в векторе будет перебирать в цикле, какое число поставить в эту позицию, как будто у нас много вложенных циклов.

Рассмотрим условие выхода из рекурсии:

if (idx == n)

В векторе a 0-индексация, поэтому, когда idx = n, первые n элементов с индексами от 0 до $(n-1)$ уже заполнены. В этом случае мы вызываем функцию out(), которая выводит содержимое вектора a.

out();

Эта функция остаётся на самостоятельную реализацию. Затем выполняется выход из функции rec:

return;

Таким образом, мы разобрали код рекурсивной функции для решения задачи. На следующей лекции мы разберём работу этой функции на примере.