Цикл while. Оператор break

Циклы. Если бы их не было, то программы бы… мгновенно заканчивали свою работу. Например, операционная система, взаимодействует с нами, именно в цикле ожидая наших команд, без него компьютер мгновенно выключался бы сразу после загрузки!

Впрочем, для алгоритмических задач, это хорошо. Тестирующая система ждать не любит. Обычно в условии задачи указывается максимальное время, которая может работать ваша программа-решение. Часто это всего одна секунда. И если программа работает дольше, тестирующая программа прекращает ее работу и выдает вердикт "Превышено время работы" (на английском "Time Limit", сокращенно TL).

Но и за секунду компьютер может успеть сделать немало. Производительность процессора 3 гигагерца означает, что такой процессор может исполнить три миллиарда операций за секунду. Но нужно учитывать, что это касается именно простейших операций, и даже простое сложение это не одна операция для процессора. Не следует забывать и о том, что ваша программа выполняется в многозадачной системе. Эту единственную секунду с вашей программой "делит" операционная система, да и сама тестирующая система.

Можно считать, что за секунду ваша программа может исполнить 200 миллионов операций. 500 может не успеть, а миллиард не успеет точно.

Это именно операции вычислений. Скажем, в цикле постоянно выполняются сложения или умножения. Операции ввода-вывода занимают значительно больше времени и Java-олимпиадники обязаны задумываться над ускорением процесса ввода-вывода. Возможные варианты оптимизации чтения данных мы обсудим, когда будем решать задачи на обработку больших объемов информации, что бывает достаточно часто в задачах.

Практически весь курс Алгоритмов это попытка сделать "обычные вещи" быстрее. Например, чтобы определить, простое число нам дано или нет, нужно проверять делится ли оно на что-нибудь меньшее его, кроме единицы. Возьмем, скажем, 25. На два – не делится, на три – нет, на четыре – нет, на пять… да, значит 25 – число составное. А число 101 не разделится ни на 2, ни на 3, ни на 27, ни на 100. Значит – простое. Сколько делать проверок для числа N? N-1 штук, то есть проверять все предыдущие числа до самого числа? Но тогда 1000000009 наша программа проверить не успеет, будет TL… Но зачем, делить на 4, если оно уже не разделилось на 2? Проверять можно только нечетные. Но и 500 миллионов делений-проверок программа скорее всего сделать не успеет. Можно серьезно сократить количество действий.

Цикл в программе организовать просто.

По структуре он очень похож на `if`.

```
//ПСЕВДОКОД
while (<условие>)
{
      <Команды> (выполняются пока условие истинно)
```

```
}
```

while . – в переводе с английского значит "до тех пор", "пока". Программа будет выполнять команды до тех пор, пока условие истинно. Проверяется условие, если оно истинно – выполняются команды и - опять на проверку условия.

Блок команд, расположенных после `while`, называют *телом цикла*.

Обратите внимание, как и в if точка с запятой после условия в скобках НЕ ставится!

```
while (<условие>);
```

Таким образом, команды могут не выполниться ни разу (если условие сразу ложно) или выполняться сколь угодно много раз. Естественно, предусматривают остановку этого процесса. Обычно условие зависит от значения переменной, а эта переменная изменяется командами.

Например, цикл в реальной программе может выглядеть так:

```
int x = 0;
while (x < 5)
{
	printf("%d", x);
	x++;
}
```

Сколько эта программа напечатает чисел?

На самом деле правильный ответ – одно, ведь значения печатаются без пробелов.

Что напечатает строка

```
printf("x = %d", x);
```

Если добавить ее после закрывающей фигурной скобки? Переменная x будет равна 5. Именно в этот момент условие станет ложным и цикл закончится.

Условия, как и в **`if`**, могут быть абсолютно любыми логическими выражениями, например, содержать логические операции **`&&`**, **`||`**.

Оператор break

Например, программа

```
int x = 10;
while (x < 100)
{
	if (x == 2 * (x / 10) * (x % 10)) break;
	x++;
}
```

выйдет из цикла, когда найдет первое двузначное число, равное удвоенному произведению своих цифр.

Не стоит злоупотреблять досрочным выходом из цикла.

Можно, конечно написать так:

```
int i = 1;
while(true)
{
      printf (i);
      i++;
      if (i == 10) break;
}
```

Но значительно лучше читается простое и естественное

```
while(i <= 10)
```

без использования **`break`**.

Следует учитывать, что **`break`** **прерывает именно цикл**. Но использовать его, конечно, стоит только в условном операторе.

Вот, собственно, и вся "теория". Наверняка вы все это уже слышали. Но вот овладеть *применением* циклов для разработки алгоритмов непросто. Тут важна практика и знание некоторых "секретов".

Например, обычно не стоит использовать условие с точным сравнением. Лучше использовать неравенства.

Например, выведем числа от 1 до 99 через пробел:

```
int x = 1;
while (x != 100)
{
      out.print(x + " ");
      x++;
}
```

Все работает правильно. Когда x станет равен 100 условие станет ложным и мы выйдем из цикла.

Но переформулируем задачу: вывести все **нечетные** числа от 1 до 99.

Мгновенное исправление

```
int x = 1;
while (x != 100)
{
      printf("%d ", x);
      x += 2;
}
```

приводит к ошибке! Цикл будет выполняться практически бесконечно.

А если бы условие изначально было **`x < 100`**, исправление работало бы корректно.

Чтобы **сделать какие-то команды ровно N раз**:

```
int i = 0;
while (i < N)
{
      //… команды выполнятся ровно N раз
      i++;
}
```

Нужно начать с `0` и поставить знак `<`.

Переменную **`i`** , которая с нуля увеличивается до некоторого значения по единице обычно называют счетчиком.

Часто в задаче дается число N, а потом следует N чисел.

В этом случае необходимо поместить команду считывания в цикл.

Для демонстрации всех этих приемов решим задачу с Московской олимпиады.

Оргкомитет Московской городской олимпиады решил организовать обзорную экскурсию по Москве для участников олимпиады. Для этого был заказан двухэтажный автобус (участников олимпиады достаточно много и в обычный они не умещаются) высотой 437 сантиметров. На экскурсионном маршруте встречаются N мостов. Жюри и оргкомитет олимпиады очень обеспокоены тем, что высокий двухэтажный автобус может не проехать под одним из них. Им удалось выяснить точную высоту каждого из мостов. Автобус может проехать под мостом тогда и только тогда, когда высота моста превосходит высоту автобуса. Помогите организаторам узнать, закончится ли экскурсия благополучно, а если нет, то установить, где произойдет авария.

Формат входных данных

Во входном файле сначала содержится число *N* ($1 \le N \le 1000$). Далее идут *N* натуральных чисел, не превосходящих 10000 - высоты мостов в сантиметрах в том порядке, в котором они встречаются на пути автобуса.

Формат выходных данных

В единственную строку выходного файла нужно вывести фразу "`No crash`", если экскурсия закончится благополучно. Если же произойдет авария, то нужно вывести сообщение "`Crash` *k*", где *k* - номер моста, где произойдет авария. Фразы выводить без кавычек ровно с одним пробелом внутри.

Примеры

Входные данные	Выходные данные
1 763	No crash

3 763 245 113	Crash 2
1 437	Crash 1

Решение

В цикле N раз считываем числа N раз и проверяем превосходит ли считанное число 437.

Одновременно увеличиваем счетчик.

Если нет - выводим ответ и завершаем цикл при помощи **`break`** иначе идём дальше.

```
while(i < N)
{
      t = _______________;
      if (t <= 437)
      {
            printf(___________);
            break;
      }
      __________;
}
```

После выхода из цикла нужно опять проверить, почему мы из него вышли (потому что доехали до конца или разбились в дороге):

```
if (t > 437)
{
      printf("No Crash");
}
```

Заполните пропуски в коде самостоятельно.

Эта задача имеет на informatics №1023.