

№5 Односвязный список

Время	1 сек.
Память	32 МБ

Реализуйте **односвязный список**, который изначально **уже содержит один элемент** со значением `100`. Текущий элемент по умолчанию указывает на этот первый узел.

Необходимо обработать последовательность следующих команд:

- **append X** — добавляет новый узел со значением `X` **после текущего** и делает его *новым текущим*. Вывод для данной команды: `append X - OK`
- **go N** — смещаемся на `N` элементов вперёд по списку. Если при движении сталкиваемся с концом списка раньше, чем будет выполнено `N` шагов, то оставляем *текущим* элементом последний элемент списка. Вывод для данной команды: `go N - OK`
- **print** — вывести значение текущего элемента списка. Вывод для данной команды: `print X - OK`
- **reset** — делаем *текущим* элементом первый элемент списка (значение `100`). Вывод для данной команды: `reset - OK`

① Caution

Для списка используйте собственную структуру `List`. Никаких массивов или `vector`'ов. Из стандартной библиотеки ТОЛЬКО `iostream` и `string`.

```
struct List {  
    // определите самостоятельно  
};
```

Входные данные:

Ввод содержит **последовательность команд**, по **одной** в каждой строке. Команды выполняются **в порядке следования**.

Выходные данные:

Вывод должен содержать результат обработки всех команд. Каждая строка вывода соответствует **одной выполненной команде** входного файла, **в том же порядке**.

Гарантии и Ограничения

- Последняя команда во входе ВСЕГДА `print`.
- Аргументы команд (`X` в `append X` и `N` в `go N`) находятся в диапазоне от $0 \leq X, N \leq 10^9$ включительно.

- Вход содержит не более 1 000 команд `append`.
- Общее число команд (строк) не превышает 10 000.

Пример:

stdin	stdout
<code>append 10</code>	<code>append 10 - OK</code>
<code>append 20</code>	<code>append 20 - OK</code>
<code>reset</code>	<code>reset - OK</code>
<code>go 2</code>	<code>go 2 - OK</code>
<code>print</code>	<code>print 20 - OK</code>
<code>append 30</code>	<code>append 30 - OK</code>
<code>go 5</code>	<code>go 5 - OK</code>
<code>print</code>	<code>print 30 - OK</code>
<code>reset</code>	<code>reset - OK</code>
<code>print</code>	<code>print 100 - OK</code>

Пояснение к примеру:

Исходные данные

В начале список **уже содержит один элемент** со значением `100`. Он же является *текущим* элементом.

```
[100]▶ nullptr
↑
current
```

1. Команда `append 10`

Добавляем элемент со значением `10` после *текущего* (`100`).

```
[100] → [10]▶ nullptr
↑
current
```

Вывод:

```
append 10 - OK
```

2. Команда `append 20`

Добавляем узел `20` после *текущего* (`10`).

```
[100] → [10] → [20]▶ nullptr
↑
current
```

Вывод:

```
append 20 - OK
```

3. Команда `reset`

Указатель возвращается к **началу списка**— на первый элемент (`100`).

```
[100] → [10] → [20]▶ nullptr  
↑  
current
```

Вывод:

```
reset - OK
```

4. Команда `go 2`

Двигаемся на **два шага вперёд**:

- первый шаг от элемента `100` к элементу `10`
- второй шаг от элемента `10` к элементу `20`

В результате *текущим* элементом становится `20`.

```
[100] → [10] → [20]▶ nullptr  
↑  
current
```

Вывод:

```
go 2 - OK
```

5. Команда `print`

Печатаем значение *текущего* узла — `20`.

Вывод:

```
print 20 - OK
```

6. Команда `append 30`

Добавляем элемент `30` после *текущего* (`20`).

```
[100] → [10] → [20] → [30]▶ nullptr  
↑  
current
```

Вывод:

```
append 30 - OK
```

7. Команда `go 5`

Пробуем сделать 5 шагов вперёд, однако *текущий* элемент `30` — последний элемент списка. По условию, если достигаем конца раньше, чем совершим необходимое число шагов — то останавливаемся на последнем элементе. Таким образом элемент `30` остаётся *текущим*.

```
[100] → [10] → [20] → [30]▶ nullptr  
↑  
current
```

Вывод:

```
go 5 - OK
```

8. Команда `print`

Текущий элемент всё тот же (`30`).

Вывод:

```
print 30 - OK
```

9. Команда `reset`

Возвращаемся к начальному элементу (`100`).

```
[100] → [10] → [20] → [30]▶ nullptr  
↑  
current
```

Вывод:

```
reset - OK
```

10. Команда `print`

Теперь текущий элемент — `100`.

Вывод:

```
print 100 - OK
```

Итог работы всей программы

```
append 10 - OK  
append 20 - OK  
reset - OK  
go 2 - OK  
print 20 - OK  
append 30 - OK  
go 5 - OK  
print 30 - OK  
reset - OK  
print 100 - OK
```

