

Корректное чтение данных из стандартного потока ввода

Постановка задачи

Предположим, что программа в процессе своей работы ожидает от пользователя ввода целого числа (значения типа `int`). Прочитать это число из стандартного потока ввода можно с помощью функции `scanf`, например, с помощью следующего фрагмента кода.

```
int x;
...
if (scanf("%d", &x) != 1) {
    // Treat error.
}
```

В этом фрагменте успешно считанное число заносится в переменную `x`. В случае ошибки чтения производится ее обработка.

Далее, предположим, что пользователь набрал на клавиатуре последовательность символов `1005o0`, завершив набор нажатием на клавишу `Enter`. В этот момент в стандартный поток ввода программы была добавлена следующая последовательность символов.

```
'1' '0' '0' '5' 'o' '0' '\n'
```

Скорее всего, пользователь хотел ввести число `100500`, но ошибся. Вместо цифры `0` ввел букву `o`. С помощью функции `scanf` программа успешно считала число `1005` и продолжит свою работу. Заметим, что в стандартном потоке ввода при этом останется последовательность символов.

```
'o' '0' '\n'
```

В дальнейшем возможны следующие варианты. Программа штатно отработает и вычислит результат, который не будет совпадать с требуемым. В наихудшем случае пользователь этого даже не заметит. Ошибка также может вскрыться при чтении следующего входного параметра. В этом случае может произойти неправильная интерпретация этой ошибки.

В любом случае описанные сценарии являются неприемлемыми. Подобные ошибки должны вскрываться в момент возникновения, а пользователю должна предоставляться возможность повторного корректного ввода данных. С этой целью должна анализироваться вся введенная (добавленная в стандартный поток ввода) последовательность символов. Для этого нам потребуется научиться работать со стандартной функцией `fgetc`.

Стандартная функция `fgetc`

Прототип стандартной функции `fgetc` определен в заголовочном файле `stdio.h` и имеет следующий вид.

```
int fgetc(FILE *fi);
```

Эта функция считывает очередной символ из потока ввода `fi` и возвращает код этого символа или константу `EOF` в случае, если был достигнут конец файла или возникла ошибка чтения. Код символа представляет собой `unsigned char`, приведенный к типу `int`.

Очистка стандартного потока ввода

Реализуем вспомогательную функцию `skip_chars`, которая осуществляет «очистку» стандартного потока ввода. Очистка состоит в чтении всех символов, пока не будет прочитан символ переход на новую строку или не будет достигнут конец файла. Эта функция также осуществляет проверку, являются ли прочитанные символы пробельными (`' '` и `'\t'`). Если были прочитаны только пробельные символы, функция возвращает число `0`, иначе `-1`.

```
1 int skip_chars(void) {
2     int r, c;
3
4     for (r = 0;;) {
5         c = fgetc(stdin);
6
7         if (c == '\n' || c == EOF)
8             break;
9
10        if (c != ' ' && c != '\t')
11            r = -1;
12    }
13
14    return r;
15 }
```

Чтение значений типа `int`

Прежде всего нужно договориться о том, какие последовательности символов, введенные пользователем, являются корректными с точки зрения чтения значения типа `int`. Будем считать, что корректной является последовательность символов, состоящая из трех частей. Первая часть – это последовательность пробельных символов. Эта часть может отсутствовать. Вторая часть – это собственно символы, представляющие (кодирующие) вводимое целое число. Третья часть – это последовательность пробельных символов, которая завершается символом перехода на новую строку.

Ниже приведены примеры корректных последовательностей. Все они используются для ввода числа `123`.

```
'1' '2' '3' '\n'
' ' ' ' '1' '2' '3' '\n'
'1' '2' '3' ' ' '\n'
' ' '1' '2' '3' ' ' ' ' '\n'
```

Разберем функцию `readInt`, которая анализирует корректность вводимых пользователем последовательностей символов и преобразует корректную последовательность символов в целое число типа `int`.

Эта функция имеет два выходных параметра. Первый параметр передается через оператор `return`, имеет тип `int` и может принимать два значения 0 или -1. Значение 0 говорит о том, что введенное пользователем целое число было успешно прочитано. Значение -1 говорит о том, что стандартный поток ввода был закрыт до того момента, как у пользователя получилось ввести корректное значение. Второй параметр имеет тип `int`, представляет собой корректно введенное пользователем целое число и передается через указатель `p`.

```

1  int readInt(int *p) {
2      int f;
3
4      for(f = 0; !feof(stdin); f = 1) {
5          if(f)
6              printf("Please, try again ...\\n");
7
8              printf("Input: ");
9
10             if(scanf("%d", p) != 1) {
11                 skip_chars();
12                 continue;
13             }
14
15             if(skip_chars() == -1)
16                 continue;
17
18             return 0;
19         }
20
21         return -1;
22     }

```

Основу функции `readInt` составляет цикл (строки 4 – 19), в рамках итераций которого пользователю предоставляется возможность корректно ввести данные. В строке 10 с помощью функции `scanf` обрабатываются первая и вторая части введенной последовательности символов. В строке 15 с помощью функции `skip_chars` обрабатывается третья часть введенной последовательности символов.

В некоторых случаях могут вводиться дополнительные ограничения на допустимые значения для вводимого числа. Например, вводимое число должно быть больше 1 и меньше 9. В этом случае

функция `readInt` должна быть модифицирована. Изменения разумно добавлять в строку 10, например.

```

...
if (scanf("%d", p) != 1 || !(1 < *p && *p < 9)) {
...

```

Чтение значений типа `double`

Для чтения вещественных чисел (значений типа `double`) необходимо внести следующие модификации в функцию `readInt`. Нужно в строке 1 заменить тип указателя `p` на `double *`, а в строке 10 спецификатор `%d` заменить на `%lf`.

Чтение символьных значений

Предположим, что пользователь должен ввести значение символьного типа. Например, один из символов `y`, `Y`, `n`, `N`. Программа может прочитать это значение с помощью следующей функции `readYorN`.

```

1  int readYorN(int *p) {
2      int f;
3
4      for(f = 0; !feof(stdin); f = 1) {
5          if(f)
6              printf("Please, try again ...\\n");
7
8              printf("Input: ");
9
10             *p = fgetc(stdin);
11             if(*p != 'y' && *p != 'Y' &&
12                *p != 'n' && *p != 'N' ) {
13                 skip_chars();
14                 continue;
15             }
16
17             if(skip_chars() == -1)
18                 continue;
19
20             return 0;
21         }
22
23         return -1;
24     }

```