Посимвольный ввод и вывод. Понятие буфера

Функции putchar() и getchar()

В заголовочном файле stdio.h содержится объявление не только функции `printf()`, но и многих других, связанных с вводом-выводом. Среди них есть функции, которые обрабатывают по одному символу за вызов — `putchar()` и `getchar()`.

Функция `putchar()` обычно принимает в качестве аргумента символ, либо символьную переменную и в результате своей работы выводит соответствующий символ на экран. Однако этой функции можно передать любое целое число, но, понятное дело, символа на экране вы можете не получить, если числу не соответствует ни один символ по таблице ASCII. Например:

```
char ch = 'c';

putchar('a');
putchar(98);
putchar('\n');
putchar(ch);
```

Результат:

```
ab
c
```

Функции `putchar()` и `printf()` в определенном смысле взаимозаменяемы, так как, используя ту или другую, можно получить один и тот же результат. Хотя программный код будет выглядеть по-разному:

```
char str[] = "Hello";
int i;

// первое Hello
printf("%s\n", str);

// второе Hello
for (i = 0; str[i] != '\0'; i++)
    putchar(str[i]);

printf("\n");
```

В результате выполнения этого кода на экране будут напечатаны два слова "Hello", разделенные переходом на новую строку. С `putchar()` это выглядит несколько сложнее. Как мы знаем, любая строка оканчивается нулевым по таблице ASCII символом, в данном случае этот символ служит сигналом для прекращения вывода на экран. Но если бы понадобилось вывести на экран строку, разделяя ее символы каким-нибудь другим символом (например, тире), то и в случае с `printf()` было бы не так все просто:

```
char str[] = "Hello";
int i;

for (i = 0; str[i] != '\0'; i++)
    printf("%c-",str[i]);
printf("%c%c %c",'\b', '\0', '\n');

for (i = 0; str[i] != '\0'; i++) {
    putchar(str[i]);
    putchar('-');
}
printf("%c%c %c",'\b', '\0', '\n');
```

Результат:

```
H-e-l-l-o
H-e-l-l-o
```

Поэтому выбор в пользу той или иной функции зависит от ситуации и ваших предпочтений.

В отличие от функции `putchar()` функция `getchar()` не имеет параметром. Когда `getchar()` выполняется, она считывает их потока ввода один символ и возвращает его в программу. Полученный таким образом символ может быть присвоен переменной, участвовать в выражениях или выводиться на экран с помощью функций вывода.

```
int a;

a = getchar();
printf("%c ", a);
putchar(a);

putchar('\n');
```

Если при выполнении этого кода ввести символ, то после нажатия Enter вы увидите два таких же символа на экране:

```
u
u u
```

Первый — результат выполнения функции `printf()`, второй — `putchar()`. Если вы перед нажатием Enter введете несколько символов, то прочитан будет только первый, остальные будут проигнорированы. Посмотрите вот на этот код:

```
char a, b, c;

a = getchar();
putchar(a);
b = getchar();
putchar(b);
c = getchar();
putchar(c);

printf("\n");
```

Как вы думает, как он будет выполняться? По идее после ввода символа, он должен сразу отображаться на экране функцией `putchar()` и запрашиваться следующий символ, потому что далее идет снова вызов `getchar()`. Если вы как корректный пользователь программы сначала введете первый символ и нажмете Enter, то символ отобразиться на экране. Потом вы введете второй символ и после Enter он тоже отобразиться. И тут программа завершится, не дав ввести вам третий символ.

Прежде чем попытаться найти объяснение, изобразим "некорректного пользователя" и перед первым нажатием Enter введем несколько символов (больше двух). После Enter вы увидите три первых символа введенной вами строки, и программа завершиться. Хотя куда логичней было бы ожидать, что будет прочитан только первый символ, потом выведен на экран и потом запрошен следующий символ.

Такое странное на первый взгляд поведение программы связано не с языком С, а с особенностью работы операционных систем, в которых реализован буферный ввод-вывод. При операциях ввода-вывода выделяется область временной памяти (буфер), куда и помещаются поступающие символы. Как только поступает специальный сигнал (например, переход на новую строку при нажатии Enter), данные из буфера передаются по месту своего назначения (на экран, в переменную и др.).

Теперь, зная это, давайте посмотрим, что происходило в нашей программе, и сначала разберем второй случай с "некорректным пользователем", т.к. для понимания этот случай проще. Когда пользователь ввел первый символ, он попал в переменную a, далее сработала функция `putchar(a)` и символ попал в буфер. Т.к. Enter'a не было, то содержимое буфера на экране не было отображено.

Пользователь ввел второй символ, переменная b получила свое значение, а `putchar(b)` отправила это значение в буфер. Аналогично с третьим символом. Как только пользователь нажал Enter, содержимое буфера было выведено на экран. Но символы, которые были выведены на экран, были выведены не программой, а операционной системой. Программа же выводила символы еще до того, как мы нажали Enter.

Почему же в первом случае при выполнении программы мы смогли ввести и увидеть на экране только два символа? Когда был введен первый символ, то он был присвоен переменной a и далее выведен в буфер. Затем был нажат Enter. Это сигнал для выброса данных их буфера, но это еще и символ перехода на новую строку. Этот символ '\n', наверное, и был благополучно записан в переменную b. Тогда в буфере должен оказаться переход на новую строку, после этого введенный символ (уже помещенный в переменную c). После нажатия Enter мы должны были бы увидеть переход на новую строку от символа '\n' и букву. Однако печатается только буква. Причина этого вероятно в том, что переход на новую строку не хранится в буфере.

Во многих учебниках по языку C приводится пример считывания символов, вводимых пользователем, и их вывод на экран:

```
int a;

a = getchar();
while (a != '\n') {
    putchar(a);
    a = getchar();
}
putchar('\n');
```

В переменной a всегда хранится последний введенный символ, но перед тем как присвоить a новое значение с помощью функции `putchar()` старое значение сбрасывается в буфер. Как только поступает символ новой строки, работа программы прекращается, а также, поскольку была нажата клавиша Enter, происходит вывод содержимого буфер на экран. Если в условии цикла `while` будет не символ '\n', а какой-нибудь другой, то программа будет продолжать обрабатывать символы, даже после нажатия Enter. В результате чего мы можем вводить и выводить множество строк текста.

Напишите программу посимвольного ввода-вывода, используя в качестве признака окончания ввода любой символ, кроме '\n'. Протестируйте ее.

При совместном использовании функций `putchar()` и `getchar()` обычно пользуются более коротким способом записи. Например:

```
while ((a = getchar()) != '~')
    putchar(a);
```

---

1. Объясните, почему сокращенный вариант записи посимвольного ввода-вывода работает правильно. Для этого опишите последовательность операций в условии цикла `while`.
2. Перепишите вашу программу на более короткий вариант.

EOF

Как быть, если требуется "прочитать" с клавиатуры или файла неизвестный текст, в котором может быть абсолютно любой символ? Как сообщить программе, что ввод окончен, и при этом не использовать ни один из возможных символов?

В операционных системах и языках программирования вводят специальное значение, которое служит признаком окончания потока ввода или признаком конца файла. Называется это значение EOF (end of file), а его конкретное значение может быть разным, но чаще всего это число -1. EOF представляет собой константу, в программном коде обычно используется именно имя (идентификатор) константы, а не число -1. EOF определена в файле stdio.h.

В операционных системах GNU/Linux можно передать функции `getchar()` значение EOF, если нажать комбинацию клавиш Ctrl + D, в Windows – Ctrl + Z.

Исправьте вашу программу таким образом, чтобы считывание потока символов прерывалось признаком EOF.

Решение задач

Не смотря на свою кажущуюся примитивность, функции `getchar()` и `putchar()` часто используются, т.к. посимвольный анализ данных при вводе-выводе не такая уж редкая задача. Используя только функцию `getchar()`, можно получить массив символов (строку) и при этом отсеять ненужные символы. Вот пример помещения в строку только цифр из потока ввода, в котором может быть набран абсолютно любой символ:

```c
#include <stdio.h>

#define N 100

int main () {
    char ch;
    char nums[N];
    int i;

    i = 0;
    while ((ch = getchar()) != EOF && i < N-1)
        if (ch >= 48 && ch <= 57) {
            nums[i] = ch;
            i++;
        }

    nums[i] = '\0';

    printf("%s\n", nums);
}
```

Здесь ввод символов может прекратиться не только при поступлении значения EOF, но и в случае, если массив заполнен (`i < N-1`). В цикле `while` проверяется условие, что числовой код очередного символа принадлежит диапазону [48, 57]. Именно в этом диапазоне кодируются цифры (0-9). Если поступивший символ является символом-цифрой, то он помещается в массив по индексу i, далее i увеличивается на 1, указывая на следующий элемент массива. После завершения цикла к массиву символов добавляется нулевой символ, т.к. по условию задачи должна быть получена строка (именно для этого символа ранее резервируется одна ячейка массива – `N-1`).