Обработка символьных данных и текстов

В языке С существует четыре базовых типа данных, которые мы используем для описания переменных:

- символьный (char);

- целый (int);

- действительный (float);

- двойной точности(double)

Символьный тип в программах на С представлен как целочисленный (любому символу соответствует в машинном представлении какое-либо число от 0 до 255).

Символьная константа – это целое число, записанное в виде символа, заключенного в одиночные кавычки. Значением символьной константы является код символа, зависящий от используемой системы кодировки. Символьные константы могут участвовать в операциях по тем же правилам, что и целые переменные или константы.

Некоторые трудно представимые символы записываются как комбинация нескольких других символов. Такие комбинации называются Esc-последовательностями. К ним, например, относятся:
\n – переход на новую строку **\a** – звонок

\t – горизонтальная табуляция **\f** – перевод страницы

\\ - собственно наклонная черта.

Строковая константа – это произвольное количество символов, заключенных в двойные кавычки, которые не являются составной частью строки. Такая строка фактически воспринимается как массив символов, а в конце этого массива присутствует специальный невидимый символ завершения строки **\0**.

Символьная константа и строка, содержащая один символ, воспринимаются компьютером совершенно по разному.

Так, “a” воспринимается как массив из двух символов, а ‘a’ – как число, равное коду буквы a.

Обработка текстов может происходить двумя способами:

- без использования подключения файлов в программе

Первый способ применим только для однопроходных задач. Это связано с тем, что при обработке текстов не используют массивы. (Объем текста может быть слишком большим.)

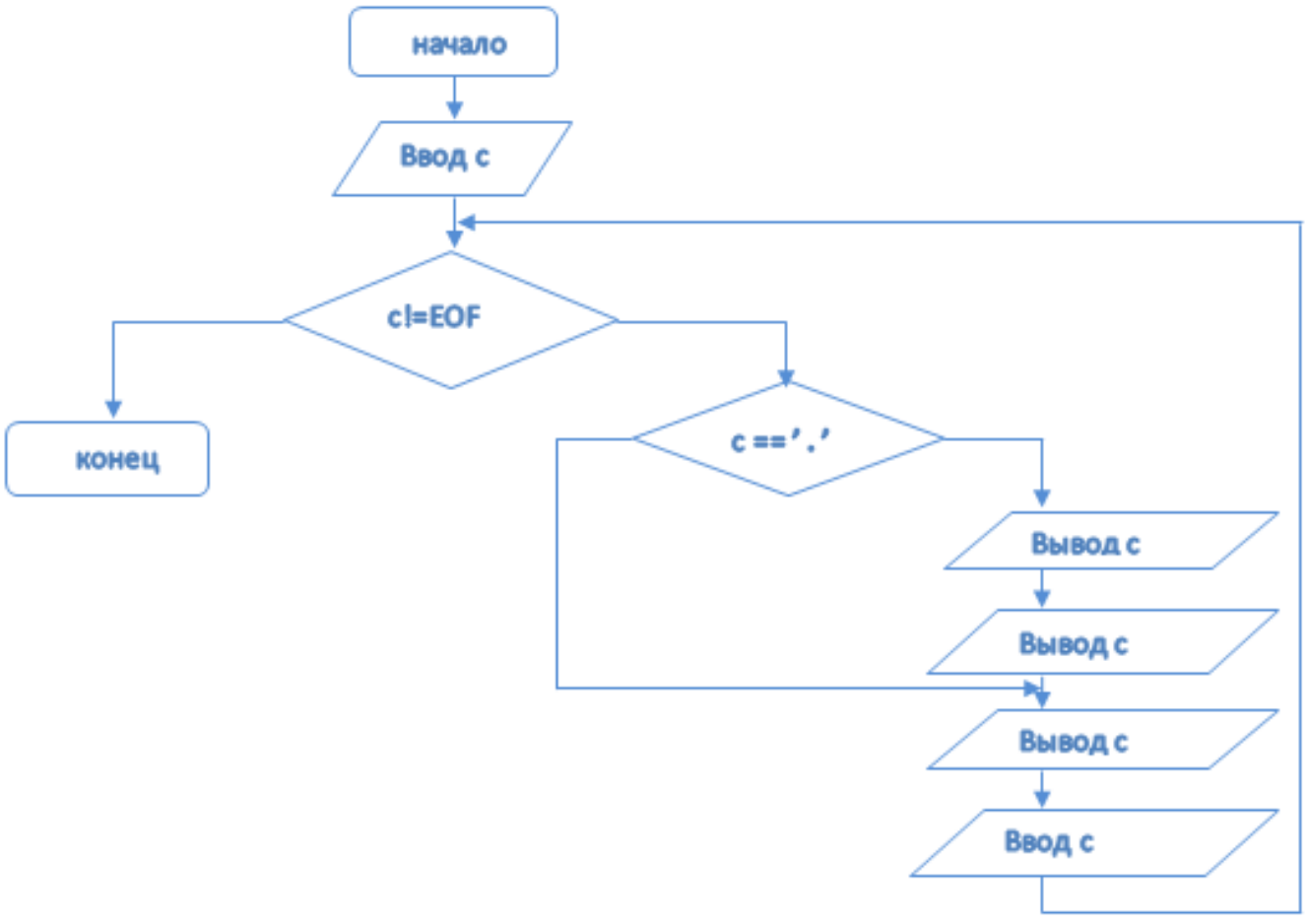
- с файлами

Допустим, есть задача обработать текст. Если текст большого размера, то чтобы каждый раз при запуске программы его не вводить, существует возможность отправить исходный текст в файл и подключать по мере необходимости. Может быть обратная ситуация - когда результатами программы нужно пользоваться несколько раз. В этом случае также удобно их хранить в файле.

Второй способ позволяет перемещаться по файлу в любом направлении, открывать и закрывать его. Поэтому этот способ подходит для многопроходных задач.

1. Решение задачи:

Задача: Дан текст. Заменить все точки в нем на многоточия.



2. Пишем программу:

```
#include<stdio.h>
#include<locale.h>
#pragma warning (disable : 4996)
void main(){
    setlocale(LC_ALL, "RUS");
    int c;
    c = getchar();
    while (c != EOF) {
        if (c == '.') {
            putchar(c);
            putchar(c);
        }
        putchar(c);
        c = getchar();
    }
}
```

c – описана как int, в отладчике просматривается код символа; может быть описана как **char,** тогда в отладчике можно будет увидеть сам символ

функция **getchar ( )** считывает очередной символ со стандартного ввода (с клавиатуры), возвращает код этого символа и помещает в переменную (в данном случае c).

EOF (end of file) –специальный символ, который указывает на конец файла. При вводе с клавиатуры сочетание клавиш Ctrl-Z (или f6) моделирует EOF.

scanf не распознает EOF, поэтому используем getchar

функция **putchar (c)** выводит на экран символ, код которого находится в переменной c.

3. Решение задачи:

Задача: Напечатать таблицу кодов символов в файл.

```c
#include<stdio.h>
#include<locale.h>
#pragma warning (disable : 4996)
void  main() {
    setlocale(LC_ALL, "RUS");
    FILE* bb; //описываем указатель на файловый поток
    int  n;
    bb = fopen("kuda.txt", "w");
    //указатель FILE теперь ссылается на kuda.txt
    for (n = 0; n <= 255; n++)
        fprintf(bb, "%c - %d\n", n, n);
    fclose(bb);
}
```

FILE * bb описывается переменная, которая является указателем на файловую структуру в памяти. (**bb** – канал связи)

Синтаксис вызова функции открытия файла

канал = fopen ("имя_файла", "режим_доступа"); Имя файла и режим доступа в ковычках, так как они передаются как строки.

Режимы доступа:

r – чтение **w** – запись **a** – добавление

Если файл и программа находятся в разных папках, необходимо указать полный путь к файлу.

fprintf печатает в файл (работает аналогично **printf**, для считывания из файла используем **fscanf**).

fprintf (канал, список форматов, список переменных);

%c – печать символа **%d** – печать целого числа

fclose закрывает канал связи.

4. Решение задачи:

Задача: Дан файл с текстом. Заменить все точки в нем на многоточия. Результат записать в другой файл.

```
#include<stdio.h>
#include<locale.h>
#pragma warning (disable : 4996)
void main() {
    setlocale(LC_ALL, "RUS");
    FILE* aa, * bb;
    char otkuda[20], kuda[20], c;
    printf("файл - источник: \n");
    scanf("%s", otkuda);
    printf("файл - потребитель: \n");
    scanf("%s", kuda);
    aa = fopen(otkuda, "r");
    bb = fopen(kuda, "w");
    while ((c = fgetc(aa)) != EOF) {
        if (c == 46) {
            fputc(c, bb);
            fputc(c, bb);
        }
        fputc(c, bb);
    }
    fclose(bb);
    fclose(aa);
}
```

Нам понадобятся 2 канала, **aa** – для чтения, **bb** – для записи.

Названия файлов вводятся с клавиатуры, для их записи понадобятся символьные массивы **otkuda[20]** и **kuda[20]**

%s указывает на строку (символьный массив), в **scanf** НЕ НУЖЕН **&**,так как **otkuda** и **kuda** являются указателями на массивы

Функция **fgetc(aa)** считывает очередной символ из файла, возвращает код этого символа и помещает в переменную (в данном случае c). Работает аналогично **getchar( )** . В отличие от **getchar**, который работает со стандартным вводом, **fgetc** получает на вход указатель на файловый поток.

46 – код символа точка (см. таблицу кодов)

Функция **fputc (c, bb)** печатает в файл символ, код которого находится в переменной c. Работает аналогично **putchar( )**. В отличие от **putchar**, **fputc** получает на входе код символа и указатель на файловый поток.

aa == NULL истинно, если путь к файлу указан не верно или файла не существует

NULL – нулевой указатель

Лабораторная работа №4.

Для выполнения лабораторной работы необходимо:

1. Получить индивидуальное задание.

2. Написать программу на языке программирования Си.

3. **ОТЛАДИТЬ И ПРОТЕСТИРОВАТЬ ПРОГРАММУ**.

4. Составить отчет о лабораторной работе.

Содержание отчета:

- Титульный лист.

- Условие задачи.

- Текст программы.

- Скриншоты, подтверждающие корректную работу программы с разными наборами входных данных.