НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
Кафедра информатики и процессов управления (№17)

Дисциплина «Информатика» (основной уровень), 1-й курс, 2-й семестр.

Методические указания

Тематическое занятие 12

Символы и строки. Работа с файлами

Содержание

Символы

Символьный тип данных и кодировки

Тип данных `char` применяется для хранения символьных данных, но фактически является целочисленным, поскольку хранит целые числа – числовые коды символов.

В США наиболее часто используется код **ASCII** (American Standard Code for Information Interchange), который использует числовые значения в диапазоне от 0 до 127. Для хранения ASCII-символа достаточно 7-и разрядов.

В кодировке ASCII, например, значение `65` представляет букву `A` (латинского алфавита) в верхнем регистре.

Многие наборы символов содержат более 128 и даже более 256 значений. Например, в кодировке **Unicode** содержится более 96 000 символов. Система Unicode совместима с более широким стандартом набора символов **ISO/IEC 10646**, разработанном ISO (International Organization for Standardization) совместно с IEC (International Electrotechnical Commission).

Компьютерная платформа, использующая один из этих наборов в качестве своего базового набора символов, может употреблять 16- и даже 32-разрядное представление типа `char`.

В языке C понятие ***байт*** **определяется как число разрядов, используемых для представления типа `char`**. Поэтому, например, в системе с 16-разрядным представлением типа `char` в байте содержится не 8, а 16 бит.

Примеры объявления символьных переменных:

```
char c;
char letter, digit;
```

Символьные константы

Значения символьных констант заключаются в апострофы (одиночные кавычки). Пример:

```
char c;
c = 'A'; /* Правильно, 'A' – символьная константа */
c = A; /* Неправильно! Здесь A – идентификатор (имя переменной) */
c = "A"; /* Неправильно! Здесь "A" – строка, заканчивающаяся символом \0 */
```

или при инициализации

```
char c = 'A'; /* Правильно, 'A' – символьная константа */
```

Символы хранятся как числовые значения, которые можно указывать непосредственно. Например:

```
char c = 65; /* Правильно в кодировке ASCII, но это плохой стиль */
```

Однако такой код теряет универсальность, он будет корректно работать только в системах с кодировкой ASCII.

Важно помнить, что язык С рассматривает символьные константы как тип `int`, а не `char`. Например, в системе с 32-разрядным `int` и 8-разрядным `char`, объявление

```
char symb = 'D';
```

представляет `'D'` как число `68`, хранящееся в 32-разрядной ячейке памяти. При этом переменная `symb` помещает в свою 8-разрядную ячейку только младшие 8 бит из 32 бит. Это позволяет выполнить такое присваивание:

```
char symb = 'ABCD';
```

при котором все четыре 8-разрядных ASCII-кода хранятся в 32-разрядной ячейке типа `int`. Но при присваивании переменной типа `char` будут скопированы только младшие 8 разрядов, т.е. переменная `symb` примет значение `'D'`.

Непечатные символы

Не все символы имеют графическое представление, их нельзя отобразить на экране или бумаге. Для их представления можно использовать несколько способов.

1) Непосредственно указать ASCII-код:

```
char beep = 7; /* ASCII-код 7 – выдача звукового сигнала */
```

но это приведет к потере универсальности программы.

2) Использовать ***управляющие последовательности***, приведенные в таблице

Управл. послед.	**Описание**	**Перемещение курсора**
`\a`	предупреждение (сигнал зависит от оборудования)	не меняет позиции курсора
`\b`	возврат на одну позицию влево (backspace)	на один символ назад
`\f`	перевод страницы	в начало следующей страницы
`\n`	новая строка	в начало новой строки
`\r`	возврат каретки	в начало текущей строки
`\t`	горизонтальная табуляция	в следующую точку гориз.табуляции
`\v`	вертикальная табуляция	в следующую точку вертик.табуляции
`\\`	обратная косая черта (backslash)	
`\'`	одиночная кавычка '	
`\"`	двойная кавычка "	
`\?`	знак вопроса ?	
`\0`**`oo`**	восьмеричное значение (**`oo`** – две восьмеричные цифры)	
`\x`**`hh`**	шестнадцатиричное значение (**`hh`** – две шестнадцатиричные цифры)	

ASCII-код коды можно указывать в 8-ричной или 16-ричной системах счисления. Например, символу <Ctrl+Z> соответствует десятичный ASCII-код 26:

```
char ch;
ch = '\032'; /* ASCII-код 26 в 8-ричной системе счисления */
ch = '\x1A'; /* ASCII-код 26 в 16-ричной системе счисления */
```

Обратим внимание, что непосредственное указание кода символа

```
ch = 032; /* возможно, но нежелательно! */
```

является нежелательным, поскольку затрудняет понимание программы и может привести к потере ее универсальности.

Для 8-ричной системы счисления нулевые старшие разряды (и даже ведущий 0) могут быть опущены, например, все три записи '\007', '\07' и '\7' – эквивалентны.

В 16-ричной системе используется символ x или X, при котором можно указать от одной до трех цифр. Например, символ с десятичным кодом 15 можно записать несколькими способами: '\x00F', '\x0F', '\xF', '\X00F', '\X0F', '\XF'.

Вывод символов

Спецификатор стандартной функции printf() определяет, как будут отображаться данные при выводе на печать:

```
#include <stdio.h>
int main(void) {
     char ch;
     printf("Введите символ (после ввода нажмите <Enter>).\n");
     scanf("%c", &ch);
     printf("Код символа %c равен %d.\n", ch, ch);
     return 0;
}
```

здесь, например, при вводе символа R будет выведено сообщение:

```
Код символа R равен 82.
```

Строки символов

Понятие строки и нулевой символ

В языке C нет отдельного типа данных для строк. ***Строка*** — это одномерный массив символов (типа char), хранящихся в смежных ячейках памяти, заканчивающийся ***нулевым (null) символом***.

Признаком окончания строки (нулевым символом) служит символ '\0'. Это — не цифра нуль, а непечатный символ, значение которого в кодировке ASCII (или другой) равно 0.

Таким образом, строка содержит символы, составляющие строку, а также нулевой символ. Это единственный вид строки, определенный в C. Все строки всегда сохраняются с нулевым символом в конце.

Строковая константа

Записанная в тексте программы строка символов, заключенных в двойные кавычки, является строковой константой, например,

```
"некоторая строка"
```

В конец строковой константы компилятор автоматически добавляет нулевой символ:

н	е	к	о	т	о	р	а	я	␣	с	т	р	о	к	а	\0

Строковая константа хранится в неименованном массиве символов.

Различие между строками и символами:

Символьная константа `'a'`:

a

Строковая константа `"a"`:

a	\0

Объявление строки

Объявляя массив символов, предназначенный для хранения строки, необходимо предусмотреть место для нулевого символа, т.е. указать его размер в объявлении на один символ больше, чем наибольшее предполагаемое количество символов. Например, объявление массива `str`, предназначенного для хранения строки из 10 символов, должно выглядеть так:

```
char str[11];
```

Последний, 11-й байт предназначен для нулевого символа `'\0'`.

Использование строк

Для использования строк в стандартных функциях `printf()` и `scanf()` применяется спецификатор `%s`. Пример:

```
#include <stdio.h>
#define HOWAREYOU "Как дела?"
int main(void) {
    char name[21];
    printf("Введите свое имя (не более 20-и символов).\n");
    scanf("%s", name);
    printf("Привет, %s! %s\n", name, HOWAREYOU);
    return 0;
}
```

здесь компилятор сам добавит нулевой символ `'\0'` в константу `HOWAREYOU`. А функция `scanf()` сама поместит символ `'\0'` в массив `name` при вводе.

Обратите внимание, что параметр `name` функции `scanf()` указывается без операции `&`, поскольку он является именем массива.

Следует отметить, что при считывании введенных данных функция `scanf()` останавливает чтение на одном из ***«пробельных» символов***, которыми являются

- ***пробел,***
- ***табуляция,***
- ***новая строка.***

Если, например, пользователь введет имя «**Иван Петров**» с пробелом между словами, то в массив `name` будет помещено

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
И	**в**	**а**	**н**	\0																

а на экран будет выведено:

```
Привет, Иван! Как дела?
```

Модель ввода-вывода

Потоки

Средства ввода-вывода не являются частью самого языка С, а содержатся в стандартной библиотеке функций ввода-вывода. В стандарте ANSI C все библиотечные функции точно определены, и их использование гарантирует **переносимость** программ.

Стандартная библиотека реализует простую модель текстового ввода-вывода. Текстовый ввод-вывод, независимо от его физического источника или места назначения, выполняется над потоками символов.

Поток (stream) символов — это последовательность символов, разбитых на строки; каждая строка заканчивается символом конца строки `'\n'` и может быть пустой или содержать некоторое количество символов.

За то, чтобы привести каждый поток ввода или вывода в соответствие с этой моделью, отвечает стандартная библиотека, а программисту нет нужды заботиться о том, как строки потока представляются вовне программы.

Поток можно ассоциировать с файлами или другими периферийными устройствами. Язык С рассматривает *файлы* и *периферийные устройства ввода-вывода* как эквивалентные категории.

Поток присоединяется к файлу или устройству путем **открытия** потока, соединение разрывается путем **закрытия** потока. Операции чтения и записи данных в файл (или устройство) осуществляются через этот поток.

В начале выполнения любой программы открывается и предоставляются в ее использование три потока: `stdin`, `stdout`, `stderr`.

Буферизация

Стандарт ANSI C требует, чтобы ввод (и вывод) был буферизированным. Это значит, что введенные символы накапливаются и хранятся в ***буфере*** (временной области памяти), а не сразу становятся доступны для программы.

Буферизация реализуется в двух видах:

- ***полностью*** *буферизированный ввод-вывод* – в момент заполнения буфера его содержимое становится доступным программе, а буфер очищается***;***
- ***построчно*** *буферизированный ввод-вывод* – нажатие клавиши **<Enter>** приводит к тому, что введенный блок символов становится доступным для программы и буфер очищается.

Второй вариант используется гораздо чаще.

Преимущества использования буферизации:

- существенное увеличение скорости обмена данными;
- возможность исправления в случае опечатки (например, при наборе с клавиатуры).

Размер буфера ограничен и зависит от конкретной вычислительной системы. Обычно он составляет не менее 512 байтов, т.е. данные передаются в программу порциями.

Стандартные функции ввода-вывода

Односимвольные функции

В стандартной библиотеке `<stdio.h>` имеется ряд функций для чтения или записи одного символа за одну операцию; простейшими из них являются `getchar()` и `putchar()`.

Каждый раз при вызове `getchar()` эта функция считывает следующий символ текстового потока ввода и возвращает его в качестве своего значения. Функция `getchar()` используется без параметров:

```
char ch;
ch = getchar();
```

переменная `ch` будет содержать следующий символ входного потока. Выполнение этого оператора дает тот же результат, что и выполнение:

```
scanf("%c", &ch);
```

Функция `putchar()` при каждом вызове выводит один символ:

```
putchar(ch);
```

выводит значение целочисленной переменной `ch` в виде символа. Выполнение этого оператора дает тот же результат, что и выполнение:

```
printf("%c", ch);
```

Вызовы `putchar()` и `printf()` можно чередовать как угодно – выводимые данные будут следовать в том порядке, в каком выполняются вызовы.

Символьные функции `ctype.h`

Стандартный набор функций для анализа символов описан в заголовочном файле `<ctype.h>`.

Имя функции	Функция возвращает ИСТИННОЕ значение, если аргумент
`isalnum()`	буква или цифра
`isalpha()`	буква
`isblank()`	пробельный символ
`iscntrl()`	управляющий символ (например, **<Ctrl+B>**)
`isdigit()`	цифра
`isgraph()`	любой печатный символ, отличный от пробельного
`islower()`	символ в нижнем регистре

Имя функции	Функция возвращает ИСТИННОЕ значение, если аргумент
`isprint()`	печатный символ
`ispunct()`	знак пунктуации (не буква, не цифра, не пробельный символ)
`isspace()`	любой непечатный и пробельный символ
`isupper()`	символ в верхнем регистре
`isxdigit()`	шестнадцатиричная цифра
Имя функции	**Действие**
`tolower()`	перевод символа в нижний регистр
`toupper()`	перевод символа в верхний регистр

Строковые функции `gets()` *и* `puts()`

В стандартной библиотеке `<stdio.h>` имеются функции для чтения или записи строк за одну операцию: `gets()` и `puts()`.

При вызове функции `gets()` она считывает из потока ввода последовательность символов до тех пор, пока не встретится символ новой строки `'\n'`, который генерируется при каждом нажатии клавиши **<Enter>**. Все считанные символы (кроме `'\n'`) функция `gets()` помещает в строку, добавляя в конце символ `'\0'`. Пример:

```
#include <stdio.h>
int main(void) {
     char st[21];    /* ограничение на размер строки */
     gets(st);    /* извлечение строки из потока ввода */
     printf("%s\n", st);    /* помещение строки в поток вывода */
     return 0;
}
```

Функция `gets()` использует адрес, указанный в аргументе, для загрузки строки в массив, а также сама возвращает адрес введенной строки:

```
#include <stdio.h>
int main(void) {
     char st[21];
     char *pst;
     printf("Введите строку (после ввода нажмите <Enter>).\n");
     pst = gets(st);
     printf("Строка %s - это то же, что строка %s.\n", st, pst);
     return 0;
}
```

Конструкция типа:

```
while (gets(st) != NULL) {
     ...}
```

позволяет одновременно считывать значение и отслеживать символ конца файла.

К сожалению, функция `gets()` **не выполняет** проверку того, что введенная последовательность символов умещается в выделенную область памяти. В случае переполнения лишние символы попадают в соседние области памяти (вместе с добавляемым символом `'\0'`). Поэтому использование `gets()` является **небезопасным.**

Вместо `gets()` обычно используют ее аналог для файлового ввода-вывода – функцию `fgets()`, описание которой приведено в разделе о файловом вводе-выводе. Например, ввод строки

```
char st[21];
gets(st);
```

может быть заменен на:

```
char st[21];
fgets(st, 21, stdin);
```

Здесь у функции `fgets()` три параметра:

- `st` – имя массива символов, в который помещается считываемая строка;
- `21` – максимальное количество символов для считывания, увеличенное на один (для символа `'\0'`), то есть будет считано не более `20` символов;
- `stdin` – считывание происходит из стандартного потока ввода (с клавиатуры).

В отличие от `gets()` функция `fgets()` сохранит символ новой строки (`'\n'`).

Функция `puts()` помещает в поток вывода строку, адрес которой указан в аргументе:

```
#include <stdio.h>
int main(void) {
    char st[21] = "это некоторая строка";
    puts("слово");
    puts(st);
    puts(&st[4]);
    return 0;
}
```

при этом функция `puts()` добавляет в конце символ новой строки. Результат:

```
слово
это некоторая строка
некоторая строка
```

Выполнение оператора вызова функции `puts()`

```
puts(st);
```

дает тот же результат, что и:

```
printf("%s\n", st);
```

Строковые функции `string.h`

Стандартный набор функций для работы со строками описан в заголовочном файле `<string.h>`.

Имя функции	Действие
`strcpy(s1,s2)`	Копирование `s2` в `s1`
`strcat(s1,s2)`	Конкатенация (присоединение) `s2` в конец `s1`
`strlen(s1)`	Возвращает длину строки `s1`
`strcmp(s1,s2)`	Возвращает `0`, если `s1` и `s2` совпадают, отрицательное значение, если `s1<s2` и положительное значение, если `s1>s2`
`strchr(s1,ch)`	Возвращает указатель на первое вхождение символа `ch` в строку `s1`
`strstr(s1,s2)`	Возвращает указатель на первое вхождение строки `s2` в строку `s1`

Пример:

```
#include <stdio.h>
#include <string.h>
int main(void)
{
  char s1[80], s2[80];
  gets(s1);
  gets(s2);
  printf("Длина: %d %d\n", strlen(s1), strlen(s2));
  if(!strcmp(s1, s2)) printf("Строки равны\n");
  strcat(s1, s2);
  printf("%s\n", s1);
  strcpy(s1, "Проверка.\n");
  puts(s1);
  if(strchr("Алло", 'л')) printf(" л есть в Алло\n");
  if(strstr("Привет", "ив")) printf(" найдено ив");
  return 0;
}
```

Если эту программу выполнить и ввести в `s1` и в `s2` одну и ту же строку `"Алло!"`, то на экран будет выведено следующее:

```
Длина: 5 5
Строки равны
Алло!Алло!
Проверка.
 л есть в Алло
 найдено ив
```

Следует помнить, что `strcmp()` принимает значение `0` (ЛОЖЬ), если строки совпадают.

Функции преобразования строк в числа

В стандартной библиотеке `<stdlib.h>` имеются функции, которые преобразуют строки цифр в числовые значения.

Имя функции	Действие
`atoi(st)`	Возвращает значение типа `int`
`atol(st)`	Возвращает значение типа `long int`
`atof(st)`	Возвращает значение типа `double`

При преобразовании символы берутся из начала строки `st` (при этом пробельные символы пропускаются) до первого символа, не являющегося частью числа. Если число в строке не найдено, то функции возвращают нулевое значение.

Пример:

```
#include <stdio.h>
#include <stdlib.h>
int main(void) {
  int i;
  double x;
```

```
  char st[15] = " -12345.6.7abc";
  i = atoi(st);
  printf("%d\n", i+4);
  x = atof(st);
  printf("%lf\n", x+4.4);
  return 0;
}
```

Результат выполнения программы:

```
-12341
-12341.200000
```

Реализация ввода-вывода

Посимвольный ввод-вывод

Копирование входного потока в выходной по одному символу (эхо-повтор ввода).

```
#include <stdio.h>
int main(void) {
     char c;
     c = getchar();          /* считать символ */
     while (c != '\n') {     /* пока это не символ конца строки */
          putchar(c);        /* вывести символ */
          c = getchar();     /* считать следующий символ */
     }
     putchar(c);             /* вывести символ новой строки */
     return 0;
}
```

Считывание символа и проверку условия можно совместить. Изменим выделенный блок:

```
     while ((c = getchar()) != '\n') /* сочетание двух действий */
          putchar(c);                /* вывести символ */
     putchar(c);             /* вывести символ новой строки */
```

Во входном потоке могут содержаться не только коды символов, но и другие числовые значения, не совпадающие ни с одним из возможных значений типа `char`. Например, если поток читается из файла, то в конце помещается признак конца файла – `EOF` (**end of file**), описанный в `<stdio.h>`. Тогда программу нужно изменить:

```
#include <stdio.h>
int main(void) {
     int c; /* тип int имеет более широкий диапазон значений, чем char */
     while ((c = getchar()) != EOF) /* пока не найден конец файла */
          putchar(c);
     putchar(c);
     return 0;
}
```

Подсчет количества символов

Определение количества символов во входном потоке.

```
#include <stdio.h>
int main(void) {
    long int nc = 0L;    /* счетчик количества символов */
    while (getchar() != EOF)
        ++nc;
    printf("В потоке %ld символов.\n", nc);
    return 0;
}
```

Вместо цикла `while` можно использовать `for`:

```
    long int nc;
    for (nc = 0L; getchar() != EOF; ++nc)
        ;
```

Подсчет количества слов и строк

Определение количества символов, слов и строк во входном потоке. Здесь «словом» считается последовательность символов, не содержащая пробельные символы.

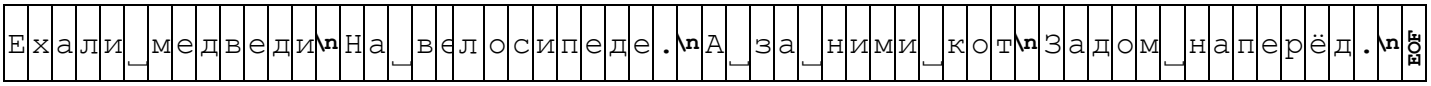
```
#include <stdio.h>
#include <ctype.h>
#define IN 1     /* внутри слова */
#define OUT 0    /* снаружи слова */
int main(void) {
    int  c,      /* текущий символ */
         state,  /* состояние потока (внутри или снаружи слова) */
         nc,     /* количество символов (number of characters) */
         nw,     /* количество слов (number of words) */
         nl;     /* количество строк (number of lines) */
    state = OUT;
    nc = nw = nl = 0;
    while ((c = getchar()) != EOF) {  /*пока не найден конец файла*/
        ++nc;
        if (c == '\n')    /* если текущий символ - конец строки */
            ++nl;
        if (isspace(c))   /* если текущий символ - пробельный */
            state = OUT;  /*текущий символ находится снаружи слова*/
        else if (state == OUT) {  /*если предыдущий символ снаружи слова*/
            state = IN;  /* текущий символ находится внутри слова */
            ++nw;
        }
    }
    printf("Символов: %d; слов: %d; строк: %d.\n", nc, nw, nl);
    return 0;
}
```

При этом, если последняя строка не оканчивается символом `'\n'`, а обрывается маркером конца файла `EOF` , то она не будет засчитана в общем количестве строк.

Допустим, нужно обработать текст стихотворения:

```
Ехали медведи
На велосипеде.
А за ними кот
Задом наперёд.
```

В стандартном потоке ввода (`stdin`) должна находиться последовательность символов:



Тогда результатом работы программы будет:

```
Символов: 58; слов: 10; строк: 4.
```

Перенаправление ввода и вывода

Приведенный текст стихотворения можно поместить в поток несколькими способами.

1) Ввод с клавиатуры. В этом случае необходимо провести эмуляцию ввода символа `EOF`. Это можно сделать нажатием определенной комбинации клавиш, которая может различаться в разных вычислительных системах.

Например, в большинстве систем на базе Unix нажатие клавиш **<Ctrl+D>** в начале строки вызывает передачу конца файла. Многие системы в консольном режиме (в том числе, основанные на MS-DOS) используют для той же цели комбинацию клавиш **<Ctrl+Z>**.

2) Перенаправление потоков средствами операционной системы. Многие операционные системы (ОС) позволяют использовать файлы для входных и выходных данных вместо клавиатуры и экрана. Для этого откомпилированную программу нужно вызвать из командной строки с операцией ***перенаправления*** и указать имена файлов, с которыми будут связаны потоки. В Unix, Linux и MS-DOS эти операции обозначаются символами:

< – перенаправление ввода из файла,

> – перенаправление вывода в файл.

Пусть `prog` является именем исполняемой программы (в MS-DOS – `prog.exe`), а `file1` и `file2` – имена файлов. Тогда вызов команд (в командной строке ОС):

Команда ОС	Действие
`prog <file1`	перенаправление ввода из файла `file1`
`prog >file2`	перенаправление вывода в файл `file2`
`prog <file1 >file2`	одновременное перенаправление ввода из файла `file1` и вывода в файл `file2`
`prog >file2 <file1`	

3) Чтение/запись данных из файла с помощью стандартных средств языка С. Этот способ позволяет организовать взаимодействий с файлами в программном коде с помощью вызова стандартных функций, которые будут рассмотрены позднее.

Создание пользовательского интерфейса

Буферизированный ввод

Программа, отгадывающая число в результате диалога с пользователем.

```
#include <stdio.h>
int main(void) {
     int number = 1;
     char response;
     printf("Загадайте целое число от 1 до 10.\n");
     printf("Нажмите клавишу y, если загаданное число верно и");
     printf("\n клавишу n в противном случае.\n");
     printf("Вашим числом является %d?\n", number);
     while ((response = getchar()) != 'y') { /* получить ответ */
          if (response == 'n')
               printf("Вашим числом является %d?\n", ++number);
          else
               printf("Следует вводить только y или n.\n");
          while (getchar() != '\n')
               continue; /* пропустить оставшуюся часть входной строки*/
     }
     printf("Число отгадано.\n");
     return 0;
}
```

Результат работы программы может иметь следующий вид:

```
Загадайте целое число от 1 до 10.
Нажмите клавишу y, если загаданное число верно и
 клавишу n в противном случае.
Вашим числом является 1?
n
Вашим числом является 2?
no
Вашим числом является 3?
no sir
Вашим числом является 4?
forget it
Следует вводить только y или n.
n
Вашим числом является 5?
y
Число отгадано.
```

Числовой и символьный ввод

Программа, которая печатает символы по строкам и столбцам.

```
#include <stdio.h>
void display(char symb, int lines, int width);
int main(void) {
     int ch = 1;          /* символ, выводимый на печать */
     int rows, cols;      /* количество строк и столбцов */
```

```
    printf("Введите символ и два целых числа:\n");
    while ((ch = getchar()) != '\n') {
        if (scanf("%d %d", &rows, &cols) != 2)
            break;
        display(ch, rows, cols);
        while (getchar() != '\n')
            continue;
        printf("Введите еще один символ и два целых числа;\n");
        printf("введите <Enter> для завершения программы.\n");
    }
    printf("Программа завершена.\n");
    return 0;
}
void display(char symb, int lines, int width) {
    int row, col;
    for (row = 1; row <= lines; row++) {
        for (col = 1; col <= width; col++)
            putchar(symb);
        putchar('\n');
    }
}
```

Результат работы программы:

```
Введите символ и два целых числа:
c 1 2
cc
Введите еще один символ и два целых числа;
введите <Enter> для завершения программы.
! 3 6
!!!!!!
!!!!!!
!!!!!!
Введите еще один символ и два целых числа;
введите <Enter> для завершения программы.

Программа завершена.
```

Проверка допустимости ввода

Функция, которая проверяет ввод целого числа.

```
int getint(void) {
    int input;
    char ch;
    while (scanf("%d", &input) != 1) {
        while ((ch = getchar()) != '\n')
            putchar(ch); /* удаление некорректных входных данных */
        printf(" - ввод не является целочисленным.\n"
               "Пожалуйста, введите целое число,\n"
               "такое как 25, -178 или 3.\n");
    }
    return input;
}
```

Вызов данной функции:

```
...
int n;
...
n = getint();
...
```

Файловый ввод-вывод

Обмен данными с файлами

Файл – именованный раздел памяти, обычно на диске. В языке С обмен данными с файлом происходит через ассоциированный с ним ***поток***. Поэтому для программиста ***файл*** – это непрерывная последовательность байтов, каждый из которых может быть прочитан индивидуально.

Стандарт ANSI C поддерживает два способа ***представления*** файлов:

- ***текстовый поток,***
- ***двоичный поток.***

Пример различия представлений. Конец строки в текстовых и двоичных потоках:

Операционная система	Конец строки в файле (или двоичном потоке)	Конец строки в текстовом потоке ANSI C
MS-DOS	`\r\n`	`\n`
Macintosh	`\r`	
Unix	`\n`	

В Unix оба представления не отличаются друг от друга.

Работа с файлом с использованием базовых функций ввода-вывода операционной системы (ОС) называется ***низкоуровневый ввод-вывод***, в стандарте языка С он не поддерживается.

Для переносимости модели ввода-вывода стандарт ANSI C поддерживает только ***стандартный высокоуровневый ввод-вывод.***

Программа на языке С автоматически открывает три потока:

- ***стандартный ввод*** (`stdin`),
- ***стандартный вывод*** (`stdout`),
- ***стандартный вывод ошибок*** (`stderr`).

Стандартный ввод и вывод ***буферизированы*** – данные передаются порциями. Размер буфера – не менее 512 байтов (или кратное 512 количество байтов).

Функция `fopen()`

Для того чтобы начать работу с файлом, с ним нужно связать поток с помощью стандартной функции `fopen()`, которая объявлена в `<stdio.h>`. После такого связывания файлом для программиста будет являться соответствующий ему поток (текстовый или двоичный). Непосредственное обращение к файлу, минуя поток, в стандарте языка С не предусмотрено.

Функция `fopen()` имеет два аргумента: первый – адрес строки, содержащей имя файла, второй – строка, определяющая режим открытия.

Строка режима	Описание
`"r"`	Открыть текстовый файл для **чтения** (**read**).
`"w"`	Открыть текстовый файл для **записи** (**write**), при этом его **длина усекается до нуля**. Если такого файла не существует, то он создается.
`"a"`	Открыть текстовый файл для **записи**, **добавляя данные в конец** (**append**). Если такого файла не существует, то он создается.
`"r+"`	Открыть текстовый файл для **обновления** (чтения и записи).
`"w+"`	
`"a+"`	
`"rb" "wb" "ab"` `"rb+" "r+b"` `"wb+" "w+b"` `"ab+" "a+b"`	То же самое, но вместо текстового режима доступа – **двоичный** (binary).

После успешного открытия файла функция `fopen()` возвратит указатель на файл, который смогут использовать все другие функции работы с файлами.

```
FILE *fp;  /* указатель на файл */
fp = fopen("myfile.txt","r");
```

Указатель на файл `fp` имеет особый тип `FILE` – это производный тип ***«указатель на файл»***, определенный в `<stdio.h>`. Он указывает на пакет, содержащий информацию о файле.

Если функция `fopen()` не может открыть файл, то она возвращает пустой указатель `NULL`.

Пример открытия файла с проверкой успешности этой операции:

```
FILE *fp;
if ( (fp = fopen("myfile.txt","r")) == NULL ) {
     printf("Не удается открыть файл.\n");
     exit(1); /* завершение программы с возвратом кода ошибки 1 */
}
```

Функция `fclose()`

После завершения работы с файлом, связанный с ним поток необходимо закрыть, используя стандартную функцию `fclose()`.

На уровне операционной системы функция `fclose()` закрывает файл, ассоциированный с указателем `fp`, буферы при этом очищаются. Эта функция возвращает `0`, если операция закрытия выполнилась успешно, и `EOF` в противном случае.

```
fclose(fp);
```

Функция `exit()`

Стандартная функция `exit()`, описанная в `<stdlib.h>`, вызывает завершение программы и закрытие всех открытых файлов. По традиционному соглашению между программистами возвращает `0` в случае успешного завершения, ненулевое значение (код ошибки) – в случае аварийного завершения.

Оператор

```
return 0;
```

почти эквивалентен оператору вызова функции

```
exit(0);
```

Исключением являются два случая.

1) При рекурсивном вызове функции `return` возвращает управление на предыдущий уровень рекурсии, а `exit()` завершит работу всей программы.

2) `exit()` завершит выполнение программы, даже будучи вызван из функции, отличной от функции `main()`.

Функции файлового ввода-вывода

Сопоставление функций файлового ввода-вывода с их аналогами:

Файловая функция	Аналог
`getc(fp)`	`getchar()`
`putc(ch, fp)`	`putchar(ch)`
`fprintf(fp, ...)`	`printf(...)`
`fscanf(fp, ...)`	`scanf(...)`
`fgets(buf, MAX, fp)`	`gets(buf)`
`fputs(buf, fp)`	`puts(buf)`

Здесь подразумевается, что переменная `buf` является массивом символов, например:

```
char buf[100];
```

Функция `getc()` считывает символ из файла `fp`. Она возвращает значение типа `int`, но код символа находится в младшем байте, а старшие байты обнулены.

Если достигнут конец файла, то `getc()` возвращает значение `EOF`.

Признак конца файла `EOF`

Чтение текстового файла и вывод его на экран посимвольно:

```
int ch;     /* текущий прочитанный символ */
FILE *fp;   /* указатель на файл */
fp = fopen("myfile.txt","r");  /* открытие файла */
ch = getc(fp);          /* получить первый символ из файла */
while (ch != EOF) {   /* проверка признака конца файла */
     putchar(ch);     /* вывод на экран */
     ch = getc(fp);   /* получить следующий символ из файла */
}
fclose(fp);  /* закрытие файла */
```

Этот программный код может быть записан более лаконично – с вызовом функции `getc()` в заголовке цикла `while`:

```
int ch;
FILE *fp;
fp = fopen("myfile.txt","r");
while ((ch = getc(fp)) != EOF)
     putchar(ch);
fclose(fp);
```

Функции `fgets()` *и* `fputs()`

У функции `fgets()` второй аргумент `MAX` – целое число, определяющее максимальный размер входной строки. Функция читает входные данные

- до первого символа новой строки,
- пока не будет прочитано количество символов, на единицу меньшее `MAX`,
- пока не будет найден признак конца файла `EOF`.

В прочитанную строку добавляется нулевой символ `'\0'`. Если считана целая строка (длина которой меньше `MAX-1`), то перед нулевым символом добавляется символ новой строки. В этом работа функции `fgets()` отличается от `gets()`.

Также функция `fputs()`, в отличие от `puts()`, при выводе не добавляет символ новой строки. Поэтому с функциями `fgets()` и `fputs()` удобно работать в тандеме.

Функции `feof()` *и* `ferror()`

Существуют ситуации, когда функция `getc()` возвращает `EOF`, но конец файла не достигнут:

1) если проводится чтение из двоичного файла и прочитанное целое число оказалось равным коду `EOF`;
2) в случае возникновения ошибки при чтении функция `getc()` тоже возвращает значение `EOF`.

Для преодоления таких ситуаций имеется функция `feof()`, которая возвращает ненулевое значение, если обнаружен маркер конца файла, и нуль – в противном случае.

Пример чтения двоичного файла:

```
int a;       /* текущее прочитанное число */
FILE *bfp;   /* указатель на файл */
bfp = fopen("binaryfile.dat","rb"); /* открытие файла */
while (!feof(bfp)) /* пока не обнаружен маркер конца файла */
     a = getc(bfp);
fclose(bfp); /* закрытие файла */
```

Имеется другой способ отслеживания ошибок при чтении/записи – это функция `ferror()`, которая возвращает ненулевое значение, если произошла ошибка:

```
getc(fp);
if (!ferror(fp)) {
     printf("Ошибка чтения/записи файла.\n");
     exit(1);
}
```

Аргументы командной строки

Чтение файла и подсчет количества символов в нем. Программа содержится в файле `prog.c`:

```
#include <stdio.h>
#include <stdlib.h>    /* функция exit() */
```

```
int main(int argc, char *argv[]) {
     int ch;     /* текущий прочитанный символ */
     FILE *fp;   /* указатель на файл */
     long int count = 0;
     if (argc != 2) {
          printf("Синтаксис команды: %s filename\n", argv[0]);
          exit(1);
     }
     if ((fp=fopen(argv[1],"r")) == NULL) {
          printf("Не удается открыть файл %s\n", argv[1]);
          exit(2);
     }
     while ((ch=getc(fp)) != EOF) {
          putc(ch, fp);
          count++;
     }
     if (fclose(fp) != 0)
          printf("Ошибка при закрытии файла %s\n", argv[1]);
     printf("Файл %s содержит %ld символов.\n", argv[1], count);
     return 0;
}
```

Эта программа может быть откомпилирована и вызвана из командной строки с параметром – именем файла, в котором содержится текст (`myfile.txt`). Например, для MS-DOS:

```
D:\>prog.exe myfile.txt
```

Заголовок функции `main()`:

```
int main(int argc, char *argv[])
```

`argc`	Количество аргументов (включая имя программы)
`argv[]`	Массив аргументов, элементами которого являются строки
`argv[0]`	Первый аргумент – имя запускаемой программы `prog.exe`
`argv[1]`	Второй аргумент – имя файла с текстом `myfile.txt`

Возможность распознавать аргументы в командной строке отсутствует в некоторых операционных системах и поэтому не является полностью переносимой.

Произвольный доступ к файлу

Функции непосредственного доступа к произвольной части файла:

Функция	Действие
`fseek(fp, Смещение, ОтправнаяТочка)`	перемещает текущую позицию в файле
`ftell(fp)`	возвращает текущую позицию в файле (типа `long int`)

`Смещение` – значение типа `long int`, которое показывает, на сколько необходимо переместиться относительно отправной точки. Оно может быть положительным (движение вперед), отрицательным (движение назад), нулевым (оставаться на месте).

`ОтправнаяТочка` – режим, идентифицирующий отправную точку, описанный в заголовочном файле `<stdio.h>`:

Режим	Отправная точка
`SEEK_SET`	начало файла
`SEEK_CUR`	текущая позиция
`SEEK_END`	конец файла

Примеры переноса текущей позиции в файле:

```
fseek(fp, 0L, SEEK_SET);    /* в начало файла */
fseek(fp, 10L, SEEK_SET);   /* на 10 байтов вперед от начала файла */
fseek(fp, -10L, SEEK_END); /* на 10 байтов назад от конца файла */
```