Введение в C++

Ввод и вывод в Си

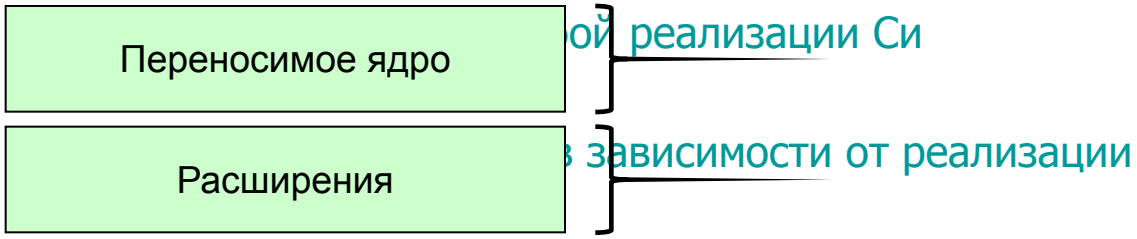
Ввод и вывод в Си

Организация ввода/вывода

В Си нет операторов ввода/вывода. Это делает ядро языка независимым от аппаратуры компьютера. Операции ввода и вывода реализуются путем обращения к библиотечным функциям.

Вообще, стандартная библиотека Си есть собрание общеупотребительных и полезных функций. Они хранятся в библиотеке в откомпилированном виде, готовые к использованию. Чтобы компилятор мог контролировать правильность обращения к библиотечным функциям, в заголовочных файлах даны их прототипы.

Стандартная библиотека Си условно состоит из двух частей:

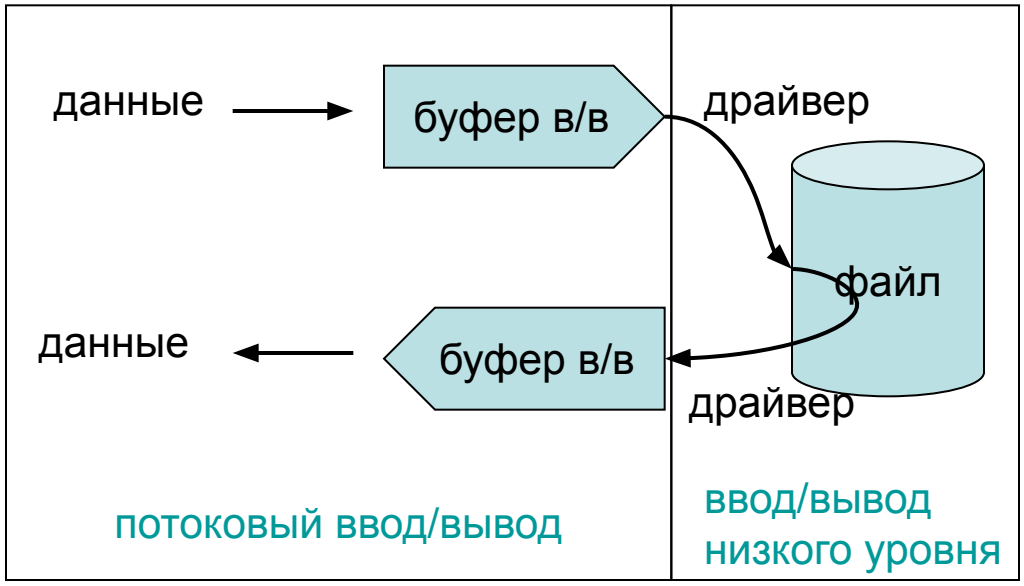


Под файлом (в узком смысле) будем понимать поименованную область долговременной памяти компьютера.

Слово «файл» также употребляют, когда говорят о потоке данных, связанном с устройством ввода/вывода: клавиатурой, монитором, принтером, модемом, др.

Ввод и вывод в Си

Есть функции ввода/вывода высокого и низкого уровня. Ввод/вывод высокого уровня унифицирован и использует абстракцию «поток».



Потоки имеют уникальные имена, когда они связаны с реальными устройствами. В Си это достигается использованием так называемых указателей на файл.

Ввод и вывод в Си

В действительности указатель на файл есть указатель на структуру следующего вида:

```
typedef struct  {
    char *_ptr; // указатель на текущий байт в буфере ввода/вывода
    int _cnt;     // счетчик байтов в буфере ввода/вывода
    char *base;// адрес буфера ввода/вывода
    char  _flag; // флаги доступа
    char  _fd;   // дескриптор файла (для ввода/вывода низкого уровня)
}  FILE;
```

В системе всегда имеется массив структур типа FILE:

```
extern FILE _iob [ 20 ];   // можно иметь открытыми до 20 файлов
```

К вашим услугам всегда есть несколько стандартных потоков ввода/вывода:

```
extern FILE *stdin;         // стандартный поток ввода
extern FILE *stdout;        // стандартный поток вывода
extern FILE *stderr;        // для сообщений об ошибках (не буферизован)
extern FILE *stdaux;        // резервный поток (не буферизован)
extern FILE *stdprn;        // стандартный поток для печати
```

Эти потоки всегда открыты, их можно перенаправлять на разные устройства.

Все прочие потоки нужно явным образом создавать (открывать файлы).

Ввод и вывод в Си

открытие файла:

```
FILE* fopen ( char *name, char *mode);
```

Здесь name – имя файла (полный путь), mode – режим использования:

- "r" - только чтение из файла
- "w" - только запись в файл
- "a" - только добавление записей в конец файла
- "r+" - чтение и запись (файл должен существовать)
- "w+" - запись и чтение (в пустой файл)
- "a+" - чтение и добавление записей в конец файла

В случае успешного открытия файла функция fopen возвращает указатель на структуру типа FILE. При ошибке возвращается значение NULL.

закрытие файла:

```
int fclose ( FILE *stream) ;
```

В случае успешного закрытия файла функция fclose возвращает 0.

При ошибке возвращается значение EOF.

EOF есть признак конца файла, эта константа описана в "stdio.h".

Обычно EOF - это длинная минус единица (-1L).

Ввод и вывод в Си

ввод из файла:

```
int getc ( FILE *stream );
```

Функция читает один символ из потока stream и возвращает его как результат.

При достижении конца файла или возникновении ошибки в/в возвращает EOF.

```
char* fgets ( char *s, int n, FILE *stream );
```

Функция читает строку символов длиной не более n символов из потока stream и размещает их начиная с адреса, на который указывает s. Необходимо позаботиться о выделении достаточной памяти для размещения вводимой строки. Ввод завершается, когда прочитан символ перевода строки '\n', исчерпан лимит количества вводимых символов n, достигнут конец файла или произошла ошибка в/в. При вводе символ перевода строки '\n' заменяется на признак конца строки '\0'.

В случае успеха функция возвращает указатель s. Значение NULL возвращается, если произошла ошибка в/в или был достигнут конец файла.

```
int fscanf ( FILE *stream, <список-форматов> [, <список-адресов>] );
```

Функция осуществляет форматный ввод полей данных из потока stream, в соответствии с заданным <списком-форматов>, и размещает введенные данные в памяти в соответствии со <списком-адресов>. В случае успеха функция возвращает количество введенных и размещенных в памяти полей данных.

При достижении конца файла или возникновении ошибки в/в возвращает значение EOF.

Ввод и вывод в Си

вывод в файл:

```
int putc ( int c, FILE *stream );
```

Функция выводит символ c в поток stream и возвращает выведенный символ как результат работы. При достижении конца файла или возникновении ошибки в/в возвращает EOF.

```
int fputs ( char *s, FILE *stream );
```

Функция выводит строку s в поток stream. В случае успеха функция возвращает некоторое неотрицательное число (обычно количество выведенных символов). Значение EOF возвращается, если произошла ошибка в/в.

```
int fprintf ( FILE *stream, <список-форматов> [, <список-данных>] );
```

Функция осуществляет форматный вывод данных в поток stream, в соответствии с заданным <списком-данных>, преобразуя данные в соответствии с заданным <списком-форматов>. В случае успеха функция возвращает количество введенных байтов. При возникновении ошибки в/в функция возвращает некоторое отрицательное значение.

Ввод и вывод в Си

особые ситуации:

```
int feof ( FILE *stream );
```

Функция возвращает некоторое ненулевое значение (истину), если при последней попытке чтения из потока stream был достигнут конец файла. Нулевое значение (ложь) возвращается, если конец файла не был достигнут.

```
int ferror ( FILE *stream );
```

Функция возвращает некоторое ненулевое значение (истину), если при работе с потоком stream произошла ошибка в/в. Нулевое значение (ложь) возвращается, если ошибок в/в при работе с потоком stream зафиксировано не было. Индикатор ошибки в/в сбрасывается после закрытия файла, связанного с данным потоком или после вызова специальной функции clearerr.

```
int ungetc ( int c, FILE *stream );
```

Функция «заталкивает» символ c в поток stream, открытый для чтения. Последующая операция чтения из потока stream выдаст символ c. В случае успеха функция возвращает символ c. При возникновении ошибки в/в поток stream не изменяется и функция возвращает значение EOF.

Ввод и вывод в Си

Пример: слияние файлов

Имена дисковых файлов заданы в командной строке.
Файлы последовательно открываются и их содержимое копируется в стандартный вывод.

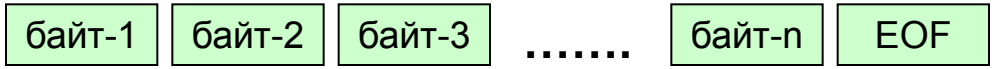
a.exe <имя-файла-1> <имя-файла-2> … <имя-файла-n>

```
#include <stdio.h>
void filecopy ( FILE *stream)  {        // копирование одного файла
int c;
    while  ( ( c = getc ( stream ) ) != EOF )  putc ( c, stdout );
}

void main ( int argc, char *argv[] )  {
FILE *stream;
    if ( argc == 1 ) filecopy ( stdin );  // если файлы не заданы, то stdin >> stdout
    else  while ( --argc )            // пока не исчерпан список файлов
       if ( ( stream = fopen ( *++argv, "r" ) ) == NULL )       // пытаемся открывать
           { fprintf ( stderr, "open err: %s\n", *argv ); break; }  // не открывается
       else { filecopy ( stream );  fclose ( stream ); }   // копируем и закрываем
}
```

Ввод и вывод в Си

Произвольный доступ к файлам

Файл можно рассматривать как последовательность байтов:



Когда файл открыт, с ним связан поток. В потоке всегда имеется внутренний указатель на текущий обрабатываемый байт. Фактически это смещение (в байтах) от начала файла. Это число типа long, оно всегда может быть получено:

```
long ftell ( FILE *stream );
```

Функция возвращает текущий внутренний указатель (т.е. номер текущего обрабатываемого байта) для потока stream или -1L при ошибке в/в.

Обычная обработка файлов – последовательная. Функция fseek() позволяет обращаться с дисковым файлом как с массивом байтов:

```
long fseek ( FILE *stream, long offset, int origin );
```

Функция позиционирует текущий внутренний указатель (т.е. задает номер текущего обрабатываемого байта) для потока stream. Возвращает 0 при успешном позиционировании или величину, отличную от нуля, при ошибке в/в. Аргумент offset задает смещение относительно некоторой позиции. Позиция задается аргументом origin.

Возможные значения origin описаны в <stdio.h>:

SEEK_SET – начало файла
SEEK_CUR – текущая позиция
SEEK_END – конец файла

Ввод и вывод в Си

Пример – что делает этот код?

```
#include <stdio.h>
#include <process.h>
main ( int argc, char *argv[] ) {        // разберитесь, что делает этот код?
FILE *stream;
long offset = 0L;
if ( argc < 2 ) { fprintf ( stderr, “usage: a.exe filename\n” ); exit (1); }
if ( ( stream = fopen ( *++argv, “r” ) ) == NULL )
        { fprintf ( stderr, “open err: %s\n”, *argv ); exit (2); }
while ( ! fseek ( stream, offset++, SEEK_SET ) )  {
        putchar ( getc ( stream ) );
        if ( ! fseek ( stream, -(offset+4), SEEK_END ) )
                putchar ( getc ( stream ) );
        }
        fclose ( stream );
}
```

Ввод и вывод в Си

Низкоуровневый ввод/вывод

Потоковый ввод/вывод почти всегда удобен. Однако имеются ситуации, когда требуется полный контроль со стороны приложения за передачей данных. Например: приложения реального времени, программирование модемов, сетей, др.

Иногда необходима передача данных без буферизации.

В этих случаях нужно использовать другую группу функций ввода/вывода.

Эти функции описаны в файле <io.h> - небуферизованный ввод/вывод низкого уровня.

открытие файла:

```
int open ( char *name, int flag );
```

В случае успеха данная функция возвращает небольшое положительное целое число, которое называется дескриптором файла. В случае ошибки возвращает -1.

возможные значения аргумента flag:

O_RDONLY	только чтение
O_WRONLY	только запись
O_RDWR	чтение и запись
O_APPEND	добавление в файл
O_CREAT	создание файла
O_TRUNC	усечение файла
O_TEXT	текстовый файл
O_BINARY	двоичный файл

стандартные потоки и их дескрипторы

0	stdin
1	stdout
2	stderr
3	stdaux
4	stdprn

Ввод и вывод в Си

закрытие файла:

```
int close ( int fd );
```

Здесь fd – дескриптор файла. В случае успеха данная функция возвращает 0, при ошибке возвращает -1.

ввод данных:

```
int read ( int fd, char *buf, unsigned count );
```

Функция пытается прочитать count байтов в буфер buf из файла с дескриптором fd. Возвращает количество действительно прочитанных байтов или -1 при ошибке в/в.

вывод данных:

```
int write ( int fd, char *buf, unsigned count );
```

Функция пытается записать count байтов из буфера buf в файл с дескриптором fd. Возвращает количество действительно записанных байтов или -1 при ошибке в/в.

Ввод и вывод в Си

произвольный доступ к файлу:

```
long tell ( int fd );
```

Функция возвращает текущий внутренний указатель (т.е. номер текущего обрабатываемого байта) для файла с дескриптором fd или -1 при ошибке в/в.

```
long lseek ( int fd, long offset, int origin );
```

Функция позиционирует текущий внутренний указатель (т.е. задает номер текущего обрабатываемого байта) для файла с дескриптором fd. Возвращает 0 при успешном позиционировании или величину, отличную от нуля, при ошибке в/в.

Аргумент offset задает смещение относительно некоторой позиции.

Позиция задается аргументом origin.

Возможные значения origin описаны в <io.h> и <stdio.h>:

SEEK_SET – начало файла

SEEK_CUR – текущая позиция

SEEK_END – конец файла

Ввод и вывод в Си

особые ситуации:

```
int eof ( int fd );
```

Функция возвращает значение 1 при достижении конца файла, значение 0, если конец файла ещё не достигнут, или значение -1 при возникновении ошибки в/в.

```
void perror ( char *string );
```

Функция выводит строку string в поток stderr, затем выводит системное сообщение об обнаруженной ошибке в/в. Данная функция должна вызываться сразу за той функций ввода или вывода, которая привела к ошибке в/в.

Пример (копирование ввода на вывод без промежуточной буферизации):

```
#include <io.h>
#define BUFSIZE 256      // размер максимального ввода
void main ( void ) {     // замечание: драйверы в/в могут использовать
char buf [ BUFSIZE ];    // собственную буферизацию
int n;
while ( ( n = read ( 0, buf, BUFSIZE ) ) > 0 )    // пытаемся читать данные
      write ( 1, buf, n );                        // выводим прочитанное
}
```

Ввод и вывод в Си

Примеры реализаций функций ввода/вывода

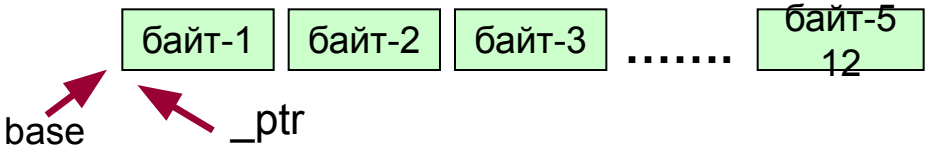
В заголовочном файле <stdio.h> описана структура следующего вида:

```
typedef struct  {
     char *_ptr; // указатель на текущий байт в буфере ввода/вывода
     int _cnt;     // счетчик байтов в буфере ввода/вывода
     char *base;// адрес буфера ввода/вывода
     ..........
}  FILE;
```

когда открывается некоторый файл (поток):

```
FILE* stream = fopen ( char *filename, char *mode);
```

система выделяет буфер размером 512 байтов:



поля переменной stream типа FILE заполняются надлежащим образом и, при операциях ввода, функция _filbuf (FILE *stream) заполняет этот буфер первой порцией данных.
В зависимости от опций компилятора, функции getc, putc, getchar, putchar реализуются как макросы или как функции. Посмотрим их реализацию в форме макросов.

Ввод и вывод в Си

Реализация getc, getchar, putc, putchar в форме макросов:

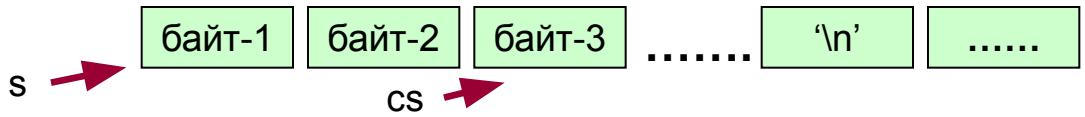
```
#define getc ( stream ) \
( --(stream) -> _cht >= 0 ) ? 0xff & *(stream) ->_ptr++ : _filbuf ( stream )

#define getchar ()  getc ( stdin )

#define putc ( c, stream ) \
( --(stream) -> _cht >= 0 ) ? 0xff & *(stream) ->_ptr++ = (char) c : _flsbuf ( ( c ), (stream) )

#define putchar ( c )  putc ( ( c ),  stdout )
```

Более подробно эти макросы можно посмотреть в заголовочном файле <stdio.h>.

Если код генерируется для dll или как реентерабельный, тогда применяется реализация getc и putc через вызов функций, и потоки блокируются на время выполнения операций ввода/вывода.

Ввод и вывод в Си

Реализация функций fgets и fputs:




```
char* fgets ( char *s, int n, FILE *stream ) {
register int c;
register char *cs = s;                  // обзаведемся быстрым указателем
while ( --n > 0 && ( c = getc ( stream ) ) != EOF )    // читаем из потока
        if ( ( *cs++ = c ) == ‘\n’ ) break;        // пишем в буфер
        *cs = ‘\0’;                    // оформляем признак конца строки
        return ( c == EOF && cs == s ) ? NULL : s;      // всё прочитано успешно?
}
int fputs ( char *s, FILE *stream ) {
register int c;
while ( c = *s++ )                      // берем символы из строки
        if ( putc ( c, stream ) == EOF ) return EOF;     // выводим их в поток
        putc ( ‘\n’, stream );          // оформляем признак перевода строки
        return 0;
}
```

Более подробно реализацию этих функций можно можно посмотреть в папке VS\src\....

Ввод и вывод в Си

Работа с файловой системой

Прототипы функций описаны в заголовочных файлах <io.h> и <direct.h>.

получить имя текущей папки (директории) можно так:

```
char* _getcwd ( char *buf, int maxlen );
```

Полный путь для текущей рабочей директории будет записан в buf. Если вместо buf был указан NULL, функция самостоятельно получит буфер длиной maxlen символов. Возвращает указатель на буфер с именем текущей рабочей директории или NULL, если произошла ошибка в/в.

изменить текущую папку (директорию):

```
int _chdir ( char *pathname );
```

Возвращает 0 в случае успеха или -1 при возникновении ошибки в/в.

создать папку (директорию):

```
int _mkdir ( char *pathname );
```

Возвращает 0 в случае успеха или -1 при возникновении ошибки в/в.

удалить папку (директорию):

```
int _rmdir ( char *pathname );
```

Возвращает 0 в случае успеха или -1 при возникновении ошибки в/в.

Ввод и вывод в Си

Для выборки файлов из текущей рабочей папки (директории) применяется следующая структура (уточнять её для конкретных IDE):

```
struct _finddata_t {
        unsigned attrib;       // атрибуты выбираемых файлов
        long time_create;      // дата и время создания файла
        long time_access;      // дата и время последнего доступа к файлу
        long time_write;       // дата и время последней записи в файл
        unsigned long size;    // длина файла в байтах
        char name[260];        // полный путь к файлу, включая его имя
};
```

Значения поля attrib могут быть комбинацией следующих констант:

```
        _A_NORMAL    // обычный файл
        _A_RDONLY    // файл, доступный только для чтения
        _A_HIDDEN    // скрытый файл
        _A_SYSTEM    // системный файл
        _A_SUBDIR    // подпапка (поддиректория)
        _A_ARCH      // архивный файл
```

Ввод и вывод в Си

выборка первого файла из текущей рабочей папки:

```
long _findfirst ( char *filespec, struct _finddata_t *fileinfo );
```

В аргументе filespec (спецификация имени файла) задается шаблон имени выбираемого файла, в шаблоне имени могут использоваться метасимволы ‘*’ и ‘?’.

Возвращает ненулевой дескриптор поиска (handle) в случае успеха (т.е. был найден файл, который удовлетворяет атрибутам attrib, заданным в _finddata_t) или -1 при возникновении ошибки в/в.

выборка следующего (очередного) файла из текущей рабочей папки:

```
int _findnext ( long handle, struct _finddata_t *fileinfo );
```

В аргументе handle задается ненулевой дескриптор поиска, который ранее был получен как результат работы функции _findfirst.

Возвращает 0 в случае успеха (т.е. был найден очередной файл, который удовлетворяет атрибутам attrib, заданным в _finddata_t) или -1 при возникновении ошибки в/в.

Ввод и вывод в Си

Пример вывода списка файлов текущей папки (шаблон “*.*”):

```
#include <io.h>
#include <stdio.h>
void main ( void ) {
struct _finddata_32_t finddata;        // структура для выборки файлов
long handle;  int flag = 0;
finddata.attrib = _A_NORMAL;        // выбираем только обычные файлы
handle = _findfirst ( “*.*”, &finddata);        // поищем первый файл
if ( handle )                    // если файлы есть, выведем
        while ( ! flag )  {                // их имена и размеры
                printf ( “%s %lu\n”, finddata.name, finddata.size );
                flag = _findnext ( handle, &finddata );        // ищем следующий файл
        }
else  printf ( “%s\n”, “Файлов не найдено” );    // нужных файлов не найдено
}
```

Перед практическим использованием данного примера уточните имена структур и функций (finddata, findfirst, findnext) для конкретного релиза Visual Studio.