ЧАСТЬ II

СТАНДАРТНАЯ БИБЛИОТЕКА ЯЗЫКА СИ

В языке Си стандартная библиотека более сильно интегрирована с языком по сравнению с другими языками программирования высокого уровня. Без использования функций стандартной библиотеки не может быть написана ни одна серьезная программа на языке Си, в частности потому, что в самом языке Си нет никаких средств ввода/вывода информации.

Стандартную библиотеку функций языка Си можно разделить на две категории: функции, которые имеются в библиотеке любой системы программирования языка Си для различных операционных систем и различных архитектур компьютеров, и функции, которые являются уникальными в рамках какой-либо системы программирования, или обеспечивают доступ к специфическим возможностям конкретной операционной системы, или связаны с конкретной архитектурой компьютера.

Функции первой категории образуют переносимое ядро библиотеки; программы, в которых используются только такие библиотечные функции, могут быть перенесены в другую систему программирования, другую операционную систему и/или на другой тип архитектуры компьютеров с наименьшими затратами. Плата за универсальность и переносимость — невозможность воспользоваться специфическими средствами, предоставляемыми конкретной вычислительной средой.

Функции второй категории предоставляют возможность получить доступ к функциям ядра данной операционной системы, к внутренним структурам данных операционной системы, к регистрам используемых аппаратных устройств. Кроме того, ко второй категории относятся функции, которые добавлены в библиотеку, исходя из вкусовых привязанностей разработчиков конкретной системы программирования — как им видится удобный набор средств для разработки различных алгоритмов (сравните, например, функции setmem и memset). В современных системах программирования Си в рамках общей тенденции к стандартизации такие необоснованные расширения библиотек сокращаются, но в ранних системах программирования разнобой был крайне высок.

К сожалению, наборы функций второй категории не согласованы даже для различных систем программирования в рамках одной операционной системы на одном типе архитектур компьютеров. Четко прослеживается это и на примере систем программирования TC и MSC. Библиотечные функции, обеспечивающие интерфейс для вызова одной и той же функции операционной системы, могут иметь не только разные параметры, но и разные названия.

Эти несогласованности объясняются, с одной стороны, коммерческими соображениями — стремлением удержать под контролем рынок программного обеспечения, чтобы пользователи, начавшие программировать с использованием одной системы программирования, покупали затем более новые программные продукты той же фирмы, а с другой стороны, поздним появлением стандарта на язык и на его библиотеку и независимости эволюции от версии к версии каждой системы программирования. При этом, надо отметить, происходит постепенное сближение различных систем программирования по мере того, как каждая из них заимствует наиболее ценные идеи у конкурентов. Так, различия между библиотеками более поздних версий систем программирования MSC и TC отчасти сокращены по сравнению с первыми версиями.

Данную часть книги следует рассматривать в первую очередь как справочник по стандартной библиотеке языка Си двух систем программирования — MSC и TC — для компьютеров типа IBM PC. Она также будет полезна для разработчиков новых систем программирования Си, поскольку в ней проводится сравнение реализации различных библиотечных функций двух широко распространенных систем программирования.

Из-за ограничений по объему в книгу не вошло описание специальных графических библиотек Си систем программирования MSC и TC.

Структура описания библиотеки такова: сначала дается краткое описание различных групп функций и вводятся основные понятия, используемые далее при описании библиотечных функций. Затем приводится полное описание всех функций в алфавитном порядке.

Предпринята попытка дать детальное описание библиотечных функций, приводится описание используемых констант, как они описываются (с помощью директивы препроцессора #define) во включаемых файлах.

Авторы считают, что для эффективной разработки качественного программного обеспечения на языке Си программист должен представлять нижний уровень реализации языка и его стандартной библиотеки. Поэтому иногда дается описание деталей (в частности, связанных с использованием системных вызовов ОС и имен типов, введенных с помощью конструкции typedef), которые часто сознательно замалчиваются в документации по библиотекам.

По нашему мнению, пользователь должен обладать полной информацией, осознавая при этом, с какими проблемами он столкнется при переносе программ в новую версию системы программирования, в новую операционную систему или на компьютер другой архитектуры.

Обозначение ANSI, используемое в полном описании библиотеки, указывает, что отмеченная библиотечная функция включена в стандарт языка Си.

9 КРАТКОЕ ОПИСАНИЕ БИБЛИОТЕКИ

Ниже приводится краткое описание основных групп функций для быстрой ориентации в библиотеке. При этом вводятся основные понятия, используемые при описании библиотечных функций (в частности, связанные с организацией ввода/вывода). Также указывается, в каком стандартном включаемом (по директиве препроцессора #include) файле содержится описание прототипа функции, относящихся к ней структур данных и констант.

9.1 Работа с областями памяти и строками

В стандартной библиотеке есть специальная группа Функций для обработки областей памяти, которые рассматриваются как последовательности байтов.

Если размер области, с которой необходимо работать, задается явно, будем называть такую область буфером.

Другое используемое понятие — строка. Отличие строки от буфера в том, что ее размер задается не явно, а определяется первым встретившимся при просмотре строки слева направо нулевым байтом (имеющим значение '\0'), причем считается, что этот нулевой байт также принадлежит строке.

Для копирования буферов, для присваивания каждому байту в пределах указанного буфера заданного значения и для сравнения содержимого двух буферов предназначены следующие функции:

Функция	*Краткое описание*
memccpy	копирует символы из одного буфера в другой до тех пор, пока не будет скопирован заданный символ или не будет скопировано определенное число символов
memchr	возвращает указатель на первое вхождение заданного символа в буфере
memcmp	сравнивает указанное число символов из двух буферов
memlcmp	сравнивает указанное число символов двух буферов, считая строчные и прописные буквы эквивалентными
memcpy	копирует указанное количество символов из одного буфера в другой
memset	инициализирует заданным значением указанное количество байтов в буфере
movedata	копирует определенное количество символов из одного буфера в другой, даже когда буфера находятся в разных сегментах

Прототипы перечисленных функций содержатся в файле memory.h (MSC) и в файлах mem.h и string.h (TC).

Система программирования TC предоставляет дополнительно следующие функции для работы с буферами:

Функция	*Краткое описание*
memove	копирует указанное количество символов из одного буфера в другой
movmem	копирует указанное количество символов из одного буфера в другой
setmem	инициализирует заданным значением указанное количество байтов в буфере

Прототипы функций memmove и movmem содержатся в файлах mem.h и string.h. Прототип функции setmem содержится в файле mem.h.

Для работы со строками существуют следующие библиотечные функции (TC & MSC):

Функция	*Краткое описание*
strcat	катенация (склеивание) строк
strchr	найти первое вхождение заданного символа в строке
strcmp	сравнить две строки
strcpy	копировать одну строку в другую
strcspn	найти первое вхождение символа из заданного набора символов в строке
strdup	дублирование строки
strerror	сформировать в строке сообщение об ошибке, состоящее из двух частей: системной диагностики и необязательного добавочного пользовательского сообщения
stricmp	сравнить две строки, считая символы нижнего и верхнего регистров эквивалентными
strlen	вычислить длину строки

strlwr	преобразовать строку в нижний регистр (строчные буквы)
strncat	добавить n символов в строку
strncmp	сравнение n символов в двух строках
strncpy	скопировать n символов из одной строки в другую
strnicmp	сравнение n символов двух строк
strnset	установить n символов в строке в заданное значение
strpbrk	найти первое вхождение любого символа из заданного набора в строке
strrchr	найти последнее вхождение заданного символа в строке
strrev	инвертировать (перевернуть) строку
strset	установить все символы строки в заданное значение
strspn	найти первую подстроку из заданного набора символов в строке
strstr	найти первую подстановку одной строки (более короткой) в другой
strtok	найти следующую точку в строке
strupr	преобразовать строку в верхний регистр (заглавные буквы)

Кроме того, система программирования MSC предоставляет дополнительно функцию strcmpi (идентична функции stricmp), а система программирования TC предоставляет функцию stpcpy (идентична функции strcpy, но возвращает в точку вызова другое значение).

Прототипы всех функций работы со строками содержатся в файле string.h. Все функции работают со строками, завершающимися нулевым байтом ('\0'). Для работы с массивом символов, не имеющим в конце нулевого байта, вы можете использовать функции преобразования буферов, описанные выше.

9.2 Определение класса символов и преобразование символов

Функция	*Краткое описание*
isalnum	проверка на букву или цифру
isalpha	проверка на букву
isascii	проверка на символ из набора кодировки ASCII
iscntrl	проверка на управляющий символ
isdigit	проверка на десятичную цифру
isgraph	проверка на печатный символ, исключая пробел
islower	проверка на малую букву
isprint	проверка на печатный символ
ispunct	проверка на знак пунктуации
isspace	проверка на пробельный символ
isupper	проверка на заглавную букву
isxdigit	проверка на шестнадцатеричную цифру
toascii	преобразование символа в код ASCII
tolower	проверка и преобразование в малую букву, если заглавная буква
toupper	проверка и преобразование малой буквы в заглавную
_tolower	преобразование буквы в малую (без проверки)
_toupper	преобразование буквы в заглавную (без проверки)

Все эти функции реализованы как макроопределения, заданные в файле ctype.h

9.3 Форматные преобразования данных

Функция	*Краткое описание*
atof	преобразование строки, в представляемое ей число типа float
atoi	преобразование строки в число типа int (целое)
atol	преобразование строки в число типа long (длинное целое)
ecvt	преобразование числа типа double в строку
fcvt	преобразование числа типа double в строку
gcvt	преобразование числа типа double в строку
itoa	преобразование числа типа int в строку
ltoa	преобразование числа типа long в строку
ultoa	преобразование числа типа unsigned long в строку

Система программирования TC предоставляет также следующие функции:

Функция	*Краткое описание*
strtod	преобразование строки в число типа double (покрывает возможности atof)
strtol	преобразование строки в число типа long (покрывает возможности atol)
strtoul	преобразование строки в число типа unsigned long

Прототипы всех перечисленных функций содержатся в файле stdlib.h. Прототип функции atof содержится, кроме того, в файле math.h.

9.4 Работа с каталогами файловой системы

Функция	*Краткое описание*
chdir	изменение текущего рабочего каталога
getcwd	получить имя текущего рабочего каталога
mkdir	создать новый каталог
rmdir	удаление каталога

Система программирования TC предоставляет, кроме перечисленных, следующие функции:

Функция	*Краткое описание*
findfirst	начало поиска файла по шаблону имени
findnext	продолжение поиска файла по шаблону имени
fnmerge	создание имени файла из отдельных компонент
fnsplit	разбиение имени файла на отдельные компоненты
getcurdir	узнать текущий каталог
getdisk	узнать текущее устройство
searchpath	поиск файла в различных каталогах
setdisk	задать текущее устройство

В системе программирования MSC прототипы функций содержатся в файле direct.h, в системе программирования TC прототипы функций содержатся в файле dir.h.

9.5 Операции над файлами

Функция	*Краткое описание*
access	определение прав доступа к файлу
chmod	изменение прав доступа к файлу
filelength	измерение длины файла
isatty	проверка, является ли устройство символьным
mktemp	генерация уникального имени файла
remove	уничтожение файла
rename	переименование файла
setmode	установить новые значения для параметров файла

Система программирования MSC предоставляет дополнительно следующие функции:

Функция	*Краткое описание*
chsize	изменение размера файла
fstat	получение информации о файле
locking	запирает область в файле (работает с версией ОС MS-DOS 3.0 и выше), временно запрещая к ней доступ со стороны других процессов, или отпирает эту область
slat	получение информации о файле
umask	установка маски для выбора режима работы по умолчанию
unlink	удаление файла

Система программирования TC предоставляет дополнительно функции:

Функция	*Краткое описание*
chmod	изменение прав доступа к файлу
lock	запирает область файла для доступа со стороны других процессов (работает с версией MS-DOS 3.0 и выше)
unlock	отпирает область (работает с версией MS-DOS 3.0 и выше)

Прототипы функций, за исключением функций **fstat** и **stat**, содержатся в файле **io.h**; прототипы функций **fstat** и **stat** описаны в файле **sys\stat.h**.

Функции **access**, **chmod**, **rename**, **stat** и **unlink** оперируют с файлами, которые задаются через имя пути (**pathname**) -или через имя файла.

Функции **chsize**, **filelength**, **isatty**, **locking**, **setmode**, **fstat** работают с уже открытыми файлами, которые определяются дескрипторами (**handle**) (смотри далее описание функций ввода/вывода нижнего уровня).

9.6 Ввод и вывод

Функции ввода и вывода в стандартной библиотеке Си позволяют читать данные из файлов или получать их с устройств ввода (например, с клавиатуры) и записывать данные в файлы, или выводить их на различные устройства (например, на принтер).

Функции ввода/вывода делятся на три класса:

1) Ввод/вывод верхнего уровня (с использованием понятия "поток").
2) Ввод/вывод для консольного терминала путем непосредственного обращения к нему.
3) Ввод/вывод нижнего уровня (с использованием понятия "дескриптор").

В библиотеке есть также функции для работы с последовательным портом (COM), они отнесены условно ко второй группе.

функции ввода/вывода верхнего уровня обеспечивают буферизацию работы с файлами. Это означает, что, когда производится чтение информации из файла или запись информации в файл, обмен информацией осуществляется не между программой и указанным файлом, а между программой и промежуточным буфером, расположенным в оперативной памяти.

Если производится операция записи в файл, то информация из буфера записывается в файл при заполнении буфера или при закрытии файла (или при выполнении каких-то других условий, смотри ниже). Если информация считывается из файла, то она на самом деле берется из буфера, а в буфер информация считывается из файла при открытии файла и впоследствии каждый раз при исчерпании (опустошении) буфера. Буферизация ввода/вывода выполняется автоматически, она позволяет ускорить выполнение программы за счет уменьшения количества обращений к сравнительно медленно работающим внешним устройствам.

Для пользователя файл, открытый на верхнем уровне, представляется как последовательность считываемых или записываемых байтов. Чтобы отразить эту особенность организации ввода/вывода, предложено понятие "поток" (соответствует английскому слову **stream**). Когда файл открывается, с ним связывается поток, выводимая информация записывается "в поток", считываемая информация берется "из потока".

Когда поток открывается для ввода/вывода, он связывается со структурой типа FILE (имя типа FILE определяется с помощью конструкции **typedef** в файле **stdio.h**). Структура содержит разнообразную информацию о файле. При открытии файла с помощью функции **fopen** возвращается указатель на структуру типа FILE. Этот указатель (указатель потока) используется для последующих операций с файлом, пользователь не обязан вникать в способ организации потока, он только должен сохранить полученный указатель и передавать его значение всем библиотечным функциям, используемым для ввода/вывода через этот поток.

Функции в/в верхнего уровня дают возможность для буферизованного форматированного и неформатированного ввода/вывода.

Функции в/в верхнего уровня относятся к числу функций, одинаково реализуемых в различных ОС и на разных компьютерах, с их помощью пользователь имеет возможность писать переносимые программы.

Функции ввода/вывода для консоли и порта распространяют возможности функций ввода/вывода верхнего уровня на этот класс устройств, добавляя новые возможности.

Они позволяют читать или записывать на консоль (терминал) или в порт ввода/вывода (например, порт принтера). Функции в/в с портом читают или записывают данные побайтно. Некоторые дополнительные режимы устанавливаются для в/в с консоли (например: ввод с эхо-печатью символов и без эхо-печати).

Функции в/в для консоли и порта являются уникальными для компьютеров типа IBM/PC.

Функции в/в низкого уровня не выполняют буферизацию и форматирование данных; они позволяют непосредственно пользоваться средствами ввода/вывода операционной системы.

При низкоуровневом открытии файла (при помощи функции **open**) с ним связывается дескриптор (**handle**). Дескриптор является целым значением, характеризующим размещение информации об открытом файле во внутренних таблицах системы. Дескриптор используется при последующих операциях с файлом.

Функции в/в нижнего уровня из стандартной библиотеки целесообразно использовать при разработке своей собственной подсистемы ввода/вывода.

Функции в/в нижнего уровня переносимы в рамках некоторых систем программирования Си, в частности относящихся к ОС UNIX.

9.6.1 Функции вода/вывода высокого уровня

Функция	*Краткое описание*
clearerr	очистка флажка ошибки для потока
fclose	закрытие потока
fcloseall	закрытие всех открытых (на верхнем уровне) файлов

fdopen	создание потока для файла, ранее открытого на нижнем уровне, используя дескриптор
feof	проверка на конец потока
ferror	проверка флажка ошибок потока
flush	сброс буфера потока на связанное с ним внешнее устройство
fgetc	чтение символа из потока
fileno	получение дескриптора файла, связанного с потоком
fgetchar	чтение символа из стандартного потока ввода stdin
fgets	чтение строки из потока
flushall	сброс буферов всех потоков
fopen	открытие потока (открыть файл и связать его с потоком)
fprint	запись данных в поток по формату
fputc	запись символа в поток
fputchar	запись символа в стандартный поток вывода stdout
fputs	запись строки в поток
fread	неформатированное чтение данных из потока
freopen	повторное открытие потока в новом режиме
fscanf	чтение из потока по формату
fseek	перемещение указателя файла в заданную позицию
ftell	получение текущей позиции указателя файла
fwrite	неформатированная запись данных в поток
getc	чтение символа из потока (реализуется через макроопределение)
getchar	чтение символа из потока stdin (версия макро)
gets	чтение строки из потока stdin
getw	чтение двух байтов (по размеру int) в формате слова из потока
printf	запись данных в поток stdout по формату
putc	запись символа в поток (версия макро)
putchar	запись символа в поток stdout (версия макро)
puts	запись строки в поток
putw	запись двух байтов (по размеру int) в формате слова в поток
rewind	установка указателя по файлу на начало файла
scanf	чтение данных из потока stdin по формату
setbuf	управление буферизацией потока
setvbuf	управление буферизацией потока и размером буфера
sprintf	запись данных в строку по формату
sscanf	чтение данных из строки по формату
tempnam	сгенерировать имя временного файла в заданном каталоге
tmpfile	создать временный файл
ungetc	вернуть символ в поток
vfprintf	запись данных в поток по формату
vsprintf	запись данных в строку по формату

Система программирования MSC дополнительно предоставляет следующие функции:

Функция	*Краткое описание*
rmtemp	удаление временных файлов, созданных посредством функции tmpfile
tmpnam	сгенерировать имя временного файла
vprintf	запись данных в поток stdout по формату

Система программирования TC дополнительно предоставляет следующие функции:

Функция	*Краткое описание*
vfscanf vscanf vsscanf	эти функции подобны функциям fscanf, scanf и sscanf, но принимают как параметр указатель на список аргументов — адресов переменных, которым присваиваются вводимые значения

Прототипы всех функций ввода/вывода верхнего уровня содержатся в файле stdio.h.
Некоторые константы, определенные в stdio.h, могут быть полезны в программе:

константа EOF	код, возвращаемый как признак конца файла
константа NULL	значение указателя, который не содержит адрес никакого реально размещенного в оперативной памяти объекта
константа BUFSIZ	определяет размер буфера потока в байтах
имя типа FILE	структура, которая содержит информацию о потоке

9.6.1.1 Высокоуровневое открытие файлов

Функции открытия потока возвращают указатель на тип FILE (этот указатель называют также указателем потока), этот указатель используется при дальнейших обращениях к потоку.

9.6.1.2 Стандартные потоки: stdin, stdout, stdeir, stdaux, stdprn.

Когда программа начинает выполняться, автоматически открываются пять потоков. Эти потоки — стандартный ввод (**stdin**), стандартный вывод (**stdout**), стандартный вывод для сообщений об ошибках (**stderr**), стандартный последовательный порт (**stdaux**) и стандартное устройство печати (**stdprn**).

По умолчанию стандартный ввод/вывод и стандартный вывод сообщений об ошибках связывается с консольным терминалом.

Назначения по умолчанию для стандартного порта и стандартного устройства печати зависят от конфигурации аппаратуры компьютера; эти потоки обычно связываются с последовательным портом и принтером, но могут быть и не установлены в отдельных системах.

Следующие указатели на структуру типа. FILE определяются в файле **stdio.h** и могут использоваться в любом месте как указатели потоков:

extern FILE * stdin; — стандартный ввод
extern FILE * stdout;— стандартный вывод
extern FILE * stderr;— стандартный вывод сообщений об ошибках
extern FILE * sidaux;— стандартный порт
extern FILE * stdprn;— стандартное устройство печати

При запуске оттранслированной программы на выполнение можно использовать символы перенаправления в/в из командного языка MS-DOS (< , > или >>) для переопределения стандартного ввода и вывода программы.

Можно переопределить **stdin**, **stdout**, **stderr**, **stdaux** или **stdprn** так, что они будут относиться к файлу на диске или устройству. Такие возможности предоставляет функция **freopen**.

9.6.1.3 Управление буферизацией потоков

Открытые файлы, для которых осуществляется высокоуровневый ввод/вывод, буферизуются по умолчанию, за исключением потоков **stdin**, **stdout**, **stderr**, **stdaux**, **stdprn**.

Потоки **stderr** и **stdaux** — не буферизованы. Если к ним применяется функция **printf** или **scanf**, создается временный буфер. Для обоих потоков может задаваться буферизация с помощью функций **setbuf** или **setvbuf**.

Буферизация для потоков **stdin**, **stdout**, **stdprn** выполняется следующим образом: буфер сбрасывается при его заполнении или когда вызванная библиотечная функция ввода/вывода завершает работу.

Использованием функции **setbuf** или **setvbuf** можно сделать поток небуферизованным или связать буфер с небуферизованным до этого потоком. Буфера, размещенные в системе, недоступны пользователю, кроме буферов, полученных с помощью **setbuf** или **setvbuf**.

Буфера должны иметь постоянный размер, равный константе BUFSIZ в **stdio.h**.

Если используется **setvbuf**, размер буфера устанавливает пользователь. Буфера автоматически сбрасываются при их наполнении, или когда связанный с буфером файл закрывается, или когда происходит нормальное завершение программы.

Можно сбросить буфера в произвольный момент времени, используя функции **fflush** и **flushall**. Функция **fflush** сбрасывает буфер одного заданного потока, а функция **flushall** сбрасывает буфера всех потоков, которые открыты и буферизованы в данный момент.

9.6.1.4 Закрытие потоков

Функции **fclose** и **fcloseall** закрывают поток или потоки. Функция **fclose** закрывает один заданный поток, **fcloseall** — все потоки, кроме потоков **stdin**, **stdout**, **stderr**, **stdaux**, **stdprn**.

Если программа не выполняет закрытия потоков, потоки автоматически закрываются, когда программа завершается неаварийно. Однако следует закрывать потоки по завершении работы с ними, так как число потоков, которые могут быть открыты одновременно, ограничено.

9.6.1.5 Чтение и запись данных

Функции ввода/вывода верхнего уровня позволяют передавать данные различными способами.

Операции чтения и записи в потоках начинаются с текущей. позиции в потоке, идентифицируемой как "file pointer"

(указатель файла) для потока. Указатель файла изменяется после выполнения операции чтения или записи.

Например, если Вы читаете один символ из потока, указатель файла продвигается на 1 байт, так что следующая операция начнется с первого несчитанного символа. Если поток открыт для добавления, указатель файла автоматически позиционируется на конец файла перед каждой операцией записи.

Поток, связанный с таким устройством, как консольный терминал, не имеет указателя файла. Программы, которые перемещают указатель файла (с помощью функции **fseek**), будут иметь в этом случае неопределенный результат.

9.6.1.6Обнаружение ошибок

Когда происходит ошибка в операции с потоком, устанавливается в ненулевое значение флажок ошибки для потока. Можно использовать макроопределение **ferror**, чтобы определить, произошла ли ошибка.

После каждой ошибки флажок ошибки остается установленным до тех пор, пока не будет сброшен вызовом функции **clearerr** или **rewind**.

9.6.2 Функции вода/вывода нижнего уровня

Функция	*Краткое описание*
close	закрыть файл
creat	создать файл
dup	создать второй дескриптор (handle) для файла
dup2	переназначить дескриптор (handle) для файла
eof	проверка на конец файла
lseek	позиционирование указателя файла в заданное место
open	открыть файл
read	читать данные из файла
sopen	открыть файл в режиме разделения
tell	получить текущую позицию указателя файла
write	записать данные в файл

Система программирования TC предоставляет дополнительно следующие функции:

Функция	*Краткое описание*
_creat	создать файл
creatnew	создать новый файл
creattemp	создать новый файл
_open	открыть файл
_read	чтение данных из файла
_write	запись данных в файл

Нижний уровень ввода и вывода не работает с буферизованными или форматированными данными. Для работы с файлами, открытыми посредством функции нижнего уровня, используется дескриптор файла (**handle**).

Для открытия файлов используются функции **open** и **_open**; В ОС MS/DOC версии 3.0 или выше может быть использована функция **sopen** для открытия файлов с атрибутами режима разделения файла.

функции нижнего уровня, в отличие от функций верхнего уровня, не требуют включения файла **stdio.h**. Тем не менее нескольких общих констант, определенных в файле **stdio.h**, как, например, признак конца файла EOF, могут оказаться полезными. Если программа использует эти константы, необходимо включить файл **stdio.h**.

Прототипы функций нижнего уровня содержатся в файле **io.h**.

9.6.2.1Открытие файлов

Файл должен быть открыт функциями **open**, **sopen** или **creat** до выполнения первой операции ввода или вывода с использованием функций нижнего уровня для этого файла.

Файл может быть открыт для чтения, записи, или для чтения и записи, может быть открыт в текстовом или в двоичном режиме.

Файл **fcntl.h** должен быть включен при открытии файла, так как содержит определения для флагов, используемых в функции **open**. В некоторых случаях также должны быть включены файлы **sys\types.h** и **sys\stat.h**.

Перечисленные функции возвращают дескриптор файла, который используется при последующих операциях с файлом. При вызове одной из функций открытия файла необходимо

возвращаемое функцией значение присвоить целочисленной переменной и потом использовать значение этой переменной, чтобы обращаться к открытому файлу.

9.6.2.29.6.2.2. Переопределение дескрипторов (handle)

Когда программа начинает выполняться, пять дескрипторов (**handle**), соответствующих стандартным вводу, выводу, выводу сообщений об ошибках, порту и устройству печати, уже назначены. Пользователь может использовать значения этих дескрипторов при вызове функций ввода/вывода нижнего уровня.

Каждый из этих дескрипторов соответствует одному из стандартных потоков, значения этих дескрипторов таковы:

поток	*значение дескриптора*
stdin	0
stdout	1
stderr	2
stdaux	3
stdprn	4

Можно использовать эти дескрипторы файлов в программе без предварительного открытия этих файлов. Они автоматически открываются при запуске программы.

Так же, как с функциями для потоков, Вы можете использовать перенаправление, чтобы переопределить стандартный ввод и вывод.

Функции **dup** и **dup2** позволяют назначать несколько **handle** для одного файла; эти функции обычно используются, чтобы связать дополнительные дескрипторы с уже используемыми файлами.

9.6.2.3Чтение и запись данных

Функции **read** и **write**, как и функции ввода/вывода верхнего уровня, начинают выполнение очередной операции с текущей позиции в файле. Текущая позиция изменяется при каждой операции чтения или записи.

Функция **eof** может быть использована для проверки на конец файла.

Когда происходит ошибка, программы в/в нижнего уровня присваивают код ошибки переменной **errno**. Можно использовать функцию **perror** для печати информации об ошибках в/в. Можно позиционировать указатель файла на определенную позицию в файле, используя функцию **lseek**. Используя функцию **tell**, можно определить текущую позицию указателя файла. Устройства типа консольного терминала не имеют указателя файла. Результат функций **lseek** и **tell** не определен, если они применяются к дескриптору, связанному с таким устройством.

9.6.2.4Закрытие файлов

Функция **close** закрывает открытые файлы. Открытые файлы также автоматически закрываются при неаварийном завершении программы.

9.6.3 Функции вода/вывода с консольного терминала и порта

Функции ввода/вывода для консольного терминала выделены в отдельную группу, потому что они используют специфические особенности компьютера IBM/PC (наличие специального видеоадаптера) и не являются переносимыми на другие типы компьютеров.

функция	*Краткое описание*
cgets	чтение строки с консоли
cprintf	запись данных на консольный терминал по формату
cputs	вывод строки на консольный терминал
getch	чтение символа с консоли
getche	чтение символа с консоли с эхо-печатью
kbhit	проверка нажатия клавиши на консоли
putch	вывод символа на консольный терминал
ungetch	возврат последнего прочитанного символа с консольного символа обратно с тем, чтобы он стал следующим символом для чтения

Система программирования MSC предоставляет дополнительно функцию cscanf - чтение данных с консоли по формату.

Система программирования TC предоставляет дополнительно функцию getpass - ввод с терминала пароля без эхо-печати

Прототипы функций содержатся в файле **conio.h**. Устройства: консольный терминал и порт не могут быть открыты или закрыты перед выполнением в/в, поэтому функции **fopen** и **fclose** не вызываются. Функции в/в с консольного терминала позволяют читать и записывать строки (**cgets** и **cputs**), форматированные данные (**cscanf** и **cprintf**) и

символы. Функция **kbhit** определяет: было ли нажатие клавиши на консольном терминале. Эта функция позволяет определить наличие символов для ввода с клавиатуры до попытки чтения.

9.7 Математические функции

Функция	*Краткое описание*
abs	нахождение абсолютного значения выражения типа int
acos	вычисление арккосинуса
asin	вычисление арксинуса
atan	вычисление арктангенса x
atan2	вычисление арктангенса от y/x
cabs	нахождение абсолютного значения комплексного числа
ceil	нахождение наименьшего целого, большего или равного x
_clear87	получение значения и инициализация слова состояния сопроцессора и библиотеки арифметики с плавающей точкой
_control87	получение старого значения слова состояния для функций арифметики с плавающей точкой и установка нового состояния
cos	вычисление косинуса
cosh	вычисление гиперболического косинуса
exp	вычисление экспоненты
fabs	нахождение абсолютного значения типа double
floor	нахождение наибольшего целого, меньшего или равного x
fmod	нахождение остатка от деления x/y
_fpreset	повторная инициализация пакета плавающей арифметики
frexp	разложение x как произведения мантиссы на экспоненту 2^n
hypot	вычисление гипотенузы
labs	нахождение абсолютного значения типа long
ldexp	вычисление $x*2^{exp}$
log	вычисление натурального логарифма
log10	вычисление логарифма по основанию 10
matherr	управление реакцией на ошибки при выполнении функций математической библиотеки
modf	разложение x на дробную и целую часть
pow	вычисление x в степени y
sin	вычисление синуса
sinh	вычисление гиперболического синуса
sqrt	нахождение квадратного корня
_status87	получение значения слова состояния с плавающей точкой
tan	вычисление тангенса
tanh	вычисление гиперболического тангенса

Система программирования MSC предоставляет дополнительно функции:

Функция	*Краткое описание*
bessel	вычисление функции Бесселя
dieeetomsbin	преобразование плавающего числа двойной точности из IEEE-формата в Microsoft-формат
dmsbintoieee	преобразование плавающего числа двойной точности из Microsoft-формата в IEEE-формат
fieeetomsbin	преобразование числа с плавающей точкой из IEEE-формата в Microsoft-формат
fmsbintoieee	преобразование числа с плавающей точкой из Microsoft-формата в IEEE-формат

Система программирования TC предоставляет дополнительно функции:

Функция	*Краткое описание*
_matherr	управление реакцией на ошибки при выполнении функций из математической библиотеки
pow10	вычисление десятичной степени

Прототипы функций содержатся в файле **math.h**, за исключением прототипов функций **_clear87**, **_control87**, **_fpreset**, **status87**, которые определены в файле **float.h**. Функция **matherr** (ее пользователь может задать сам в своей программе) вызывается любой библиотечной математической функцией при возникновении ошибки. Эта программа определена в библиотеке, но может быть переопределена пользователем, если она необходима, для установки различных процедур обработки ошибок.

9.8 Динамическое распределение памяти

Библиотека языка Си предоставляет механизм распределения динамической памяти (**heap**). Этот механизм позволяет динамически (по мере возникновения необходимости) запрашивать из программы дополнительные области оперативной памяти.

Работа функций динамического распределения памяти различается для различных моделей памяти, поддерживаемых системой программирования (смотри первую части книги).

В малых моделях памяти (**tiny**, **small**, **medium**) доступно для использования все пространство между концом сегмента статических данных программы и вершиной программного стека, за исключением 256-байтной буферной зоны непосредственно около вершины стека.

В больших моделях памяти (**compact**, **large**, **huge**) все пространство между стеком программы и верхней границей физической памяти доступно для динамического размещения памяти.

Следующие функции используются для динамического распределения памяти:

Функция	*Краткое описание*
calloc	выделить память для массива
free	освободить блок, полученный посредством функции calloc, malloc или realloc
malloc	выделить блок памяти
realloc	переразместить ранее выделенный блок памяти, изменив его размер
sbrk	переустановить адрес первого байта оперативной памяти, недоступного программе (начала области памяти вне досягаемости программы)

Система программирования MSC предоставляет дополнительно функции:

Функция	*Краткое описание*
alloca	выделение блока памяти из программного стека
_expand	изменение размера блока памяти, не меняя местоположения блока
_ffree	освобождение блока, выделенного посредством функции fmalloc
_fmalloc	выделение блока памяти вне данного сегмента
_freect	определить примерное число областей заданного размера, которые можно выделить
_fmsize	возвращает размер блока памяти, на который указывает дальний (far) указатель
halloc	выделить память для большого массива (объемом более 64 Кбайтов)
hfree	освободить блок памяти, выделенный посредством функции halloc
_memavl	определить примерный размер в байтах памяти, доступной для выделения
_msize	определить размер блока, выделенного посредством функций calloc, malloc, realloc
_nfree	освобождает блок, выделенный посредством _nmalloc
_nmalloc	выделить блок памяти в заданном сегменте
_nmsize	определить размер блока, на которой указывает близкий (near) указатель
stackavail	определить объем памяти, доступной для выделения посредством функции alloca

Система программирования TC предоставляет дополнительно функции:

Функция	*Краткое описание*
brk	переустановить адрес первого байта оперативной памяти, недоступного программе (начала области памяти вне досягаемости программы)
allocmem	низкоуровневая функция выделения памяти
freemem	низкоуровневая функция возврата памяти операционной системе
coreleft	узнать, сколько осталось памяти для выделения в данном сегменте
farcalloc	выделить блок памяти вне данного сегмента
farcoreleft	определить, сколько памяти для размещения осталось вне данного сегмента
farmalloc	выделить блок памяти вне данного сегмента
farrealloc	изменить размер блока, ранее выделенного функцией farmalloc или farcalloc
farfree	освободить блок, ранее выделенный функцией farmalloc или farcalloc

Прототипы функций содержатся в файле **malloc.h** для системы программирования MSC и в файле **alloc.h** для системы программирования TC.

Функции **calloc** и **malloc** выделяют блоки памяти, функция **malloc** выделяет заданное число байтов, тогда как **calloc** выделяет и инициализирует нулями массив элементов заданного размера.

Функции **_fmalloc** и **_nmalloc** подобны **malloc**, за исключением того, что **_fmalloc** и **_nmalloc** позволяют выделить блок байтов в том случае, когда существуют ограничения

адресного пространства текущей модели памяти. Функция **halloc** выполняется аналогично **calloc**, но **halloc** выделяет память для большого массива (больше 64 К).

Функции **realloc** и **_expand** изменяют размер полученного блока.

Функция **free** (для **calloc**, **malloc** и **realloc**), функция **ffree** (для **_fmalloc**), функция **_nfree** (для **_nmalloc**) и функция **hfree** (для **halloc**) освобождают память, которая была выделена ранее, и делают ее доступной для последующего распределения.

Функции **_freect** и **_memavl** определяют: сколько памяти доступно для динамического выделения в заданном сегменте; **_freect** возвращает примерное число областей заданного размера, которые могут быть выделены; **_memavl** возвращает общее число байтов, доступных для выделения.

Функции **_msize** (для **calloc**, **malloc**, **realloc** и **_expand**), **_fmsize** (для **_fmalloc**) и **_nmsize** (для **_nmalloc**) возвращают размер ранее выделенного блока памяти.

Функция **sbrk** — это функция нижнего уровня для получения памяти. Вообще говоря, программа, которая использует функцию **sbrk**, не должна использовать другие функции выделения памяти, хотя их использование не запрещено.

Все выше описанные функции распределяли области памяти из общей памяти. Система программирования MSC предоставляет 2 функции, **alloca** и **stackavail**, для выделения памяти из стека и определения количества доступной памяти в стеке.

9.9 Использование системных вызовов операционной системы MS-DOS

Функция	*Краткое описание*
bdos	вызов системы MS-DOS; используются только регистры DX и AL
dosexterr	получение значений регистров из системы MS-DOS вызовом 59H
FP_OFF	возвращает смещение far-указателя
FP_SEG	возвращает сегмент far-указателя
int86	вызов прерывания MS-DOS
int86x	вызов прерывания MS-DOS
intdos	системный вызов MS-DOS
intdosx	системный вызов MS-DOS
segread	возвращает текущее значение сегментных регистров

Прототипы функций и макроопределения содержатся в файле **dos.h**.

Система программирования MSC предоставляет дополнительно функции:

Функция	*Краткое описание*
inp	чтение с указанного порта в/в
outp	вывод в указанный порт в/в

Прототипы функций **inp** и **outp** содержатся в файле **conio.h**.

Система программирования TC предоставляет дополнительно следующие функции:

Функция	*Краткое описание*
absread	чтение с диска по номеру сектора
abswrite	запись на диск по номеру сектора
bdosptr	вызов системы MS-DOS
country	определение способа записи времени в данной стране
ctrlbrk	установить реакцию на <CTRL/BREAK>
disable	отменить прерывания
enable	разрешить прерывания
freemem	освободить память
getinterrupt	возбудить прерывание
getcbrk	узнать установленную реакцию на <CTRL/BREAK>
getdfree	узнать объем свободного места на диске
getdta	узнать адрес области передачи данных диска
getfat	получить информацию из таблицы размещения файлов
getfatd	получить информацию из таблицы размещения файлов
getpsp	получить сегментный префикс для текущего программного адреса текущего выполняемого процесса
getvect	узнать значение вектора прерывания
getverify	узнать режим проверки записи на диск
harderr	регистрация функции обработки аппаратных ошибок
hardresume	возврат из функции обработки аппаратных ошибок
hardretn	возврат из функции обработки аппаратных ошибок
inport	ввести слово из порта
inportb	ввести байт из порта
intr	аналог функции int86
keep	зафиксировать программу в памяти

MK_FP	составить far-указатель из компонент
outport	вывести слово в порт
outportb	вывести байт в порт
parsfnm	выделение имени файла из командной строки MS-DOS
peek	получить значение слова по адресу
peekb	получить значение байта по адресу
poke	записать слово в память по адресу
pokeb	записать байт в память по адресу
randbrd	чтение с диска
randbwr	запись на диск
setdta	установить адрес области передачи данных диска
setvect	задать значение вектора прерывания
setverify	включить режим проверки записи на диск
sleep	задержка
unlink	удаление файла

Прототипы функций и макроопределения содержатся в файле **dos.h**.

Система программирования TC предоставляет также следующие функции для обращения к BIOS (базовой подсистеме ввода/вывода операционной системы):

Функция	*Краткое описание*
bioscom	управление последовательным каналом
biosdisk	управление диском
biosequip	узнать конфигурацию аппаратуры
bioskey	управление клавиатурой
biosmemory	узнать объем оперативной памяти
biosprint	управление устройством печати
biostime	управление BIOS-таймером

Прототипы функций обращения к BIOS содержатся в файле **bios.h**.

9.10 Управление процессами

Функция	*Краткое описание*
abort	завершить процесс
execl	выполнить порождаемый процесс со списком аргументов
execle	выполнить порождаемый процесс со списком аргументов и заданным окружением (контекстом имен командного языка операционной системы)
execlp	выполнить порождаемый процесс, используя переменную PATH и список аргументов
execlpe	выполнить порождаемый процесс, используя переменную PATH, заданное окружение и список аргументов
execv	выполнить порождаемый процесс с массивом аргументов
execve	выполнить порождаемый процесс с массивом аргументов и заданным окружением
execvp	выполнить порождаемый процесс, используя переменную PATH и массив аргументов
execvpe	выполнить порождаемый процесс, используя переменную PATH, заданное окружение и массив аргументов
exit	завершить процесс
_exit	завершить процесс без скидывания буферов
signal	управление сигналом прерывания
spawnl	выполнить порождаемый процесс со списком аргументов
spawnle	выполнить порождаемый процесс со списком аргументов и заданным окружением
spawnlp	выполнить порождаемый процесс, используя переменную PATH и список аргументов
spawnlpe	выполнить порождаемый процесс, используя переменную PATH, заданное окружение и список аргументов
spawnv	выполнить порождаемый процесс с массивом аргументов
spawnve	выполнить порождаемый процесс с массивом аргументов и задан ным окружением
spawnvp	выполнить порождаемый процесс, используя переменную PATH и массив аргументов
spawnvpe	выполнить порождаемый процесс, используя переменную PATH, заданное окружение и массив аргументов
system	выполнение команды MS-DOS

Система программирования MSC предоставляет дополнительно функции:

Функция	*Краткое описание*
getpid	получить номер процесса
onexit	выполнить функцию при завершении программы

Термин "процесс" относится к программе, которая выполняется под управлением операционной системы. Процесс состоит из кодов программы и данных, а также информации о состоянии процесса, такой, как число открытых файлов. Где бы ни выполнялась программа на уровне MS-DOS, запускается процесс. Можно запустить, остановить и управлять процессом из программы, используя функции управления процессом. Прототипы всех функций управления процессами объявлены в файле **process.h** (исключая функцию **signal**). Прототип функции **signal** содержится в файле **signal.h**. Функции управления процессом позволяют следующее:

1) Узнать уникальный номер процесса (**getpid**).
2) Завершить процесс (**abort**, **exit**, **_exit**).
3) Управлять сигналами прерывания (**signal**).
4) Начать новый процесс (разновидности **exec** и **spawn** функции, **system** функция).

Функции **abort** и **_exit** осуществляют немедленное завершение без скидывания буферов потоков, функция **exit** осуществляет выход после скидывания буферов потоков. Функция **system** вызывает на выполнение заданную команду MS-DOS. Функции **exec** и **spawn** создают новый процесс, называемый порождаемым процессом. Разница между функциями **exec** и **spawn** в том, что **spawn** способна возвращать управление из порождаемого процесса к его родителю. Оба, и родитель, и порождаемый процесс, размещаются в памяти (если не указан флаг P_OVERLAY).

В функции **exec** порождаемый процесс перекрывает порождающий процесс, так что возврат управления в родительский процесс невозможен (если не произошла ошибка во время попытки запуска на выполнение порождаемого процесса).

В таблице описывается способ формирования **exec** и **spawn**. Имя функции задается в первом поле. Второе поле определяет: используется ли переменная PATH для поиска файла для выполнения, который определяет порождаемый процесс.

Третье поле описывает метод передачи аргументов порождаемому процессу. Передача аргументов списком означает, что аргументы в порождаемый процесс передаются один за одним, в том порядке, как пользователь перечислил их в обращении к функции **exec** или **spawn**. Передача аргументов массивом означает, что аргументы помещаются в массив и указатель на массив передается порождаемому процессу. Передача списком обычно используется, когда число аргументов постоянно и известно заранее, а метод передачи аргументов массивом полезен, когда число аргументов должно быть определено во время работы. Последнее поле определяет: унаследует ли порождаемый процесс от родителя окружение, или оно будет изменено для него.

Таблица 9.1.

функция	*Использование PATH переменной*	*Способ передачи аргументов*	*Окружение*
execl spawnl	не использует PATH	список аргументов	наследует от родителя
execle spawnle	не использует PATH	список аргументов	указатель на таблицу окружения (последний аргумент)
execlp spawnlp	использует PATH	список аргументов	наследует от родителя
execlpe spawnlpe	использует PATH	список аргументов	указатель на таблицу окружения (последний аргумент)
execv spawnv	не использует PATH	массив аргументов	наследует от родителей
execve spawnve	не использует PATH	массив аргументов	указатель на таблицу окружения (последний аргумент)
execvp spawnvp	использует PATH	массив аргументов	наследует от родителя
execvpe spawnvpe	использует PATH	массив аргументов	указатель на таблицу окружения (последний аргумент)

9.11 Поиск и сортировка

Следующие библиотечные функции предназначены для поиска и сортировки в массиве:

Функция	*Краткое описание*

bsearch	выполняет двоичный поиск
lfind	выполняет линейный поиск для заданного значения
lsearch	выполняет линейный поиск для заданного значения, которое добавляется в массив, если не найдено
qsort	выполняет быструю сортировку

Прототипы функций содержатся в файле **search.h** в системе программирования MSC, в файле **stdlib.h** в системе программирования TC.

9.12 Функции работы со временем

Функция	*Краткое описание*
asctime	преобразование времени из структуры (внутренней формы) в символьную строку
ctime	преобразование времени из длинного целого (long int) в строку символов
gmtime	преобразование времени из целого (int) в структуру
localtime	преобразование времени из целочисленного (int) в структуру с локальной поправкой
tzset	установить переменную времени из переменной времени среды

Система программирования MSC предоставляет дополнительные функции:

Функция	*Краткое описание*
ftime	получить текущее время системы как структуру
time	получить текущее системное время как длинное целое (long int)
utime	установить время изменения файла

Система программирования TC предоставляет дополнительные функции:

Функция	*Краткое описание*
difftime	вычислить разность по времени
dostounix	преобразование времени из формате ОС MS-DOS в формат ОС UNIX
getdate	получить системную дату как структуру
getftime	получить системную дату
gettime	получить системное время как структуру
setdate	установить системную дату
setftime	установить системное время
settime	установить системное время
stime	установить системное время
unixtodos	преобразовать время из формата ОС UNIX в формат ОС MS-DOS

Функции **time** и **ftime** возвращают текущее время как число секунд, прошедших с 1 января 1970 Гринвичского Всемирного времени. Эта величина может быть преобразована, скорректирована и сохранена посредством функций **asctime**, **ctime**, **gmtime** и **localtime**.

Функция **utime** устанавливает время модификации для указанного файла, используя текущее время или значение времени, заданное в структуре.

Функция **ftime** требует включения двух файлов: **sys\types.h** и **sys\timeb.h**. Прототип функции **ftime** содержится в **sys\timeb.h**.

Функция **utime** также требует включения двух файлов: **sys\types.h** и **sys\utime.h**. Прототип функции **utime** содержится в файле **sys\utime.h**.

Прототипы функций **dostounix**, **getdate**, **gettime**, **setdate**, **settime**, **unixtodos** содержатся в файле **dos.h**.

Прототипы функций **getftime** и **setftime** определены в файле **io.h**.

Прототипы остальных функций работы со временем времени содержатся в файле **time.h**.

При использования функции **ftime** или **localtime**, чтобы сделать поправку для местного времени, необходимо определить переменную командного языка операционной системы TZ.

9.13 Функции работы со списком аргументов

Функция	*Краткое описание*
va_arg	выбрать аргумент из списка
va_end	переустановить указатель
va_start	установить указатель на начало списка аргументов

Эти макроопределения дают возможность получить доступ к аргументам функции, когда число аргументов переменно.

В системе программирования MSC для совместимости с ОС UNIX System V можно использовать включаемый файл **vararg.h**, для совместимости со стандартом ANSI на язык Си можно использовать включаемый **stdarg.h**. В этих файлах содержится две различных версии макроопределений.

В системе программирования TC доступна только версия **stdarg.h**.

9.14 Другие функции

Функция	*Краткое описание*
assert	проверка утверждения о состоянии переменных
getenv	получить значение переменной среды (окружения)
perror	напечатать сообщение об ошибке
putenv	изменить значение переменной среды
swab	поменять местами два смежных байта
rand	получить псевдо-случайное число
srand	инициализация датчика случайных чисел
setjmp	запоминание точки для многоуровневого возврата
longjmp	многоуровневый возврат из функции

Прототипы всех функций, исключая **assert**, **longjmp** и **setjmp**, описаны в **stdiib.h**.

Assert — это макроопределение из файла **assert.h**.

Прототипы функций **setjmp** и **longjmp** содержатся в файле **setjmp.h**.

Программы **getenv** и **putenv** предоставляют доступ к таблице среды процесса. Глобальная переменная **environ** также указывает на таблицу среды, но рекомендуется использование функций **getenv** и **putenv** для доступа и изменения установленной среды вместо обращения к таблице среды напрямую.

Функция **perror** печатает диагностическое сообщение о последней ошибке, произошедшей при вызове какой-либо библиотечной функции.

Функция **swab** обычно используется для преобразования данных в формат других компьютеров, где используется иной порядок следования байтов в слове в оперативной памяти.