[9. Переключение ввода-вывода.](#_Toc515770064)

[9.1. Ввод и вывод одного символа: функции getchar() и putchar().](#_Toc515770065)

[9.2. Буферы.](#_Toc515770066)

[9.3. Чтение данных.](#_Toc515770067)

[9.3.1. Чтение строки.](#_Toc515770068)

[9.3.2. Чтение файла.](#_Toc515770069)

[9.4. Переключение и работа с файлами.](#_Toc515770070)

[9.5. Операционная система UNIX.](#_Toc515770071)

[9.5.1. Переключение вывода.](#_Toc515770072)

[9.5.2. Переключение ввода.](#_Toc515770073)

[9.5.3. Комбинированное переключение.](#_Toc515770074)

[9.5.4. Конвейерная пересылка](#_Toc515770075)

[9.5.5. Операционные системы, отличные от ОС UNIX.](#_Toc515770076)

[9.6. Системно-зависимые средства: порты ввода-вывода микропроцессоров INTEL 8086/8088.](#_Toc515770077)

[9.6.1. Использование порта.](#_Toc515770078)

1. Переключение ввода-вывода.

В вычислительной технике слова «ввод» и «вывод» применяются в нескольких разных смыслах. Мы можем говорить об устройствах ввода и вывода, таких, как терминалы, накопители на магнитных дисках, точечно-матричные принтеры и т. п., или о данных, используемых при вводе и выводе, или же, наконец, о функциях, реализующих ввод и вывод. Основной целью данной темы является обсуждение функций, применяемых при вводе и выводе, но, кроме этого, мы коснемся и двух других аспектов этого понятия.

Под функциями ввода-вывода подразумеваются функции, которые выполняют транспортировку данных в программу и из нее. Мы уже использовали две такие функции: printf() и scanf(). Теперь же рассмотрим несколько других возможностей, предоставляемых языком С.

Функции ввода-вывода не входят в определение языка С; их разработка возложена на программистов, реализующих компилятор с языка С. Если вы являетесь проектировщиком такого компилятора, то можете реализовать любые функции ввода-вывода. Если вычислительная система, для которой вы его создаете, обладает той или иной особенностью, например тем, что каналы ввода-вывода построены на основе портов микропроцессора INTEL 8086, вы можете встроить в нее специальные функции ввода-вывода, ориентированные на эту особенность. Мы рассмотрим пример применения такого подхода в конце. С другой стороны, выгода использования стандартного набора функций ввода-вывода на всех системах очевидна. Это дает возможность писать «переносимые» программы, которые легко можно применять на разных машинах. В языке С имеется много функций ввода-вывода такого типа, например printf() и scanf(). Ниже мы рассмотрим функции getchar() и putchar().

Эти две функции осуществляют ввод и вывод одного символа при каждом обращении к ним. На первый взгляд, выполнение операций подобным образом может показаться довольно странным, так как, учитывая все сказанное выше, мы уже можем с легкостью осуществить ввод нескольких символов подряд. Но этот способ ввода данных лучше соответствует возможностям машины. Более того, такой подход служит основой построения большинства программ обработки текстов, являющихся последовательностями обычных слов. Мы увидим, как можно применять эти функции в программах, занимающихся подсчетом символов, чтением и копированием файлов. Попутно мы узнаем про буферы, эхо-печать и переключение ввода-вывода.

## Ввод и вывод одного символа: функции getchar() и putchar().

Функция getchar() получает один символ, поступающий с пульта терминала (и поэтому имеющий название), и передает его выполняющейся в данный момент программе. Функция putchar() получает один символ, поступающий из программы, и пересылает его для вывода на экран. Ниже приводится пример очень простой программы. Единственное, что она делает, это принимает один символ с клавиатуры и выводит его на экран. Мы будем постепенно модифицировать данную программу до тех пор, пока она не приобретет ряд полезных возможностей. Из дальнейшего вы узнаете, что представляют из себя эти возможности, но сначала давайте посмотрим на наш скромный первый вариант

// ввод-вывод1

#include <stdio.h>

void main()

{

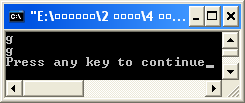
char ch;

ch = getchar(); // строка 1

putchar (ch); // строка 2

}

Для большинства систем спецификации функций getchar и putchar содержатся в системном файле <stdio.h>, и только по этой причине мы указали данный файл в программе. Использование такой программы приводит к следующему:



Обозначение [ввод] служит указанием, что вы должны нажать клавишу [ввод]. В любом случае, первый символ g вы набираете на клавиатуре сами, а второй выводится компьютером.

Результат зависит от того, есть в вашей системе «буферизованный» ввод или нет. Если перед тем как получить на экране ответ, вы должны нажать клавишу [ввод], то буферизация в вашей системе имеется. Давайте закончим рассмотрение функций getchar() и putchar() перед тем, как приступить к обсуждению понятия буферов.

Функция getchar() аргументов не имеет (т. е. при ее вызове в круглые скобки не помещается никакая величина). Она просто получает очередной поступающий символ и сама возвращает его значение выполняемой программе. Например, если указанная функция получает букву Q, ее значением в данный момент будет эта буква. Оператор, приведенный в строке 1, присваивает значение функции getchar() переменной ch.

Функция putchar() имеет один аргумент. При ее вызове необходимо в скобках указать символ, который требуется вывести на печать. Аргументом может быть одиночный символ (включая знаки, представляемые управляющими последовательностями), переменная или функция, значением которой является одиночный символ. Правильным обращением к функции putchar() является указание любого из этих аргументов при ее вызове.

putchar ('S'); /\* напомним, что символьные \*/

putchar (' \n'); /\* константы заключаются в апострофы \*/

putchar ('\007');

putchar (ch); /\* ch — переменная типа char \*/

putchar (getchar ());

Форму записи, приведенную в последнем примере, мы можем использовать для того, чтобы представить нашу программу в следующем виде:

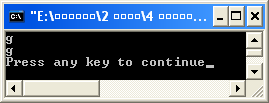
#include <stdio.h>

void main(void)

{

putchar (getchar());

}



Такая запись очень компактна и не требует введения вспомогательных переменных. Кроме того, в результате компиляции такая программа оказывается более эффективной, но, пожалуй, менее понятной.

После того как мы ознакомились с работой этих двух функций, можно перейти к обсуждению понятия буферов.

## Буферы.

При выполнении данной программы (любой из двух ее версий) вводимый символ в одних вычислительных системах немедленно появляется на экране («эхо-печать»), в других же ничего не происходит до тех пор, пока вы не нажмете клавишу [ввод]. Первый случай относится к так называемому «небуферизованному» (прямому) вводу, означающему, что вводимый символ оказывается немедленно доступным ожидающей программе. Второй случай служит примером «буферизованного» ввода, когда вводимые символы собираются и помещаются в некоторую область временной памяти, называемую «буфером». Нажатие клавиши [ввод] приводит к тому, что блок символов (или один символ) становится доступным программе. В нашей программе применяется только первый символ, поскольку функция getchar() вызывается в ней один раз. Например, работа нашей программы в системе, использующей буферизованный ввод, будет выглядеть следующим образом:

Вот длинная входная строка. [ввод]

В

В системе с небуферизованным вводом отображение на экране символа В произойдет сразу, как только вы нажмете соответствующую клавишу. Результат ввода-вывода при этом может выглядеть, например, так:

ВВот длинная входная строка.

Символ В, появившийся на второй позиции данной строки, — это непосредственный результат работы программы. В каждом случае, программой обрабатывается только один символ, поскольку функция getchar() вызывается лишь один раз.

Keyboard

Ввод АЛЛО!

!ОЛЛА

Передача всего сообщения программе

Программа

АЛЛО!

**Небуферизованный ввод**

Keyboard

Ввод АЛЛО!

По мере ввода символы

по одному поступают в

буфер

Программа

АЛЛО!

**Буферизованный ввод**

А

Л

Л

О

!

Содержимое буфера

передается программе

Буфер

Зачем нужны буферы? Во-первых, оказывается, что передачу нескольких символов в виде одного блока можно осуществить гораздо быстрее, чем передавать их последовательно по одному. Во-вторых, если при вводе символов допущена ошибка, вы можете воспользоваться корректирующими средствами терминала, чтобы ее исправить. И когда в конце концов вы нажмете клавишу [ввод], будет произведена передача откорректированной строки.

Однако для некоторых диалоговых программ небуферизованный ввод может оказаться приемлемым. Например, в программах обработки текстов было бы желательно, чтобы каждая команда вводилась, как только вы нажимаете соответствующую клавишу. Поэтому как буферизованный, так и небуферизованный ввод имеет свои достоинства.

Вы можете захотеть узнать, какой способ ввода реализован в вашей системе. Для этого необходимо выполнить нашу программу и посмотреть, как будет выглядеть результат ее работы. Некоторые компиляторы с языка Си предоставляют возможность выбора требуемого способа. В нашей микрокомпьютерной системе, например, функция getchar() реализует буферизованный ввод, между тем как функция getch() — прямой.

## Чтение данных.

Теперь возьмемся за что-нибудь несколько более сложное, чем чтение и вывод на печать одного символа — например, за вывод на печать групп символов. Желательно также, чтобы в любой момент можно было остановить работу программы; для этого спроектируем ее так, чтобы она прекращала работу при получении какого-нибудь специального символа, скажем \*. Поставленную задачу можно решить, используя цикл while:

// ввод-вывод2

// ввод и печать символов до поступления завершающего символа

#include <stdio.h>

#define STOP '\*'

void main(void)

{

char ch;

ch=getchar(); // строка 9

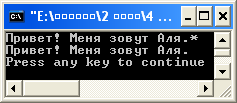
while (ch != STOP){ // строка 10

putchar(ch); // строка 11

ch=getchar(); //строка 12

}

}



В данном примере была использована структура программы, уже обсуждавшаяся нами. При первом прохождении тела цикла функция putchar() получает значение своего apгумента в результате выполнения оператора, расположенного в строке 9; в дальнейшем, вплоть до завершения работы цикла, значением этого аргумента является символ, передаваемый программе функцией getchar(), расположенной в строке 12. Мы ввели новую операцию отношения !=, смысл которой выражается словами «не равно». В результате всего этого цикл while будет осуществлять чтение и печать символов до тех пор, пока не поступит признак STOP. Мы могли бы опустить в программе директиву #define и использовать лишь символ \* в операторе while, но наш способ делает смысл данного знака более очевидным.

Перед тем как приступить к выполнению этой замечательной программы на своей машине, взгляните на ее следующий вариант. Программа, приведенная ниже, делает то же самое, но стиль ее написания лучше отвечает духу языка Си:

// ввод-вывод3

#include <stdio.h>

#define STOP '\*'

void main(void)

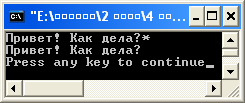
{

char ch;

while ((ch = getchar()) != STOP) // строка 8

putchar (ch);

}



Одна строка 8 этой программы заменяет строки 9, 10 и 12 программы ввод-вывод2. Как же работает этот оператор? Начнем с того, что рассмотрим содержимое внутренних скобок:

ch = getchar()

Это — выражение. Его смысл заключается в вызове функциия getchar() и присваивании полученного значения переменной ch. Одним таким действием мы выполним то, чему в программе ввод-вывод2 были посвящены строки 9 и 12. Далее напомним, что любое выражение имеет значение и что значение выражения, включающего в себя операцию присваивания, совпадает со значением переменной, расположенной слева от знака =. Следовательно, значение выражения (ch = getchar()) — это величина переменной ch, так что

(ch = getchar()) != STOP

имеет то же действие, что и

ch != STOP

Тем самым выполняется проверка, которую в программе ввод-вывод2 осуществлял оператор, расположенный в строке 10. Конструкции подобного сорта (объединение в одном выражении операций присваивания и сравнения) довольно часто используются при программировании на языке Си.

**while ((ch = getchar()) != stop)**

1-е: прием символов

2-е: присвоение символа переменной ch

3-е: проверка - не равна ли ch признаку stop

Аналогично нашему предыдущему примеру, в котором применялась конструкция while (++ size < 18,5), данная форма записи обладает тем преимуществом, что позволяет объединять в одном выражении проверку условия окончания цикла и действие по изменению одного из операндов операции сравнения. Подобная структура очень напоминает нам рассуждения, которыми мог бы сопровождаться данный процесс: «Я читаю символ, анализирую его и решаю, что делать дальше».

Теперь вернемся к нашей программе и попробуем ее выполнить. Если в вашей системе реализован небуферизованный ввод, результат может выглядеть, например, следующим образом:

ИИннттеерреесснноо,, ррааббооттааеетт ллии ооннаа ..\*

Думаю, что да.

При вводе все символы вплоть до признака STOP (звездочка) немедленно отображаются на экране (эхо-печать). Дублируются даже пробелы. Однако, как только вы ввели признак STOP, работа программы прекращается и все, что вы набираете на пульте после этого, появляется на экране без эхо-дублирования.

Теперь посмотрим, что будет происходить в системе, обладающей буферизованным вводом. В этом случае программа не начнет работать до тех пор, пока вы не нажмете на клавишу [ввод]. Вот пример возможного диалога:

Интересно, работает ли она. \*Гм, не знаю.[ввод]

Интересно, работает ли она.

Первая строка была целиком передана программе. Программа последовательно читает эту строку по одному символу и также по одному символу выводит на печать до тех пор, пока не встретит символ \*.

Теперь напишем несколько более полезную программу. Мы заставим ее подсчитывать символы, которые она читает. Нам требуется для этого ввести в предыдущую программу лишь некоторые изменения.

// подсчет символов!

// X22.C

#include <stdio.h>

#define STOP '\*'

void main(void)

{

char ch;

int count=0;

printf("Для окончания работы введите <\*>\n");

while ((ch=getchar()) != STOP)

{

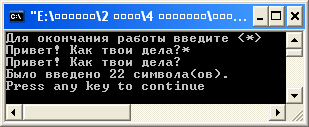
putchar(ch);

count++;

}

printf("\nБыло введено %d символf(ов).\n",count);

}



Если мы хотим просто подсчитывать число введенных символов без отображения их на экране, функцию putchar() можно опустить. Эта маленькая программа подсчитывает символы, и нам осталось сделать всего лишь несколько шагов для получения программы, которая будет подсчитывать строки и слова.

### Чтение строки.

Давайте подумаем, какие дополнительные усовершенствования можно ввести в программу, используя только те средства, которыми мы владеем. Первое, что легко можно сделать — это заменить признак окончания ввода данных. Но можно ли предложить что-то лучшее, чем символ \*? Одной из возможностей является использование символа «новая строка» (\n). Для этого нужно лишь переопределить признак STOP:

#define STOP '\n'

Какой это даст эффект? Очень большой: ведь символ «новая строка» пересылается при нажатии клавиши [ввод]; следовательно, в результате наша программа будет обрабатывать одну вводимую строку. Предположим, например, что мы внесли указанное изменение в программу “подсчет символов!”, а затем при ее выполнении ввели следующую строку:

О! Быть сейчас во Фресно, когда здесь лето. [ввод]

В ответ на экране появятся следующие строки:

О! Быть сейчас во Фресно, когда здесь лето

Всего было просчитано 43 символа.

(Если бы мы не включили в управляющую строку оператора printf() в качестве первого символа признак \n, второе сообщение появилось бы справа от запятой, после слова лето. Мы предпочли избежать такого склеивания строк.)

Признак, появляющийся в результате нажатия клавиши [ввод], не входит в число символов (43), подсчитанных программой, поскольку подсчет осуществляется внутри цикла.

Теперь у нас есть программа, которая может прочесть одну строку. В зависимости от того, какой оператор помещен в тело цикла while, программа может осуществлять эхо-печать, подсчет числа символов в строке, или и то и другое. Эти средства представляются нам в принципе полезными, скажем, как часть некоторой большей программы. Но было бы хорошо иметь возможность читать большие порции данных, например, файл данных. Это может быть осуществлено путем надлежащего выбора признака STOP.

### Чтение файла.

Каким может быть идеальный признак STOP? Это должен быть такой символ, который обычно не используется в тексте и, следовательно, не приводит к ситуации, когда он случайно встретится при вводе, и работа программы будет остановлена раньше, чем мы хотели бы.

Проблема подобного сорта не нова, и, к счастью для нас, она уже была успешно решена проектировщиками вычислительных систем. На самом деле задача, которую они рассматривали, была несколько отличной от нашей, но мы вполне можем воспользоваться их решением. Занимавшая их проблема касалась «файлов». Файлом можно назвать участок памяти, в который помещена некоторая информация. Обычно файл хранится в некоторой долговременной памяти, например, на гибких или жестких дисках, или на магнитной ленте. Чтобы отмечать, где кончается один файл и начинается другой, полезно иметь специальный символ, указывающий на конец файла. Это должен быть символ, который не может появиться где-нибудь в середине файла, точно так же как выше нам требовался символ, обычно не встречающийся во вводимом тексте. Решением указанной проблемы служит введение специального признака, называемого «End-of-File» (конец файла), или EOF, для краткости. Выбор конкретного признака EOF зависит от типа системы: он может состоять даже из нескольких символов. Но такой признак всегда существует, и компилятор с языка Си, которым вы пользуетесь, конечно же знает, как такой признак действует в вашей системе.

_07_03

Каким образом можно воспользоваться символом EOF? Обычно его определение содержится в файле <stdio.h>. Общеупотребительным является определение:

#define EOF (- 1)

Это дает возможность использовать в программах выражения, подобные, например, такому:

while((ch=getchar()) != EOF)

Поэтому мы можем переписать нашу предыдущую пporpaммy, осуществляющую ввод и эхо-печать символов, так:

// ввод-вывод4

#include <stdio.h>

void main(void)

{

int ch;

while ((ch = getchar()) != EOF)

putchar (ch);

}

// X23.C

#include <stdio.h>

void main(void)

{

int ch;

int count=0;

printf("Для окончания работы введите <CTRL/Z>\n");

while ((ch=getchar()) != EOF)

{

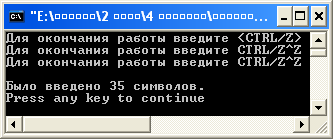
putchar(ch);

count++;

}

printf("\n Было введено %d символов.\n",count);

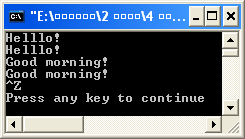
}



Отметим следующие моменты:

1. Нам не нужно самим определять признак EOF, поскольку заботу об этом берет на себя файл <stdio.h>.
2. Мы можем не интересоваться фактическим значением символа EOF, поскольку директива #define, имеющаяся в файле <stdio.h>, позволяет нам использовать его символическое представление.
3. Мы изменили тип переменной ch с char на int. Мы поступили так потому, что значениями переменных типа char являются целые числа без знака в диапазоне от 0 до 255, а признак EOF может иметь числовое значение -1. Эта величина недопустима для переменной типа char, но вполне подходит для переменной типа int. К счастью, функция getchar() фактически возвращает значение типа int, поэтому она в состоянии прочесть символ EOF.
4. Переменная ch целого типа никак не может повлиять на работу функции putchar(). Она просто выводит на печать символьный эквивалент значения аргумента.
5. При работе с данной программой, когда символы вводятся с клавиатуры, необходимо уметь вводить признак EOF. He думайте, что вы можете просто указать буквы E-O-F или число -1. (Число -1 служит эквивалентом кода ASCII данного символа, а не самим этим символом.) Вместо этого вам необходимо узнать, какое представление используется в вашей системе. В большинстве реализаций операционной системы UNIX, например, ввод знака [CTRL/D] (нажать на клавишу [D], держа нажатой клавишу [CTRL]) интерпретируется как признак EOF. Во многих микрокомпьютерах для той же цели используется знак [CTRL/Z].

Приведем результат работы программы ввод-вывод4 в системе, обладающей буферизованным вводом:



Каждый раз при нажатии клавиши [ввод] производится обработка символов, попавших в буфер, и копия строки выводится на печать. Это продолжается до тех пор, пока мы не введем признак EOF.

Давайте остановимся и подумаем о возможностях программы ввод-вывод4. Она осуществляет вывод на экран символов независимо от того, откуда они поступают. Предположим, мы сумели сделать так, что программа вводит символы из некоторого файла. В этом случае она будет осуществлять вывод содержимого файла на экран и остановится только тогда, когда достигнет конца файла, поскольку обнаружит признак EOF. Или предположим, что у нас есть способ организовать вывод результатов работы программы в некоторый файл. Тогда можно набрать какой-нибудь текст на клавиатуре и при помощи программы ввод-вывод4 поместить его во внешнюю память. Или мы могли бы выполнить оба действия одновременно: например, осуществить ввод данных из одного файла и переслать их в другой. В этом случае программа ввод-вывод4 использовалась бы для копирования файлов. Следовательно, наша маленькая программа могла бы просматривать содержимое файлов, создавать новые файлы и получать копии файлов. Неплохо для такой короткой программы! Ключ к решению этих проблем — в управлении вводом и выводом. Это послужит темой представленного ниже обсуждения.

## Переключение и работа с файлами.

Понятие ввода-вывода включает в себя функции, данные и устройства. Рассмотрим, например, нашу программу ввод-вывод4. В ней используется функция getchar(), осуществляющая ввод, причем устройство ввода — клавиатура (в соответствии с нашим предположением), а входные данные — отдельные символы. Нам бы хотелось сохранить функции ввода и тип данных, но изменить источник их поступления в программу. Зададимся вопросом: откуда программа узнает, где искать входные данные?

По умолчанию Си-программа рассматривает «стандартный ввод» как источник поступления данных. «Стандартным вводом» называется устройство, принятое в качестве обычного средства ввода данных в машину. Это может быть устройство чтения данных с магнитной ленты или перфокарт, телетайп или (как мы продолжаем считать) терминал. Современная машина — это послушный инструмент, и мы можем воздействовать на нее так, чтобы она вводила данные из любого источника. В частности, мы можем сообщить программе, что источник входных данных — файл, а не клавиатура.

Существуют два способа написания программ, работающих с файлами. Первый способ заключается в явном использовании специальных функций, которые открывают и закрывают файлы, организуют чтение и запись данных и т. п.; мы не хотим пока касаться этого вопроса. Второй способ состоит в том, чтобы использовать программу, спроектированную первоначально в предположении, что данные в нее вводятся с клавиатуры и выводятся на экран, но переключить ввод и вывод на другие информационные каналы, например, из файла в файл. Этот способ в некоторых отношениях обладает меньшими возможностями, чем первый, но зато гораздо проще в использовании. Мы изучим понятие переключения в данном разделе.

Операция переключения — это средство ОС UNIX, а не самого языка Си. Но она оказалась настолько полезной, что при переносе компилятора с языка Си на другие вычислительные системы часто вместе с ним переносится в какой-то форме и эта операция. Более того, многие из вновь созданных операционных систем, таких, как MS-DOS 2, включают в себя данное средство. Поэтому, даже если вы не работаете в среде ОС UNIX существует большая вероятность того, что вы в той или иной форме сможете воспользоваться операцией переключения. Мы обсудим сначала возможности этой операции в ОС UNIX, а затем и в других системах.

## Операционная система UNIX.

### Переключение вывода.

Предположим, вы осуществили компиляцию программы ввод-вывод4 и поместили выполняемый объектный код в файл с именем getput4. Затем, чтобы запустить данную программу, вы вводите с терминала только имя файла

getput4

и программа выполняется так, как было описано выше, т. е. получает в качестве входных данных символы, вводимые с клавиатуры. Теперь предположим, что вы хотите посмотреть, как наша программа работает с «текстовым файлом» с именем words. (Текстовой файл — это файл, содержащий некоторый текст, т. е. данные в виде символов. Это может быть, например, очерк или программа на языке Си. Файл, содержащий команды на машинном языке, например файл, полученный в результате компиляции данной программы, не является текстовым. Поскольку наша программа занимается обработкой символов, она должна использоваться вместе с текстовыми файлами.) Все, что для этого требуется — ввести вместо команды, указанной выше, следующую:

getput4 < words

Символ < служит обозначением операции переключения, используемой в ОС UNIX. Выполнение указанной операции приводит к тому, что содержимое файла words будет направлено в файл с именем getput4. Сама программа ввод-вывод4 не знает (и не должна знать), что входные данные поступают из некоторого файла, а не с терминала; на ее вход просто поступает поток символов, она читает их и последовательно, по одному выводит на печать до тех пор, пока не встретит признак EOF. В операционной системе UNIX файлы и устройства ввода-вывода в логическом смысле представляют собой одно и то же, поэтому теперь файл для данной программы является устройством ввода-вывода. Если вы попробуете ввести команду

getput4 < words

то в результате на экране могут появиться, например, следующие строки:

В одном мгновеньи видеть вечность,

Огромный мир - в зерне песка,

В единой горсти - бесконечность,

И небо - в чашечке цветка

Но мы, конечно, не можем гарантировать, что в файле, который выберете вы, тоже окажется четверостишие Вильяма Блейка.

### Переключение ввода.

Теперь предположим (если вы еще не устали и в состоянии что-нибудь предположить), вам хочется, чтобы слова, вводимые с клавиатуры, попадали в файл с именем mywords. Для этого вы должны ввести команду

getput4 > mywords

и начать ввод символов. Символ > служит обозначением еще одной операции переключения, используемой в ОС UNIX. Ее выполнение приводит к тому, что создается новый файл с именем mywords, а затем результат работы программы ввод-вывод4, представляющий собой копию вводимых символов, направляется в данный файл. Если файл с именем mywords уже существует, он обычно уничтожается, и вместо него создается новый. (В некоторых реализациях ОС UNIX, однако, вам предоставляется возможность защитить существующие файлы.) На экране в данном случае появятся лишь вводимые вами символы, их же копии будут направлены в указанный файл. Чтобы закончить работу программы, введите признак EOF (в системе UNIX это обычно символ [CTRL/D]). Попробуйте воспользоваться описанной здесь операцией. Если вам ничего другого не придет в голову, просто воспроизведите на своей машине пример, приведенный ниже. Знак приглашения, выводимый на экран интерпретатором команд SHELL, обозначается здесь символом %. Не забывайте оканчивать каждую введенную строку символом [возврат], чтобы содержимое буфера пересылалось в программу.

% getput4 > mywords

У вас не должно быть трудностей с запоминанием того, какая операция переключения для чего предназначена. Необходимо помнить только, что знак каждой операции указывает на направление информационного потока. Вы можете по ассоциации представлять себе этот знак в виде воронки. [CTRL/D]

%

После того как введен символ [CTRL/D], программа заканчивает свою работу и возвращает управление операционной системе UNIX, на что указывает повторное появление знака приглашения. Как убедиться в том, что наша программа вообще работала? В ОС UNIX существует команда Is, которая выводит на экран имена файлов; обращение к ней должно продемонстрировать вам, что файл с именем mywords теперь существует. Чтобы проверить его содержимое, вы можете воспользоваться командой cat или запустить заново программу ввод-вывод4, направляя в нее на этот раз содержимое входного файла.

% getput4 < mywords

У вас не должно быть трудностей с запоминанием того, какая операция переключения для чего предназначена. Необходимо помнить только, что знак каждой операции указывает на направление информационного потока. Вы можете по ассоциации представлять себе этот знак в виде воронки.

### Комбинированное переключение.

Предположим теперь, что вы хотите создать копию файла mywords и назвать ее savewords. Введите для этого команду

getput4 < mywords >savewords

и требуемое задание будет выполнено. Команда

getput4 >savewords < mywords

приведет к такому же результату, поскольку порядок указания операций переключения не имеет значения.

Нельзя использовать в одной команде один и тот же файл и для ввода и для вывода одновременно.

getput4 <mywords >mywords /\*НЕПРАВИЛЬНО\*/

Причина этого заключается в том, что указание операции >mywords приводит к стиранию исходного файла перед его использованием в качестве входного.

Теперь, мы думаем, настало время суммировать правила, касающиеся использования двух операций переключения < и >.

1. Операция переключения связывает выполняемую программу (в том числе и стандартные команды ОС UNIX) с некоторым файлом. Она не может использоваться для связи одного файла с другим или одной программы с другой.
2. Имя выполняемой программы должно стоять слева от знака операции, а имя файла — справа от него.
3. При использовании этих операций ввод не может осуществляться более чем из одного файла, а вывод — более чем в один файл.
4. Обычно между именем и операцией пробелы не обязательны кроме тех редких случаев, когда используются некоторые символы специального назначения в интерпретаторе команд UNIX. Мы могли бы писать, например, так: getput4< words, или, что более предпочтительно, getput4 < words.

Мы уже привели выше несколько примеров правильного использования операций переключения. Ниже дается несколько ошибочных примеров (addup и count — выполняемые программы, a fish и stars — текстовые файлы).

fish > stars //Нарушение правила 1

addup < count //Нарушение правила 1

stars > count //Нарушение правила 2

addup < fish < stars //Нарушение правила 3

count > stars > fish //Нарушение правила 3

В ОС UNIX применяются также операция >>, позволяющая добавлять данные в конец существующего файла, и операция «канал» (]), связывающая файл вывода одной программы с вводом другой. Для получения более детальной информации обо всех этих операциях вам необходимо обратиться к руководству по ОС UNIX (по аналогии с этим нам приходит в голову название «ОС UNIX: руководство для начинающих»).

Рассмотрим еще один пример: напишем очень простую программу, шифрующую сообщения. C этой целью мы немного изменим программу ввод-вывод4 и получим:

// простой шифр

// заменяет каждый символ текста

// следующим по порядку из кода ASCII

// X24.C

#include <stdio.h>

void main(void)

{

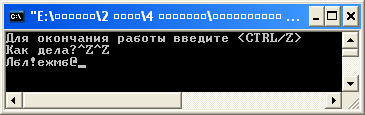
char ch;

printf("Для окончания работы введите <CTRL/Z>\n");

while ((ch=getchar()) != EOF)

putchar(ch+1);

}



Функция putchar() переводит целое «ch + 1» в соответствующий символ.

Выполните теперь компиляцию программы и поместите выполняемый объектный код в файл с именем simplecode. Затем занесите приведенные ниже строки (Перевод: знание орфографии — залог четкости письма) в файл с именем original. (Для этого можно воспользоваться системным текстовым редактором или, как было показано ранее, программой ввод-вывод4.)

Good spelling is an aid

to clear writing.

Теперь введите команду

simplecode < original

Результат должен выглядеть приблизительно так:

!!!!!Hppe!tqfmmjoh!jtbo!bje^Kup!dmfbs!xsjujohl^k

Буква G заменится на Н, о на р и т. д. Вас может удивить следующее: во-первых, что пробелы превратились в восклицательные знаки. Это служит напоминанием, что пробел — такой же символ, как и все остальные. Во-вторых, две строки слились в одну. Почему? Потому что в тексте, содержащемся в файле original, в конце первой строки находится символ «новая строка», служащий указанием компьютеру начать вывод следующего слова с новой строки. Но этот символ также был изменен. В нашей системе он был заменен символом ^К, являющимся аналогом специального символа [CTRL/K], и поэтому последующий вывод на печать был продолжен на прежней строке. Если мы хотим иметь программу шифровки сообщений, сохраняющую первоначальную структуру текста (по строкам), нам необходимо средство, позволяющее изменять все символы, кроме символа «новая строка». В дальнейшем мы узнаем, как это сделать.

### Конвейерная пересылка

Можно также осуществить соединение выходного потока одной программы с входным потоком другой. Это называется конвейерной пересылкой. Если имеются две выполнимые программы example1 и example2, то конвейерная пересылка между ними организуется с помощью символа вертикальной черты ‘|’. Следующая строка организует конвейерную пересылку между example1 и example2:

example1 | example2

Организацию конвейерной пересылки обеспечит операционная система.

### Операционные системы, отличные от ОС UNIX.

Здесь мы главным образом рассмотрим, чем отличаются другие операционные системы от ОС UNIX; поэтому, если вы пропустили предыдущий раздел, вернитесь назад и прочтите его (это не займет у вас много времени). Все отличия можно разделить на две группы:

1. В других операционных системах реализована операция переключения.
2. Компиляторы с языка Си предоставляют возможность использовать операцию переключения.

Мы не можем рассмотреть все возможные операционные системы, поэтому приведем пример только одной из них, но весьма широко распространенной. Это система MS-DOS 2; она вначале была просто «отпрыском» ОС СР/М, а сейчас самостоятельно развивается в сторону операционной системы XENIX, подобной ОС UNIX. В версию 2 MS-DOS были введены операции переключения <и>; они работают в ней точно так же, как было описано в предыдущем разделе.

У нас нет возможности рассмотреть все компиляторы с языка Си. Однако в пяти из шести версий компилятора, предназначенных для микрокомпьютеров, с которыми мы имели дело, для указания операции переключения используются символы <и>. Операция переключения, реализуемая компилятором с языка Си, отличается от аналогичной операции, выполняемой ОС UNIX, в двух аспектах :

1. Указанная операция выполняется при работе программ, написанных только на Си, в то время как в ОС UNIX она может использоваться при работе любой программы.
2. Между именем программы и знаком операции должен быть один пробел, а между знаком операции и именем файла пробел должен отсутствовать. Ниже приведен пример правильной команды

getput4 < words

**Комментарий**. Операция переключения — это простое, но мощное средство. С ее помощью мы можем превратить нашу крошечную программу ввод-вывод4 в инструмент для создания, чтения и копирования файлов. Данный способ служит иллюстрацией подхода, принятого в языке Си (и ОС UNIX) и заключающегося в конструировании простых средств, которые можно комбинировать различные образом для выполнения конкретных задач.

**Резюме**: как переключать ввод и вывод. На большинстве машин, в которых реализован компилятор с языка Си, операцию переключения можно использовать либо для всех программ, благодаря поддержке операционной системы, либо только для программ, написанных на Си, благодаря наличию компилятора с этого языка. Ниже prog будет именем выполняемой программы, a file1 и file2 — именами файлов.

**Переключение вывода в файл: >**

prog >file1

**Переключениe ввода из файла: <**

prog <file2

Комбинированное переключение:

prog <file2 >file1 или

prog >file1 <file2

В обеих формах записи файл с именем file2 используется для ввода данных, а файл с именем file1 — для вывода.

**Расположение пробелов.** Некоторые системы (в особенности компиляторы с языка Си) требуют наличия пробела слева от знака операции переключения и его отсутствия справа от этого знака. Другие системы (ОС UNIX, например) допускают любое число пробелов (в том числе и ни одного) слева и справа от знака данной операции.

**Графический пример.** Мы можем воспользоваться функциями getchar() и putchar() для изображения геометрических фигур при помощи символов. Ниже приведена программа, которая это делает. Она читает символ, а затем печатает его некоторое число раз, зависящее от кода ASCII этого символа. Кроме того, она печатает на каждой строке требуемое число пробелов, чтобы текст оказывался в центре строки.

// фигуры

// изображает симметричную фигуру из символов

#include <stdio.h>

void main(void)

{

int ch; // переменная для ввода символа

int index;

int chnum;

while ((ch = getchar()) != '\n'){

chnum = ch %26; // получение числа от 0 до 25

index = 0;

while (index++ <(30 - chnum))

putchar (' '); // печать пробелов сдвига к центру

index = 0;

while (index++ <(2\* chnum + 1))

putchar (ch); // повторная печать символа

putchar (' \n');

}

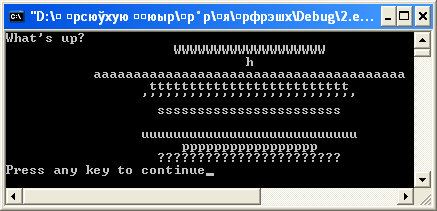
}

Единственный новый технический прием здесь — это использование подвыражений, таких, как (30-chnum), при записи условий в циклах while. Один цикл while управляет печатью необходимого числа начальных пробелов в каждой строке, а второй — выводом символов на печать.

Результат работы программы зависит от данных, которые вводятся. Если, например, вы введете

What's up? (Что случилось?)

то на экране появится следующее:



Что вы можете делать с помощью этой программы? Можете просто игнорировать ее, или же (переписав ее по-другому) изменять вид фигур, которые она выводит на печать, либо, наконец, искать такие комбинации входных символов, что в результате на экране будут появляться привлекательные фигуры: например, при вводе такой последовательности

hijklmnopqrstuiii

результат работы программы будет выглядеть так:

h

iii

jjjjj

kkkkkkk

lllllllllllllllll

mmmmmmmm

nnnnnnnnnnnnn

ooooooooooooooo

ррррррррррррррррр

qqqqqqqqqqqqqqqqqqq

rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr

ssssssssssssssssssssssssssss

tttttttttttttttttttttttttttttttttttttttttttttttttttttttttt

uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu

iii

iii

iii

## Системно-зависимые средства: порты ввода-вывода микропроцессоров INTEL 8086/8088.

Рассмотрим различные устройства ввода-вывода, поскольку теперь мы хотим обсудить вопрос о том, как приспособить реализацию компилятора с языка Си к требованиям конкретной вычислительной системы. Многие из современных микрокомпьютеров спроектированы на основе микропроцессорных интегральных схем (ИС) INTEL 8086 и INTEL 8088. Наиболее известным является персональный компьютер IBM PC, в котором применяются ИС второго типа. Конкретный пример, который приведен ниже, относится к упомянутому компьютеру, но обсуждаемые принципы применимы и при рассмотрении других мини-машин, построенных на базе семейства микропроцессоров 8086/8088.

В компьютере типа IBM PC кроме ИС 8088 имеются и другие устройства, например клавиатура, громкоговоритель, возможно, накопитель на мини-кассете или магнитном диске, монитор, встроенная память, таймеры, а также микропроцессоры для управления потоком данных. Центральный процессор (встроенный в кристалл 8088) должен иметь возможность взаимодействовать с остальными частями компьютера. Некоторые из таких взаимодействий осуществляются при помощи адресов памяти, другие — при помощи «портов» ввода-вывода. У микропроцессора 8088 имеется 65536 портов, которые могут использоваться при различных взаимодействиях. Для связи с этим микропроцессором каждому устройству назначается свой определенный порт или порты. (Заметим, что используются не все 65536 портов!) Например, порты 992, 993, 1000—1004 используются для связи с адаптером цветной графики. Работа громкоговорителя управляется портом с номером 97. Это выглядит несколько проще, чем управление адаптером цветной графики, поэтому мы используем его для иллюстрации работы портов ввода-вывода.

Порт 97 не управляет непосредственной работой громкоговорителя. Устройство, осуществляющее эти функции, называется «Программируемый параллельный интерфейсный контроллер 8255». Этот микропроцессор имеет три «регистра» (небольших, легко доступных элемента памяти), в каждом из которых содержится некоторое число. Числа в регистрах используются для управления работой данного устройства. Каждый регистр связан с ИС 8088 через порт, и регистру, управляющему громкоговорителем, выделен для связи порт 97. С его помощью управление данным устройством осуществляется путем изменения числа в регистре. При посылке «правильного» числа громкоговоритель издает звуковой сигнал; посылка же «неправильного» числа может вызвать ряд проблем. Поэтому нам необходимо знать, какие числа требуется посылать и как их нужно посылать. В частности, нам хотелось бы знать, как использовать язык Си для подобного рода операций.

Давайте сначала посмотрим, какие нужно посылать числа. Первое, что необходимо знать — регистр контроллера 8255 может принять 8-разрядное число, которое помещается туда в двоичном коде, например, 01011011. Каждый из восьми разрядов памяти рассматривается как переключатель «включено-выключено» для соответствующего устройства или воздействия. Наличие 0 или 1 в соответствующей позиции определяет, включено или нет соответствующее устройство. Например, разряд 3 (разряды нумеруются от 0 до 7 справа налево) определяет, включен или нет электродвигатель накопителя на мини-кассете, а разряд 7 разрешает или запрещает работу с клавиатурой терминала. При передаче числа в регистр необходимо соблюдать осторожность. Если при включении громкоговорителя мы не обратим внимания на остальные разряды, то случайно можем выключить клавиатуру! Поэтому давайте посмотрим, чему соответствует каждый разряд. (Используемая информация взята из технического справочного руководства фирмы IBM, и мы вовсе не должны знать, что большинство из этих разрядов означает.)

_07_04

разряд 0 + включение громкоговорителя через таймер 2

разряд 1 + наличие данных для работы громкоговорителя

разряд 2 + (чтение ключа размера оперативной памяти) или (чтение резервного ключа)

разряд 3 + выключение двигателя накопителя на мини-кассете

разряд 4 - разблокировка оперативной памяти

разряд 5 - разблокировка контроля ввода-вывода

раздяр 6 - поддержание низкой тактовой частоты задающего генератора клавиатуры

разряд 7 - (разблокировка клавиатуры) или + (сброс клавиатуры & разрешение опроса программно-опрашиваемых переключателей)

Обратите внимание на знаки + и -. Знак + указывает, что в соответствующем разряде выполнение условия обозначается через 1, а знак - указывает, что выполнение условия в разряде обозначается через 0. Поэтому 1 в 3-м разряде показывает, что двигатель накопителя на мини-кассете выключен, в то время как 0 в 4-м разряде указывает на возможность доступа к памяти.

Каким образом можно включить громкоговоритель? Оказывается, для этого необходимо в 0-й разряд (включение громкоговорителя через таймер 2) и в 1-й разряд (наличие данных для работы громкоговорителя) заслать 1. Это означает, что для включения громкоговорителя через порт 97 необходимо послать в регистр двоичное число 11 (или десятичное число 3). Но, перед тем как приступить к этому, учтите, что данная операция имеет такие побочные эффекты, как, например, установка разряда 4 в 0, что может оказаться вовсе нежелательным. Одна из причин, по которой мы не рассказали, как использовать порты, заключается в том, чтобы предотвратить неприятные последствия вашей поспешности.

Для надежности мы должны проверить сначала, что содержится в регистре. К счастью, это совсем не трудно (мы продемонстрируем это чуть позже). Ответ выглядит так: в регистре обычно содержатся числа «76» или «77». Давайте переведем их в двоичную систему. (Здесь вам, возможно, захочется заглянуть в таблицу преобразования в двоичный код, которая приводится в конце книги в приложении.) Результаты преобразования некоторых чисел приведены в таблице.

Не вдаваясь в подробности по поводу значения слов «поддержание низкой тактовой частоты задающего генератора клавиатуры», можно сказать, что надежный способ выполнения указанной операции заключается в том, чтобы оставить без изменения значения всех разрядов, кроме нулевого и первого. Это достигается путем передачи в регистр двоичного числа 01001111(или десятичного 79). В качестве дополнительных мер предосторожности мы должны запомнить исходное значение, содержащееся в регистре, а затем после звукового сигнала громкоговорителя восстановить содержимое указанного регистра. (Битовые операции, рассматриваемые в приложении в конце данной книги, предоставляют другую возможность для занесения некоторого значения в регистр.)

Таблица. Двоичное преобразование некоторых десятичных чисел

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Десятичное число** | **Номер разряда** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 76 |  | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 77 |  | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 78 |  | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 79 |  | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |

Теперь мы готовы к тому, чтобы заставить громкоговоритель подать звуковой сигнал.

### Использование порта.

Существуют две операции, которые могут выполняться с помощью порта: микропроцессор 8088 может послать информацию в подсоединенное устройство или прочитать данные из него. В языке ассемблера эти операции выполняются при помощи команд OUT и IN, а в языке Си использование указанных средств зависит от компилятора. Некоторые из них предоставляют возможность вызова специальных функций (в соответствии с тем, как это обычно делается в языке Си). В компиляторах Lattice С и Supersoft С, например, с этой целью применяются функции outp() и inp(), в других же аналогичные функции могут носить другие имена. Если вы работаете с компилятором, в котором такие возможности отсутствуют, для задания указанных функций можно либо воспользоваться ассемблером, либо просто включить в свою программу соответствующий ассемблерный код (что очень просто). В любом случае вам необходимо ознакомиться с документацией по вашему компилятору. Пока же будем предполагать, что у вас имеется возможность вызова функций outp() и inp().

Приведем пример программы, представляющей собой первую попытку извлечь звуковой сигнал из громкоговорителя:

/\* сигнал1 \*/

/\* заставляет громкоговоритель подавать сигнал \*/

#include <stdio.h>

void main()

{

int store;

/\* запоминание начального значения с помощью порта 97 \*/

store= inp (97);

printf("пopт 97 = %d\n", store);/\* проверка результатов \*/

/\* посылает 79 в порт 97; включение громкоговорителя \*/

outp(97, 79);

/\* восстановление начального значения \*/

outp(97, store);

}

/\* X25.C \*/

#include <stdio.h>

#include <dos.h>

#define MAX1 32000

#define MAX2 3000

void main()

{

int store,

count=0;

store=inport(97);

while(count++ < MAX1)

outport(97,79);

outport(97,store);

/\* ....................................... \*/

count=0;

while(count++ < MAX1)

;

count=0;

while(count++ < MAX1)

;

count=0;

while(count++ < MAX1)

;

/\* ....................................... \*/

count=0;

while(count++ < MAX2)

outport(97,79);

outport(97,store);

}

Несмотря на то что, по-видимому, вы и сами можете догадаться, что выполняют функции inp() и outp(), ниже приведем их формальное описание:

inp(номер порта) Эта функция возвращает (т. е. формирует) 8-разрядное целое значение (которое преобразуется в 16-разрядное число типа int путем добавления нулей слева), полученное из порта ввода с указанным номером. Обращение к ней не зависит от номера подключенного порта.

outp (номер порта, значение) Эта функция передает 8-разрядное целое значение в порт вывода с указанным номером.

Заметим, что один и тот же порт может быть как портом ввода, так и портом вывода в зависимости от того, как он используется.

Давайте теперь выполним программу. В итоге вы, можете быть не совсем удовлетворены, поскольку компьютер выключает громкоговоритель довольно быстро после включения. Было бы лучше, если бы мы смогли заставить компьютер подождать немного, прежде чем выключить громкоговоритель. Как это можно сделать? Довольно просто! Нужно только дать компьютеру какую-нибудь работу на это время. Приведенная ниже программа показывает, как этого достичь.

/\* сигнал2 \*/

/\* более длинный сигнал \*/

#include <stdio.h>

#define LIMIT 10000

void main()

{

int store;

int count = 0; /\* счетчик для организации задержки \*/

store = inp (97);

outp (97, 79);

while (count++ < LIMIT)

; /\* задержка на время работы пустого цикла \*/

outp (97, store);

}

Отметим, что вся работа оператора while состоит в увеличении (на каждом шаге цикла) значения переменной count до тех пор, пока оно не станет равным величине константы LIMIT. Символ «точка с запятой», следующий за оператором while, — это «пустой» оператор, который не выполняет никаких действий. Поэтому программа сигнал2 включает громкоговоритель, считает до 10000, а затем выключает его. Вы можете изменять значение константы LIMIT, чтобы регулировать продолжительность звучания, или можете заменить константу LIMIT переменной и использовать функцию scanf() для ввода соответствующего значения, определяющего длительность сигнала.

Было бы прекрасно иметь возможность регулировать и высоту тона. Это и в самом деле осуществимо. После того как мы изучим функции более полно, в приложении в конце книги вы сможете познакомиться с программой, которая превращает клавиатуру терминала в клавиатуру музыкального инструмента.

Резюме. Мы опять имели дело с устройствами, данными и функциями ввода-вывода. В качестве устройств рассматривались контроллер 8255 и громкоговоритель, в качестве данных — числа, пересылаемые в один из регистров контроллера (а также из него), в качестве функций — функции inp() и oupt(). Использование этих функций или их эквивалентов на ассемблере необходимо в том случае, если мы хотим пользоваться портами ввода-вывода ИС INTEL 8086/8088, и компиляторы с языка Си предоставляют нам одну или обе эти возможности.

Использование скрытой мощности (в лошадиных силах) вашего компьютера. Хотите узнать чудовищный потенциал машины для «перемалывания чисел»? Как раз для этого мы написали замечательную программу, приведенную ниже. Чтобы оценить ее полностью, вам необходимо выполнить ее на вашем компьютере. Предупреждение: для получения желаемого эффекта вы должны выбрать подходящую для вашей системы величину константы LIMIT. Дополнительные подробности будут обсуждены ниже, а сначала рассмотрим саму программу.

// Ганс

#include <stdio.h>

#define LIMIT 8000L

void main()

{

int num1, num2;

long delay =0;

int count = 0;

printf (" Лошадь-компьютер Ганс сложит для вас два очень");

printf ("маленьких целых числа\n");

printf (“Укажите, пожалуйста, первое маленькое число.\n");

scanf ("%d", &num1);

printf ("Спасибо. А теперь введите второе число\n");

scanf ("%d", &num2);

printf ("Итак, Ганс, сколько у тебя получится?\n");

while (delay++ < LIMIT);

while (count++ < num1 + num2 - 1)

{

putchar ('\007');

delay = 0;

while (delay++ < LIMIT);

putchar ('\n');

}

printf (" Да? И это все?\n" );

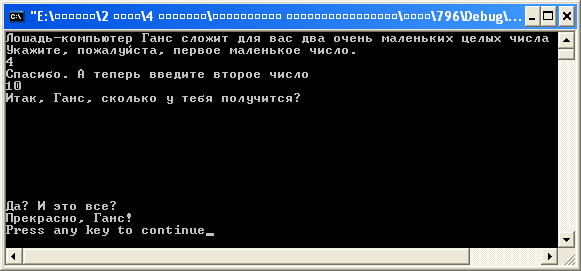
delay = 0;

while (delay++ < 3\*LIMIT);

putchar ('\007');

printf ("Прекрасно, Ганс! \n");

}



Технические замечания: операторы while, в которых содержится переменная delay, не делают ничего другого, кроме организации задержки по времени. Символ «точка с запятой» в конце строки указывает на конец тела цикла while, т. е. последующие операторы в него не входят. Цикл while, использованный внутри другого цикла while, называется «вложенным». Мы полагаем, что на IBM PC подходящим значением для константы LIMIT является число 8000. На VAX 11/750 мы предпочитаем число порядка 50000, но на выбор может влиять также уровень загрузки системы, работающей в режиме разделения времени. Мы полагаем LIMIT равной значению константы типа long (как раз на это и указывает символ L, стоящий в конце) для того, чтобы избежать трудностей, связанных с превышением максимального значения величины типа int. (Для числа 8000 подобные меры предосторожности на самом деле необязательны, но, например, его замена числом 12000 на IBM PC делает это необходимым, поскольку тогда выражение 3\*LIMIT будет равно 36000, что превышает максимальное значение величины типа int в этой системе.)

Если в вашей вычислительной системе отсутствует громкоговоритель или звонок, вы могли бы заменить оператор putchar('\007') на printf ("Стук копыт").

Эта программа произведет впечатление на ваших друзей и, возможно, успокоит тех, кто боится компьютеров.

Мы думаем, такая программа может составить ядро какого-нибудь «Си-вычислителя», но оставляем развитие этой идеи нашим читателям.

**Что вы должны были узнать.**

Что делает функция getchar(): вводит в программу символ, поступающий с клавиатуры терминала.

Что делает функция putchar(ch): отображает символ, содержащийся в переменной ch, на экран.

Что символ != означает: не равно.

Что такое EOF: специальный символ, указывающий на конец файла.

Как переключить стандартный ввод на ввод из файла: program < file

Как переключить стандартный вывод на вывод в файл: program > file

Что такое порты: средства доступа к подсоединенным устройствам. Как использовать порты: путем вызова функций inp() и outp().

**Вопросы и ответы.**

***Вопросы:***

Выражение putchar(getchar()) является правильным. Будет ли правильным выражение getchar(putchar())?

Что произойдет в результате выполнения каждого из следующих операторов?

а. putchar ('H');

б. putchar (' \007');

в. putchar (' \n');

г. putchar (' \b').

Допустим, у вас есть программа count, подсчитывающая число символов в файле. Напишите команду, в результате выполнения которой будет произведен подсчет числа символов в файле essay, а результат будет помещен в файл essayct.

Даны программа и файлы, описанные в вопросе 3. Какие из приведенных ниже команд правильны?

а. essayct < essay

б. count essay

в. count < essayct

г. essay > count

Что делает оператор outp(212, 23)?

**Ответы.**

1. Нет. У функции getchar() аргумент должен отсутствовать, а у функции putchar() аргумент обязательно должен быть.

а. печать буквы Н

б. вывод символа ' \007', в результате чего сработает громкоговоритель.

в. переход на новую строку на устройстве вывода.

г. шаг назад на одну позицию.

1. count <essay >essayct или иначе count >essayct <essay.

а. неправильно, поскольку essayct не является выполняемой программой.

б. неправильно, поскольку опущен знак операции переключения (Позже вы научитесь писать программы, для которых не нужно будет использовать операцию переключения.)

в. правильно, при выполнении этой команды число символов, полученное в результате работы программы count из вопроса 3, появится в виде сообщения на экране.

г. неправильно, имя выполняемой программы должно стоять первым.

5. Он посылает число 23 через порт 212???????????

**Упражнения.**

1. Напишите программу, описанную в п. 3, т. е программу, подсчитывающую число символов в файле.
2. Модифицируйте программу count так, чтобы при учете каждого символа раздавался звуковой сигнал. Введите в программу короткий цикл, реализующий временную задержку, для того чтобы отделить один сигнал от другого.
3. Модифицируйте программу сигнал2 так, чтобы во время ее выполнения можно было вводить величину, определяющую число повторений тела цикла.