МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Изучаем программирование на C

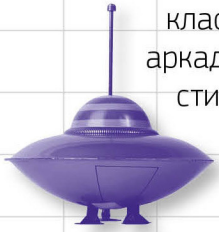
Открой для себя секреты гуру-программистов

Узнай, как утилита make может изменить твою жизнь

Научись избегать глупых ошибок при работе с указателями

Представь, как функции с переменным параметром помогают Сью быть более гибкой

Порезвись в стандартной библиотеке C



Создай классическую аркадную игру в стиле ретро

Дэвид Гриффитс
Дон Гриффитс

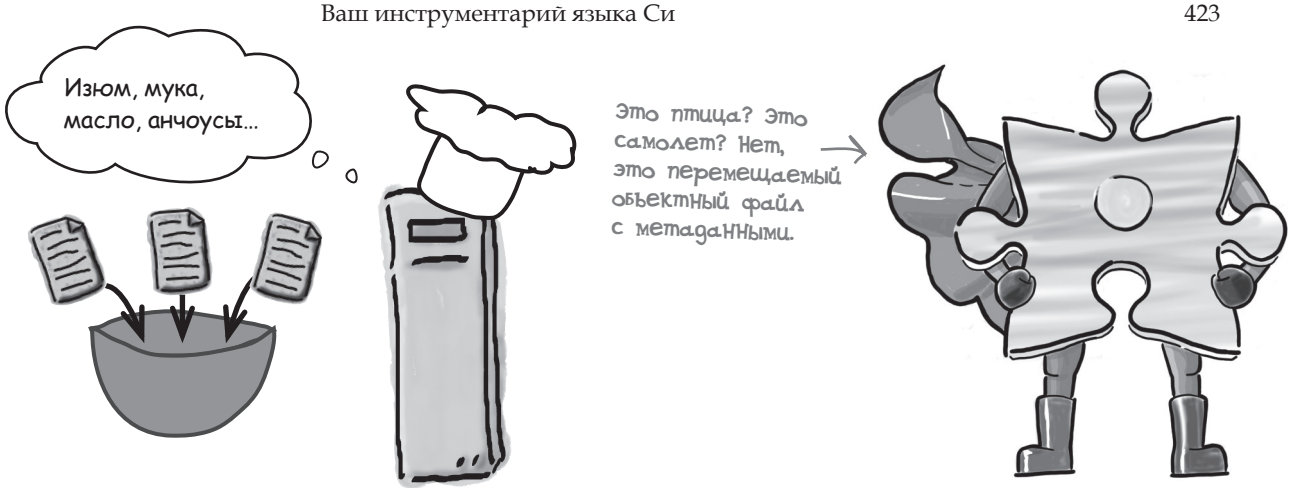
ЭКСМО
O'REILLY®

8 Статические и динамические библиотеки

Легко заменяемый код

Вы уже знаете, какой мощью обладают стандартные библиотеки.

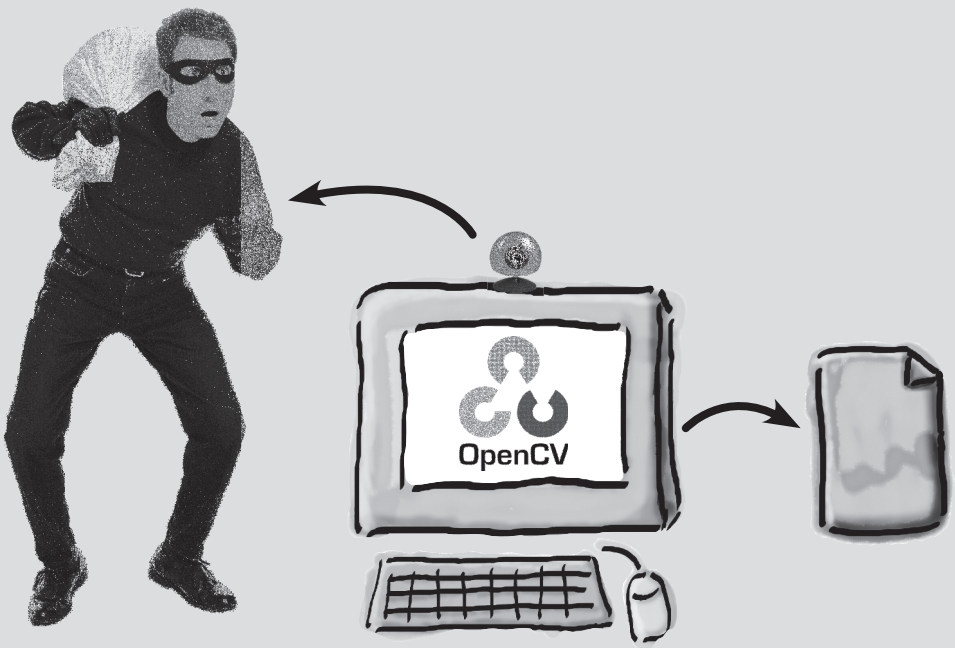
Пришло время применить эту мощь в *собственном* коде. Здесь вы узнаете, как создавать **собственные библиотеки** и **использовать один и тот же код в нескольких приложениях**. Вы научитесь разделять код во время выполнения программы с помощью **динамических библиотек**. И дочитав эту главу, вы сможете писать масштабируемый, простой и эффективный в управлении код.



Лабораторная работа 2

OpenCV

Представьте, что компьютер в ваше отсутствие может присматривать за домом и сообщать о подозрительных личностях, которые бродят вокруг. В лабораторной работе 2 вы создадите детектор для обнаружения незваных гостей, используя возможности языка Си и мастерство OpenCV.

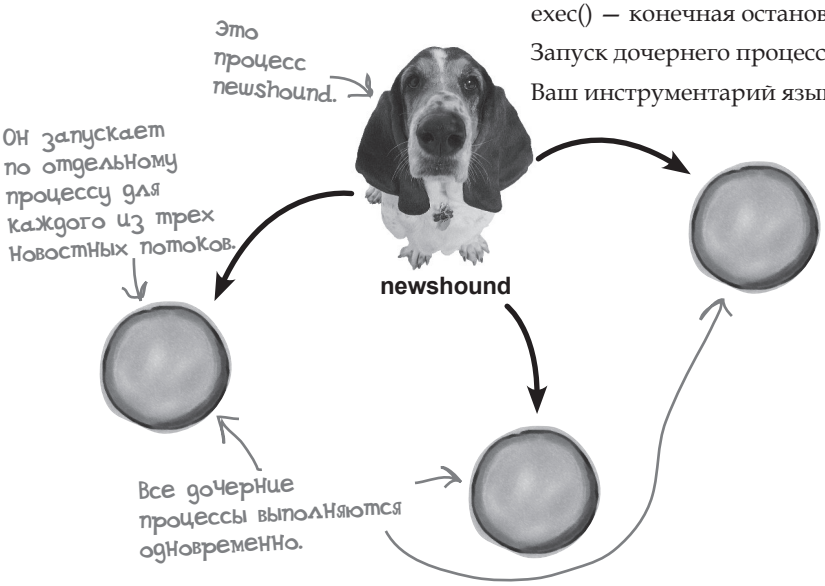


Процессы и системные вызовы
Разрушая границы

9

Пришло время мыслить нестандартно.

Вы уже знаете, что с помощью командной строки можно собирать сложные приложения из небольших утилит. Но что если вы хотите *вызывать другие программы* из собственного кода? В этой главе вы научитесь использовать **системные сервисы** для создания ***процессов*** и управления ими. Благодаря этому вы получите доступ к *электронной почте*, *браузеру* и *любому установленному внешнему приложению*. К концу этой главы вы сможете выйти ***за рамки языка Си***.

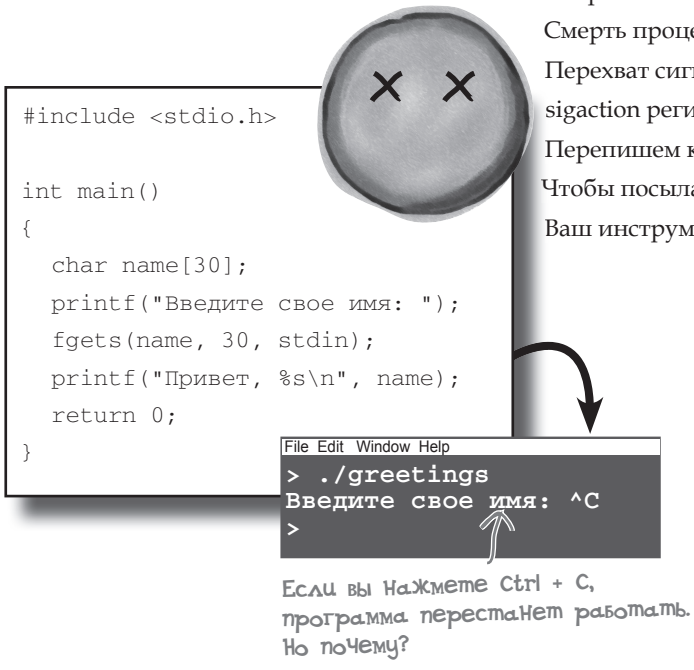


10 Межпроцессорное взаимодействие

Общение — это хорошо

Создание процессов — это только первый шаг.

Что если вам захочется *управлять* запущенным процессом? Что если вам захочется *отправить ему данные*? Или *прочитать его вывод*? Благодаря **межпроцессному взаимодействию** процессы могут *выполнять работу* сообща. Вы увидите, что **мощь** вашего кода может возрасти многократно, если позволить ему ***общаться*** с другими программами системы.



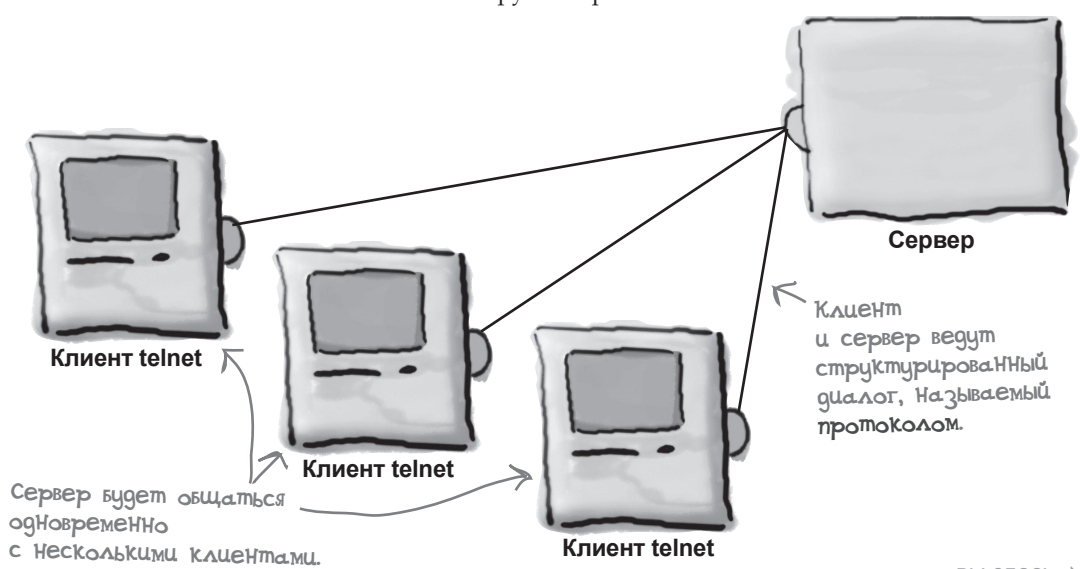
Если вы нажмете Ctrl + C, программа перестанет работать. Но почему?

11 Сокеты и работа в Сети

Нет места лучше, чем 127.0.0.1

Программам на разных компьютерах тоже нужно общаться.

Вы уже научились использовать ввод/вывод для организации взаимодействия с файлами и узнали, каким образом могут общаться два процесса на одном и том же компьютере. Теперь вы сможете *разговаривать со всем остальным миром*, создавая код на языке Си, способный взаимодействовать с другими программами **по Сети в любой точке *планеты***. К концу этой главы вы научитесь создавать как **серверные**, так и **клиентские** приложения.

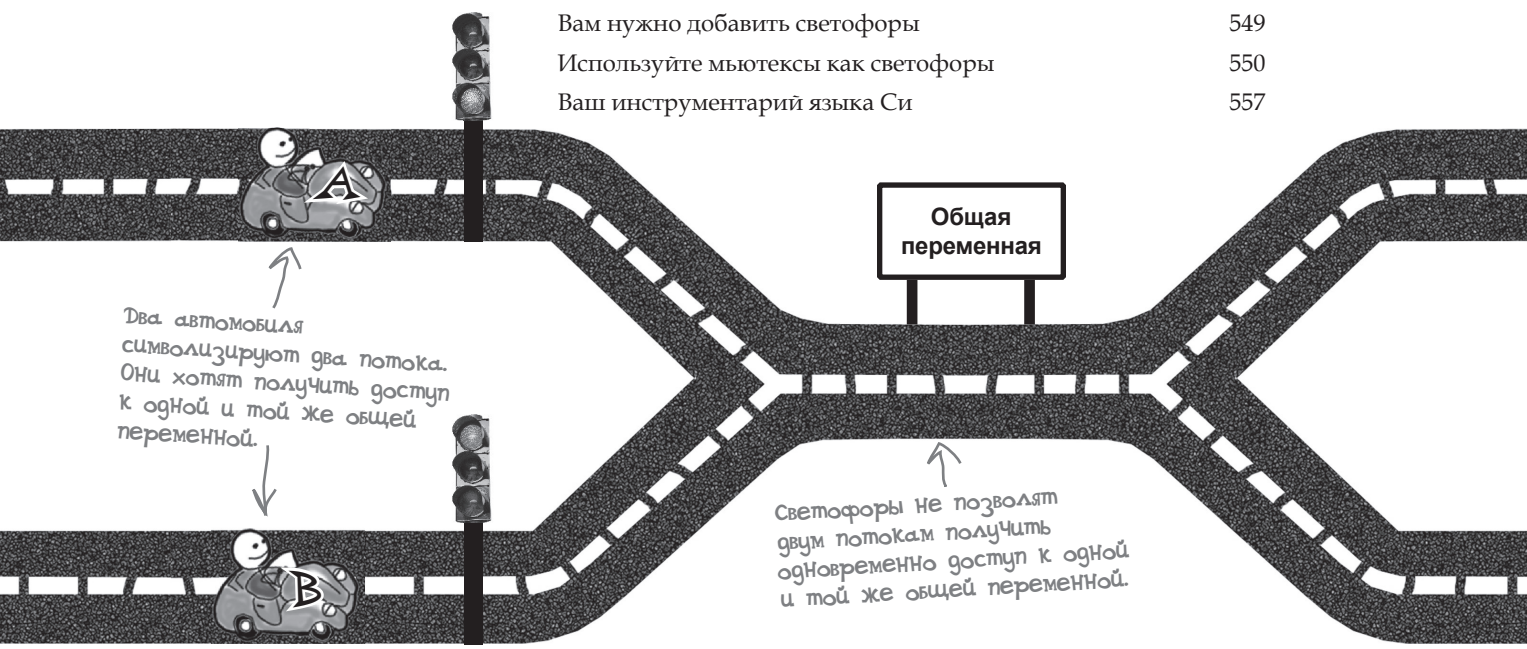


Потоки

12 Это параллельный мир

Программам часто приходится выполнять несколько заданий одновременно.

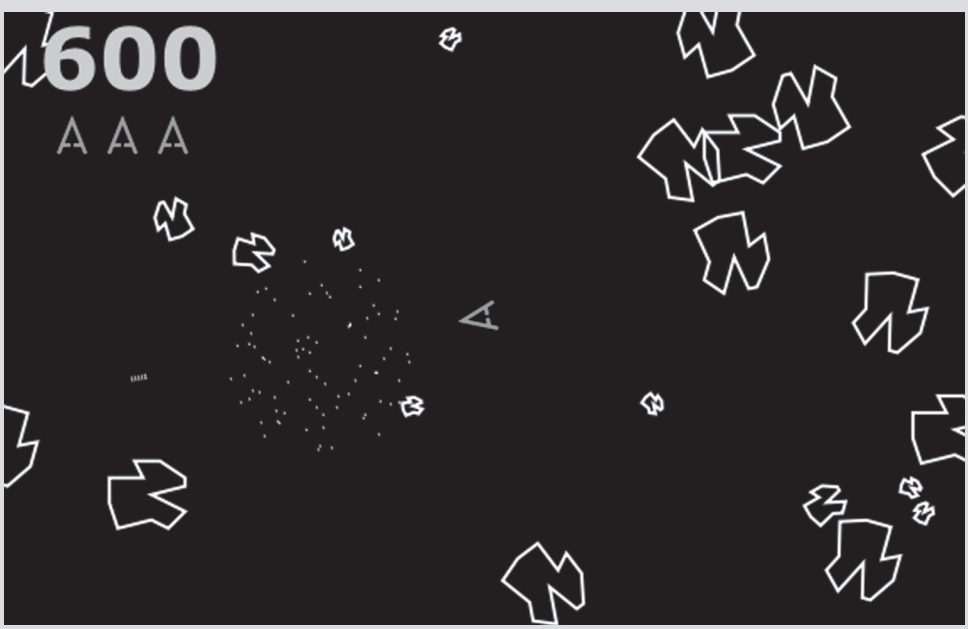
С помощью стандартных POSIX-потоков вы можете сделать код более отзывчивым, **разделяя его на несколько параллельно выполняемых частей**. Но будьте осторожны! Потоки являются мощным инструментом, однако вряд ли вы захотите, чтобы они столкнулись друг с другом. В этой главе вы научитесь расставлять светофоры и наносить **разметку** так, чтобы ***в вашем коде не образовывались пробки***. В результате вы узнаете, как создавать **POSIX-потоки** и как с помощью **механизмов синхронизации *сохранять целостность хрупких данных***.



Лабораторная работа 3

Бластероиды

В лабораторной работе 3 вы отдадите дань уважения одной из самых популярных и долговечных видеоигр. Пришло время написать «Бластероиды»!



Приложение I На закуску

Топ-10 фактов (которым мы не уделили внимание)

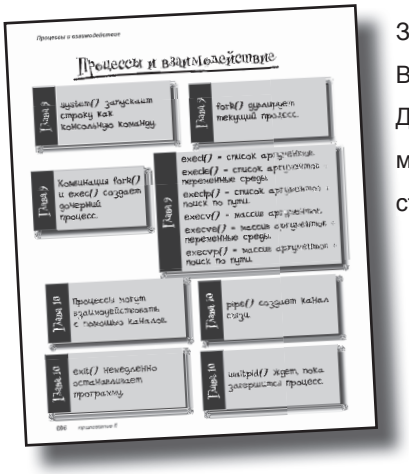
Даже после всего сказанного еще кое-что осталось.

Есть несколько фактов, о которых, как нам кажется, вы должны знать. С одной стороны, нам не хотелось делать эту книгу объемной настолько, чтобы ее нельзя было поднять без специальной физической подготовки. Но с другой стороны, мы считаем, что кое-какие моменты заслуживают хотя бы краткого упоминания и проигнорировать их было бы неправильно. Поэтому прежде чем окончательно отложить книгу в сторону, **ознакомьтесь со следующими темами**.

Приложение II Все темы

Вспомнить все

Всегда хотелось собрать все замечательные факты о Си в одном месте?


Процессы и взаимодействие

Процессы и взаимодействие

system() запускает строку как командную команду

fork() дублирует текущий процесс.

Комбинация fork() и exec() создает дочерний процесс.

exec() - список аргументов
execle() - список аргументов, переменные среды
execlp() - список аргументов, поиск по пути
execv() - массив аргументов
execve() - массив аргументов, переменные среды
execvp() - массив аргументов, поиск по пути

Процессы могут взаимодействовать с помощью каналов

pipe() создает канал связи

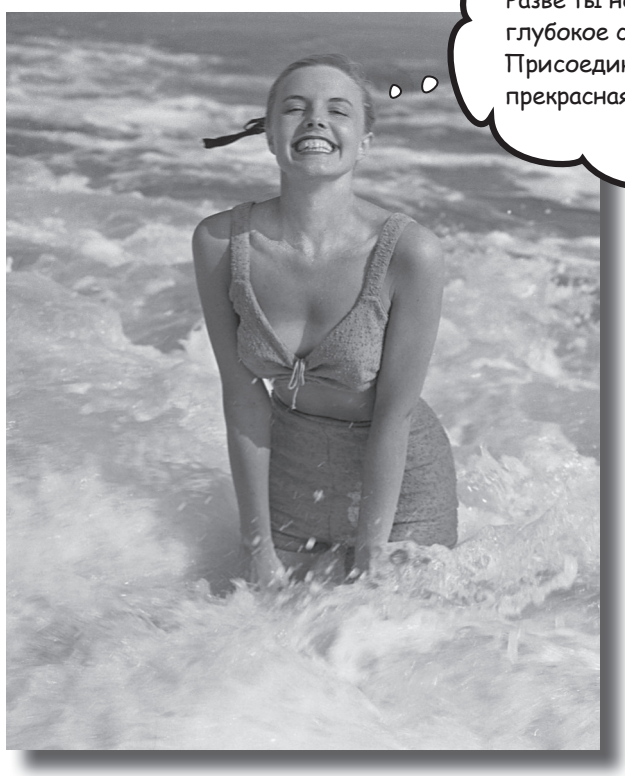
exit() немедленно останавливает программу

wait(pid) ждет, пока завершится процесс.

Здесь мы собрали все темы и правила, которые были затронуты в этой книге. Взгляните на них и постарайтесь вспомнить все, о чем мы вам рассказали. Для каждого факта указана глава, где он был рассмотрен, в случае чего вы можете запросто вернуться назад. Возможно, вы даже захотите вырвать эти страницы и приклеить их на стену.

1 Начинаем работать с языком Си

Погружаемся



Хотите узнать, как думает компьютер?

Вам нужно написать **высокопроизводительный код** для новой игры? Запрограммировать контроллер **Arduino**? Или использовать продвинутую **стороннюю библиотеку** в своем приложении для iPhone? Тогда язык Си вам поможет. Си работает на **гораздо более низком уровне**, чем большинство других языков программирования, поэтому понимание Си позволит вам лучше разобраться в том, **что на самом деле происходит внутри компьютера**. Си также может помочь в освоении других языков. Поэтому хватайте свой компилятор и немедленно приступайте к делу.

Си — это язык для компактных быстрых программ

Язык Си предназначен для написания компактных быстрых программ. Он более низкоуровневый, чем большинство других языков программирования. Это означает, что *созданный с его помощью код наиболее понятен для компьютера*.

Как работает Си

В действительности компьютеры понимают только один язык — машинный код — бинарный поток единичек и нулей. Ваш код на Си преобразуется в машинный с помощью **компилятора**.

1 Исходник

Все начинается с создания исходного файла. Исходный файл содержит код на языке Си, пригодный для восприятия человеком.

2 Компиляция

Исходный код пропускается через компилятор. Компилятор ищет ошибки и, если все хорошо, компилирует исходный код.

3 Вывод

Компилятор создает новый файл, который называется *исполняемым*. Этот файл содержит машинный код — поток единиц и нулей, понятный компьютеру. Это и есть программа, которую вы запускаете.

Си используется там, где важны скорость и объем. Большинство операционных систем написано на Си. Многие компьютерные языки также написаны на Си, равно как и большая часть игровых программ.

Вы можете столкнуться с тремя стандартами языка Си. ANSI C был создан в конце 1980-х гг. и используется для наиболее старого кода. В 1999 г. многие аспекты языка были исправлены в стандарте C99. А в 2011 г. появился современный стандарт C11 с некоторыми новыми интересными возможностями. Отличия между разными версиями не очень большие, и мы будем указывать на них время от времени.

Наточите свой карандаш

Попытайтесь угадать, что делает каждый из этих фрагментов.

Опишите действия, которые, по вашему мнению, выполняет этот код.

```
int card_count = 11;
if (card_count > 10)
    puts("Карты в колоде удачные. Повышаем ставку.");
```

.....
.....
.....

```
int c = 10;
while (c > 0) {
    puts("Я не должен писать код в виде
классов");
    c = c - 1;
}
```

.....
.....
.....
.....
.....

```
/* Подразумевается, что имя короче 20 латинских (и 10
кириллических) символов. */
char ex[20];
puts("Введите имя вашей девушки:");
scanf("%19s", ex);
printf("Дорогая %s. \n\n\tC тобой покончено.\n”, ex);
```

.....
.....
.....
.....
.....

```
char suit = 'H';
switch(suit) {
case 'C':
    puts("Clubs(Трефы)");
    break;
case 'D':
    puts("Diamonds(Бубны)");
    break;
case 'H':
    puts("Hearts(Черви)");
    break;
default:
    puts("Spades(Пики)");
}
```

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Наточите свой карандаш
Решение

Не волнуйтесь, если вам что-то непонятно. Далее в этой книге будут даны более подробные пояснения.

```
int card_count = 11;
if (card_count > 10)
    puts("Карты в колоде удачные. Повышаем
ставку.");
```

int – целое число.

Здесь текст выводится в командной строке или на терминал.

Создаем целочисленную переменную и устанавливаем для нее значение 11.
Счетчик больше 10?
Если да, то выводим сообщение в командной строке.

```
int c = 10;
while (c > 0) {
    puts("Я не должен писать код в виде
классов");
    c = c - 1;
}
```

Фигурные скобки определяют блочную инструкцию.

Создаем целочисленную переменную и устанавливаем для нее значение 10.
До тех пор пока значение больше 0,...
...выводим сообщение...
...и уменьшаем счетчик.
Конец кода, который должен повторяться.

```
/* Подразумевается, что имя короче 20 латинских
(и 10 кириллических) символов. */
char ex[20];
puts("Введите имя вашей девушки: ");
scanf("%19s", ex);
printf("Дорогая %s. \n\n\tC тобой покончено.\n",
ex);
```

Это означает «поместить все, что ввел пользователь, внутрь массива ex».

Здесь строка символов вставляется на место %s.

Это комментарий.
Создаем массив из 20 символов.
Выводим сообщение на экран.
Сохраняем в массив то, что ввел пользователь.
Выводим сообщение, включающее введенный текст.

```
char suit = 'H';
switch(suit) {
case 'C':
    puts("Clubs(Трефы)");
    break;
case 'D':
    puts("Diamonds(Бубны)");
    break;
case 'H':
    puts("Hearts(Черви)");
    break;
default:
    puts("Spades(Пики)");
}
```

Инструкция switch проверяет одну и ту же переменную на разные значения.

Создаем символьную переменную, помещаем в нее букву «H».
Проверяем значение переменной.
Оно равно «C»?
Если да, то выводим «Clubs (Трефы)».
Затем пропускаем остальные проверки.
Оно равно «D»?
Если да, то выводим «Diamonds (Бубны)».
Затем пропускаем остальные проверки.
Оно равно «H»?
Если да, то выводим «Hearts (Черви)».
Затем пропускаем остальные проверки.
В противном случае...
...выводим «Spades (Пики)».
На этом проверки заканчиваются.

Как выглядит программа на Си целиком

Чтобы создать полноценную программу, вам необходимо написать свой код на языке Си и сохранить его в *исходный файл*. Исходные файлы можно создавать с помощью любого текстового редактора, имя файла при этом, как правило, заканчивается на *.c*.

Это всего лишь условность, но вы должны ее соблюдать.

Давайте взглянем на типичный исходный файл на языке Си.

1 **Программы на Си обычно начинаются с комментария.** Комментарий описывает назначение кода, содержащегося в файле, и, возможно, содержит информацию о лицензии и авторском праве. Острой необходимости добавлять его сюда (или в любую другую часть файла) нет, но так принято, и большинство программистов на Си рассчитывают подобный комментарий там увидеть.

Комментарий начинается с символов /*.

Эти звездочки необязательны и нужны только для красоты.

Комментарий заканчивается символами */.

```
/*
 * Программа для подсчета количества карт в колоде.
 * Этот код выпущен под публичной Лас-Вегасской лицензией.
 * (c)2014, Команда колледжа по блек-джеку.
 */
```

2 **Дальше следует часть с подключением внешних файлов.** Си — очень лаконичный язык, и без использования *внешних библиотек* он практически ни на что не способен. Компилятору необходимо сообщить, какой внешний код использовать, подключая заголовочные файлы соответствующих библиотек. Чаще других будет встречаться заголовок *stdio.h*. Это библиотека `stdio`, содержащая код для вывода данных на терминал и считывания их оттуда.

```
#include <stdio.h>
```

```
int main()
{
    int decks;
    puts("Введите количество колод.");
    scanf("%i", &decks);
    if (decks < 1) {
        puts("Вы ввели недопустимое количество колод.");
        return 1;
    }
    printf("Всего карт %i\n", (decks * 52));
    return 0;
}
```

3 **Последнее, что вы найдете в исходном файле, — это функции.**
Весь код на Си выполняется внутри функций. Самая важная функция в любой программе, написанной на Си, называется **главной — `main()`**. Она является отправной точкой для всего кода в вашей программе.

Давайте рассмотрим функцию main() более подробно.