Глава 8

Принятие решений

В этой главе...

- Сравнения условий с помощью `if`
- Использование операторов сравнения
- Добавление `else` к решению
- Структура `if-else-if-else`
- Принятие логических решений
- Структура `switch-case`
- Использование тернарного оператора

Принятие решений является частью программирования, которая заставляет вас думать об уме компьютера. Компьютер, конечно, — не интеллектуальная личность, но чтобы можно обмануть почти любого, код должен иметь возможность варьировать свое поведение в зависимости от выполнения определенных условий или сравнений.

А что если?..

Все человеческие драмы основаны на неповиновении. Независимо от используемых правил и их строгости в средине истории появляется какой-то шутник, который их с легкостью нарушает, — и с этой минуты история становится куда интереснее. Любое приключение начинается с простого вопроса “А что если?..” Точно такая же концепция используется и в программировании, хотя здесь нам достаточно только одного слова — “если” (`if`).

Простое сравнение

Мы постоянно принимаем решения. Что надеть с утра? Следует ли отложить поход к стоматологу или пора бежать к нему без оглядки? Постараться ли сегодня не встретиться с боссом или, напротив, желательно зайти к нему в кабинет для беседы? Эти решения основаны на сравнениях. Мы узнаем, какая нас ждет погода, мы оцениваем силу зубной боли, мы узнаем у секретаря шефа о его настроении — и сравниваем эту информацию с имеющейся в нашей памяти. Компьютер поступает точно так же, только сравнивает не абстрактные концепции, а конкретные значения (листинг 8.1).

Листинг 8.1. Простое сравнение

```
#include <stdio.h>

int main()
{
    int a,b;

    a = 6;
    b = a - 2;

    if (a > b)
    {
        printf("%d больше, чем %d\n",a,b);
    }
    return(0);
}
```

Упражнение 8.1. Создайте новый проект Code::Blocks и введите в него исходный текст из листинга 8.1. Сохраните и постройте проект, а затем выполните полученную программу. Вывод программы должен имеет следующий вид:

```
6 больше, чем 4
```

Быстро и верно. Вот как работает этот код.

В строке 5 объявлены две целочисленные переменные: `a` и `b`. Эти переменные получают значения в строках 7 и 8, при этом переменная `b` получает значение, на 2 меньшее значения переменной `a`.

В строке 10 выполняется сравнение:

```
if (a > b)
```

Программисты читают эту строку как "если *a* больше, чем *b*". Или, если они только начинают изучать язык программирования C, то как "если значение переменной `a` больше значения переменной `b`". Скобки при этом не читаются.

Строки с 11 по 13 принадлежат инструкции `if`. Мясом в этом сандвиче является строка 12; фигурные скобки не играют иной роли, кроме ограничения инструкции в строке 12. Если сравнение в строке 10 истинно, выполняется инструкция в строке 12. В противном случае все инструкции в фигурных скобках пропускаются.

Упражнение 8.2. Отредактируйте исходный текст из листинга 8.1 так, чтобы вместо вычитания в строке 8.8 выполнялось сложение. Можете ли вы объяснить вывод программы?

Ключевое слово `if`

Для принятия решения на основе простого сравнения в исходном тексте используется ключевое слово `if`. Вот формат его применения:

```
if (условие)
{
    инструкция;
}
```

Здесь *условие* представляет собой сравнение, математическую операцию, результат функции или некоторое иное условие. Если это условие истинно, выполняется *инструкция* (или инструкции) в фигурных скобках; в противном случае они пропускаются.

- ✓ Условие в инструкции `if` не обязано быть математическим. Это может быть вызов функции, которая возвращает значение "истина" или "ложь", например:

  ```
  if(ready())
  ```

  Эта инструкция проверяет значение, возвращаемое функцией `ready()`. Если функция возвращает значение "истинно", выполняются инструкции, принадлежащие инструкции `if`.
- ✓ В языке программирования C в качестве истинного рассматривается любое ненулевое значения. Нулевое значение рассматривается как ложное. Поэтому данная инструкция всегда истинна:

  ```
  if(1)
  ```

  А эта инструкция всегда ложна:

  ```
  if(0)
  ```
- ✓ В инструкции `if` вы не можете сравнивать таким же образом строки — для сравнения строк следует использовать специальные функции сравнения строк, которые рассматриваются в главе 13, "Работа с текстом".
- ✓ Если к инструкции `if` относится единственная инструкция, фигурные скобки не обязательны.

Упражнение 8.3. Перепишите код из листинга 8.1, удалив фигурные скобки перед и после строки 12. Сохраните и постройте проект, выполните полученную программу и убедитесь, что она корректно работает.

Различные сравнения значений

Язык программирования C использует несколько математических операторов сравнения. Я собрал их в табл. 8.1, которую вы можете использовать как справку по операторам сравнения.

Таблица 8.1. Операторы сравнения языка программирования C

Оператор	Пример	Истинно, когда
`!=`	`a != b`	`a` не равно `b`
`<`	`a < b`	`a` меньше, чем `b`
`<=`	`a <= b`	`a` меньше или равно `b`
`==`	`a == b`	`a` равно `b`
`>`	`a > b`	`a` больше, чем `b`
`>=`	`a >= b`	`a` больше или равно `b`

Сравнения в языке программирования C работают слева направо, т.е. оператор “больше или равно” выглядит как >=, а не как =>. Порядок имеет значение для всех операторов, кроме оператора равенства (листинг 8.2).

Листинг 8.2. Сравнение значений

```
#include <stdio.h>

int main()
{
    int first,second;

    printf("Введите первое значение: ");
    scanf("%d",&first);
    printf("Введите второе значение: ");
    scanf("%d",&second);

    puts("Вычисляю...");
    if(first<second)
    {
        printf("%d меньше, чем %d\n",first,second);
    }
    if(first>second)
    {
        printf("%d больше, чем %d\n",first,second);
    }
    return(0);
}
```

Упражнение 8.4. Создайте новый проект Code::Blocks и введите в него исходный текст из листинга 8.2. Сохраните и постройте проект, а затем выполните полученную программу.

Пожалуй, наиболее распространенным оператором сравнения является двойной знак равенства. Вам он может показаться странным, но оператор == — не то же самое, что оператор =. Оператор = является *оператором присваивания*, который устанавливает значения переменных. Оператор == представляет собой *оператор сравнения*, который проверяет, равны ли два значения (листинг 8.3).

Упражнение 8.5. Добавьте новый раздел в исходный текст из листинга 8.2, который выполняет последнюю проверку — равны ли введенные значения между собой.

Листинг 8.3. Проверка равенства значений

```
#include <stdio.h>

#define SECRET 17

int main()
{
    int guess;
```

```
    printf("Угадайте секретное число: ");
    scanf("%d",&guess);
    if(guess==SECRET)
    {
        puts("Вы угадали!");
        return(0);
    }
    if(guess!=SECRET)
    {
        puts("Вы ошиблись!");
        return(1);
    }
}
```

Упражнение 8.6. Введите исходный текст из листинга 8.3 в новый проект Code::Blocks. Сохраните и постройте проект, а затем выполните полученную программу.

Обратите внимание на возвращаемое программой значение: 0 — при верном ответе и 1 — в противном случае. Это значение вы можете увидеть в окне вывода Code::Blocks.

Разница между = и ==

Одна из наиболее распространенных ошибок, которые делал каждый программист на языке C — как новичок, так и профессионал, — это использование вместо двойного знака равенства одинарного. Взгляните на листинг 8.4.

Листинг 8.4. Всегда истинно

```
#include <stdio.h>

int main()
{
    int a;

    a = 5;

    if(a=-3)
    {
        printf("%d равно %d\n",a,-3);
    }
    return(0);
}
```

Упражнение 8.7. Введите исходный текст из листинга 8.4 в новый проект. Сохраните и постройте проект, а затем выполните полученную программу.

Вывод может вас озадачить. Вы видите следующее:

```
-3 равно -3
```

Ну, вообще говоря, это истинное утверждение, но ведь `a` должно быть равно 5? Что же произошло?

Все очень просто. В строке 9 переменной a присваивается значение -3. Поскольку эта инструкция находится в круглых скобках, она вычисляется первой. Результат присваивания переменной в языке программирования C всегда является истинным для любого ненулевого значения.

Упражнение 8.8. Отредактируйте исходный текст из листинга 8.4, заменив символ равенства двойным символом равенства, представляющим собой операцию сравнения. Выполните программу и убедитесь в ее корректной работе.

Точка с запятой не на своем месте

Листинг 8.5 основан на листинге 8.4, с использованием того факта, что если сравнению if принадлежит единственная инструкция, то ее можно не заключать в фигурные скобки.

Листинг 8.5. Проблема с точкой с запятой

```
#include <stdio.h>

int main()
{
    int a,b;

    a = 5;
    b = -3;

    if(a==b);
        printf("%d равно %d\n",a,b);
    return(0);
}
```

Упражнение 8.9. Аккуратно и точно перенесите код из листинга 8.5 в свой проект. Особое внимание уделите строке 10. Убедитесь, что она введена в точности так, как представлена в листинге. Сохраните и постройте проект, а затем выполните полученную программу.

Вот что вы должны увидеть:

```
5 равно -3
```

Проблема достаточно распространенная, и эту ошибку время от времени делает каждый программист на языке C: завершающая точка с запятой в листинге 8.5 (строка 10) говорит программе о том, что при истинности условия инструкция if не должна ничего делать. Дело в том, что точка с запятой сама по себе является завершенной, хотя и пустой, инструкцией C:

```
if (условие)
    ;
```

Эта конструкция, по сути, та же, что в строке 10 в листинге 8.5. Будьте внимательны и не повторяйте эту ошибку — особенно при вводе исходного текста большого размера, когда вы уже чисто автоматически завершаете каждую строку точкой с запятой.

Множественные решения

Не каждое решение является выбором между “да” и “нет”. Мы постоянно сталкиваемся с выбором из нескольких возможных альтернатив. Язык программирования C обеспечивает несколько способов работы с такими исключениями, позволяя выбирать одну из нескольких возможностей.

Более сложное решение

Для выполнения выбора в ситуации “или–или” ключевое слово `if` имеет напарника — ключевое слово `else`. Вместе они работают следующим образом:

```
if (условие)
{
    инструкция или инструкции;
}
else
{
    инструкция или инструкции;
}
```

Когда *условие* в структуре `if-else` истинно, выполняется инструкция (или инструкции) из раздела `if`; в противном случае выполняются инструкции из раздела `else`. Так осуществляется принятие решения “или–или”.

Листинг 8.6 представляет собой измененный код из листинга 8.1. Структура, состоящая из единственного `if`, заменена структурой `if-else`. Если сравнение в `if` ложно, выполняется инструкция, принадлежащая части `else` кода.

Листинг 8.6. Сравнение `if-else`

```
#include <stdio.h>

int main()
{
    int a,b;

    a = 6;
    b = a - 2;

    if( a > b)
    {
        printf("%d больше %d\n",a,b);
    }
    else
    {
        printf("%d не больше %d\n",a,b);
    }
    return(0);
}
```

Упражнение 8.10. Введите исходный текст из листинга 8.6 в новый проект. Сохраните и постройте проект, а затем выполните полученную программу.

Упражнение 8.11. Измените исходный текст так, чтобы пользователь мог ввести значение переменной `b`.

Упражнение 8.12. Измените исходный текст из листинга 8.3, заменив структуру `if-else` уродливой конструкцией `if-if`. (*Указание*: оптимальное решение состоит в замене только одной строки исходного текста.)

Добавление третьей альтернативы

Но не всякое решение, принимаемое в программе, является решением "или–или". Для описания структуры, приведенной ниже, нет адекватного термина в человеческом языке, но эта структура вполне осуществима в языке C:

```
if (условие)
{
    инструкция или инструкции;
}
else if (условие)
{
    инструкция или инструкции;
}
else
{
    инструкция или инструкции;
}
```

Если первое *условие* ложно, выполняется следующая проверка инструкцией `else if`. Если это новое *условие* оказывается истинным, выполняются соответствующие инструкции. Если оба условия ложны, выполняются инструкции, относящиеся к последнему `else`.

Упражнение 8.13. Взяв за основу исходный текст из листинга 8.2, создайте структуру `if-if else-else`, работающую с тремя альтернативами. Первые два условия указаны в листинге 8.2, и вам надо добавить к ним еще один вариант с помощью структуры, аналогичной рассмотренной в данном разделе.

Язык программирования C не имеет ограничений на количество инструкций `else if`. Ваш код может иметь инструкцию `if`, за которой следуют, скажем, десять конструкций `else-if`, а за ними — конечная конструкция `else`. Такой подход работоспособен, хотя и не является оптимальным. Более эффективный подход описан ниже, в разделе "Выбор из множества альтернатив".

Множественное сравнение с использованием логики

Некоторые сравнения являются более сложными, чем представленные ранее, в табл. 8.1. Рассмотрим, например, такую математическую запись:

```
-5 <= x <= 5
```

В переводе на человеческий язык это означает, что `x` представляет собой значение между `-5` и `5` включительно. Это сравнение не является сравнением `if` в языке C, но его аналог можно получить, прибегнув к логическим операторам.

Построение сравнения с использованием логики

Можно выполнить два или более сравнений в одной инструкции `if`. Результаты этих сравнений обрабатываются с помощью логического оператора. Если в результате все выражение имеет значение “истинно”, выполняются инструкции, принадлежащие конструкции `if` (листинг 8.7).

Листинг 8.7. Применение логического оператора

```
#include <stdio.h>

int main()
{
    int coordinate;

    printf("Введите координату: ");
    scanf("%d",&coordinate);
    if( coordinate >= -5 && coordinate <= 5 )
    {
        puts("Достаточно близко!");
    }
    else
    {
        puts("За пределами досягаемости!");
    }
    return(0);
}
```

В строке 9 в конструкции `if` выполняются два сравнения. Это сравнение можно прочесть следующим образом: “Если значение переменной `coordinate` больше или равно `-5` и меньше или равно `5`”.

Упражнение 8.14. Создайте новый проект с использованием исходного текста из листинга 8.7. Сохраните и постройте проект, а затем выполните полученную программу несколько раз, чтобы убедиться в ее корректной работе.

Логические операторы

Операторы логического сравнения языка C приведены в табл. 8.2. Их можно использовать в сравнении `if`, когда должны быть выполнены два или более условий.

Таблица 8.2. Логические операторы

Оператор	Имя	Истинно, когда
`&&`	и	оба условия истинны
`||`	или	истинно хотя бы одно условие
!	не	условие ложно

Листинг 8.7 использует оператор `&&` для проверки выполнения обоих условий. Согласно табл. 8.2, чтобы условие в скобках считалось истинным, должны быть истинны оба сравнения.

Упражнение 8.15. Измените исходный текст из листинга 8.7 так, чтобы использовалась операция логического ИЛИ — чтобы условие было истинным, когда значение переменной `coordinate` было меньше `-5` или больше `5`.

Упражнение 8.16. Создайте новый проект, который задает вопрос пользователю, ответ на который имеет вид буквы `Y` или `N` в любом регистре. Убедитесь в корректности работы программы, если введена буква, отличная от указанных.

- ✓ Логические операции часто записываются прописными буквами: И, ИЛИ. Это делается для того, чтобы отличать их от обычных слов *и* или *или*.
- ✓ Оператор логического И имеет вид `&&`.
- ✓ Оператор логического ИЛИ имеет вид `||`.
- ✓ Оператор логического НЕ имеет вид `!`.
- ✓ Оператор логического НЕ является унарным оператором, в отличие от бинарных операторов И и ИЛИ. Он просто предшествует значению и обращает его, преобразуя ложное значение в истинное и обратно.

Конструкция выбора

Башня в исходном тексте из конструкций `if-else` может быть эффективна, но это не лучший способ выбора из нескольких вариантов. В частном случае язык программирования C предлагает более красивое решение — структуру `switch-case`.

Выбор из множества альтернатив

Структура switch-case позволяет запрограммировать на языке C выбор альтернативы на основании одного значения (листинг 8.8).

Листинг 8.8. Выбор из множества альтернатив

```
#include <stdio.h>

int main()
{
    int code;

    printf("Введите код ошибки (1-3): ");
    scanf("%d",&code);

    switch(code)
    {
        case 1:
            puts("Ошибка диска.");
            break;
        case 2:
            puts("Неверный формат.");
            break;
        case 3:
            puts("Неверное имя файла.");
            break;
```

```
        default:
            puts("Введено не 1, 2 или 3");
    }
    return(0);
}
```

Упражнение 8.17. Создайте проект с использованием исходного текста из листинга 8.8. Просто введите этот исходный текст; я опишу его позже. Сохраните и постройте проект, а затем выполните полученную программу несколько раз с разными значениями, чтобы посмотреть, как она работает.

Просмотрите исходный код в таком редакторе, в котором можно увидеть номера строк, на которые я ссылаюсь ниже.

Структура `switch-case` начинается со строки 10 инструкцией `switch`. Она вычисляет значение в круглых скобках. В отличие от инструкции `if`, `switch` получает только одно значение. В строке 10 это целочисленное значение, вводимое пользователем (в строке 8).

Часть `case` структуры располагается в фигурных скобках между строками 11 и 23. Каждая инструкция `case` содержит единственное значение, как, например, `1` в строке 12. После значения следует двоеточие.

Значения после каждого `case` сравниваются со значением, вычисленным в инструкции `switch`. Если эти значения равны, выполняются инструкции, следующие за соответствующим `case`. Если нет, программа переходит для сравнения к следующему `case`.

Ключевое слово `break` прекращает выполнение инструкций выбранной альтернативы, и управление передается инструкции за завершающей структуру `switch-case` фигурной скобкой; в листинге 8.8 это инструкция в строке 24.

После последнего сравнения структура `switch-case` выполняет инструкции, относящиеся к метке `default`, находящейся в строке 21. Эти инструкции выполняются, если не найдено ни одно соответствие значения `switch` значениям `case`. Метка `default` обязана присутствовать в структуре `switch-case`.

Упражнение 8.18. Создайте программу с помощью исходного текста, подобного исходному тексту из листинга 8.8, но в котором пользователь должен вводить буквы *A*, *B* и *C*. Если вы забыли, как вводится отдельный символ, обратитесь к главе 7, “Ввод и вывод”.

- ✓ В структуре `switch-case` выполняется сравнение значения из круглых скобок в конструкции `switch` и значений, следующих за каждым ключевым словом `case`. Если сравнение истинно, т.е. если значения совпадают, выполняются инструкции, принадлежащие данной метке `case`.
- ✓ Ключевое слово `break` используется для прекращения выполнения инструкций и выхода из структуры `switch-case`. Однако главным образом это ключевое слово используется в циклах (см. главу 9, “Циклы”).
- ✓ Не забывайте указывать ключевое слово `break` после инструкций `case`, чтобы не были выполнены остальные инструкции структуры (читайте приведенный ниже раздел “Если забыть break”).

Структура switch-case

Познакомимся теперь с самой сложной вещью в языке C подробнее. Я серьезно: вы не найдете нигде в языке большего количества правил, чем правила, касающиеся структуры `switch-case`. Вот ее скелет:

```
switch (выражение)
{
    case значение1:
        инструкции 1;
        break;
    case значение2:
        инструкции 2;
        break;
    case значение3:
        инструкции 3;
        break;
    ...
    default:
        инструкции;
}
```

Конструкция `switch` открывает структуру, заключенную между парой фигурных скобок. Эта структура должна содержать как минимум одну конструкцию `case` и конструкцию `default`.

Конструкция `switch` содержит *выражение* в скобках. Оно должно давать одно целочисленное значение. Это может быть переменная, значение, возвращенное функцией или математической операцией.

За каждым ключевым словом `case` следует непосредственное значение, а за ним — двоеточие. За двоеточием может располагаться одна или больше инструкций. Эти инструкции выполняются, если непосредственное значение, следующее за ключевым словом `case`, соответствует значению выражения в конструкции `switch`. В противном случае инструкции опускаются, и выполняется сравнение значения *выражения* со значением следующей метки `case`.

Ключевое слово `break` используется для прерывания выполнения структуры `switch-case`. В противном случае программа переходит к выполнению следующей инструкции.

Структура `switch-case` завершается конструкцией `default`. Она содержит инструкции, которые выполняются в случае несовпадения значения *выражения* со всеми значениями конструкций `case`. Если в такой ситуации никакие действия выполнять не требуется, в конструкции `default` может не быть ни одной инструкции.

Часть `case` структуры `switch-case` не должна содержать вычислений. Если вам требуются множественные сравнения, используйте структуры на основе `if-else`.

Если забыть `break`

Допустимо применение структуры `switch-case` без `break`. Такая структура может даже оказаться полезной при определенных условиях, как показано в листинге 8.9.

Листинг 8.9. План питания

```
#include <stdio.h>

int main()
{
    char choice;

    puts("План питания:");
    puts("A - завтрак, обед и ужин");
    puts("B - только обед и ужин");
    puts("C - один ужин");
    printf("Ваш выбор: ");
    scanf("%c",&choice);

    printf("Вы выбрали ");
    switch(choice)
    {
        case 'A':
            printf("завтрак, ");
        case 'B':
            printf("обед и ");
        case 'C':
            printf("ужин ");
        default:
            printf("в качестве плана питания.\n");
    }
    return(0);
}
```

Упражнение 8.19. Создайте проект с использованием исходного текста из листинга 8.9. Сохраните и постройте проект, а затем выполните полученную программу.

Упражнение 8.20. Если вы понимаете, как работает листинг 8.9, измените исходный текст из упражнения 8.18 так, чтобы в структуре `switch-case` программа могла обрабатывать как прописные, так и строчные буквы.

Странное решение ?:

Мне осталось описать еще один инструмент принятия решения. Это, пожалуй, самый загадочный из инструментов, любимый программистами, которым нравится запутывать код. Взгляните на листинг 8.10.

Листинг 8.10. Применение оператора ?:

```
#include <stdio.h>

int main()
{
    int a,b,larger;

    printf("Введите значение A: ");
    scanf("%d",&a);
    printf("Введите еще одно значение B: ");
    scanf("%d",&b);

    larger = (a > b) ? a : b;
    printf("Значение %d больше.\n",larger);
    return(0);
}
```

В частности, взгляните на строку 12, которую я повторю здесь, чтобы вы прониклись ее загадочностью:

```
larger = (a > b) ? a : b;
```

Упражнение 8.21. Создайте проект с использованием исходного текста из листинга 8.10. Сохраните и постройте проект, а затем выполните полученную программу, чтобы убедиться в работоспособности оператора `?:`.

Официально `?:` известен как *тернарный* оператор: он состоит из трех частей. После сравнения идут две части: значение, если сравнение истинно, и значение, если оно ложно:

```
результат = сравнение ? если_истинно : если_ложно;
```

Инструкция начинается со сравнения. Сравнение может быть любым, которое работает в конструкции `if`, т.е. использовать любые математические и логические операторы. Я обычно заключаю сравнение в кавычки, хотя это и не обязательно.

Если сравнение истинно, вычисляется операнд *`если_истинно`*, и его значение сохраняется в переменной `result`. В противном случае в переменной сохраняется вычисленное значение *`если_ложно`*.

Упражнение 8.22. Перепишите исходный текст из листинга 8.10 с использованием структуры `if-else`, выполняющей те же действия, что и тернарный оператор в строке 12.