[10. Выбор вариантов.](#_Toc515770098)

[10.1. Операции отношения и выражения.](#_Toc515770099)

[10.2. Понятие «истина».](#_Toc515770100)

[10.3. Условные операторы.](#_Toc515770101)

[10.3.1. Оператор if.](#_Toc515770102)

[10.3.2. Оператор if-else.](#_Toc515770103)

[10.3.3. Вложенные операторы if-else.](#_Toc515770104)

[10.3.4. Операторы if-else-if.](#_Toc515770105)

[10.3.5. Условный оператор ?.](#_Toc515770106)

[10.3.6. Оператор switch.](#_Toc515770107)

[10.3.7. Совместное использование операторов if-else-if и switch.](#_Toc515770108)

1. Выбор вариантов.

Хотите научиться создавать мощные, «интеллектуальные», универсальные и полезные программы? Тогда вам потребуется язык, обеспечивающий три основные формы управления процессом выполнения программ. Согласно теории вычислительных систем (которая является наукой о вычислительных машинах, а не наукой, создаваемой такими машинами), хороший язык должен обеспечивать реализацию следующих трех форм управления процессом выполнения программ:

* Выполнение последовательности операторов.
* Выполнение определенной последовательности операторов до тех пор, пока некоторое условие истинно.
* Использование проверки истинности условия для выбора между различными возможными способами действия.

Первая форма хорошо известна; все наши предыдущие программы представляли собой некоторую последовательность операторов. Цикл while служит одним из примеров использования второй формы; другие способы будут рассмотрены позднее. Последняя форма — выбор между различными возможными способами действия — делает программы гораздо более «интеллектуальными» и чрезвычайно увеличивает эффективность работы компьютера. Здесь мы и займемся этим вопросом.

## Операции отношения и выражения.

Операции отношения используются для сравнений. Мы уже использовали ранее некоторые из них, а сейчас приведем полный список операций отношения, применяемых при программировании на языке Си.

Операция Смысл

< меньше

< = меньше или равно

= = равно

> = больше или равно

> больше

!= не равно

Этот список довольно хорошо соответствует возможным числовым соотношениям. (Вообще говоря, числа, даже комплексные, менее сложны, чем люди (игра слов: complex — сложный и комплексный). Главное предостережение, которое мы хотим сделать, состоит в том, чтобы не использовать знак = вместо = =. В некоторых языках программирования (например, Бейсике) один и тот же знак используется и для операции присваивания, и для операции отношения «равенство», хотя они совершенно различны. С помощью операции присваивания некоторое значение присваивается переменной слева от знака равенства. В то же время с помощью операции отношения «равенство» проверяется: равно ли выражение, стоящее слева от знака, выражению справа от него. Эта операция не изменяет значения переменной в левой части, если она там присутствует.

canoes = 3; присваивает значение 3 переменной canoes

canoes = = 5 проверяет, равняется ли значение переменной canoes 5

При программировании требуется аккуратность, потому что в ряде случаев компилятор не сможет обнаружить ошибки, связанной с неправильным использованием знаков этих операций, что приведет к результатам, отличным от тех, которые вы должны были получить. Ниже мы кратко остановимся на одном примере.

|  |  |  |
| --- | --- | --- |
| **Язык** | **Операция**  **присваивания** | **Операция отношения**  **«равенство»** |
| Бейсик | = | = |
| Фортран | = | .EQ. |
| Си | = | = = |
| Паскаль | := | = |
| ПЛ/1 | = | = |
| Лого | make | = |

Операции отношения применяются при формировании условных выражений, используемых в операторах if и while. Указанные операторы проверяют, истинно или ложно данное выражение. Ниже приводятся четыре не связанные между собой оператора, содержащие условные выражения; их смысл, мы надеемся, понятен.

if (number < 6)

printf(" Ваше число слишком мало.\n");

while (ch != '$')

count++;

if (total = = 100)

printf(" Вы набрали максимум очков! \n" );

if (ch > ' M')

printf(" Отправьте этого человека по другому маршруту.\n" );

08_03

Обратите внимание, что в условных выражениях разрешается использовать также и символы, причем при сравнении их берется машинный код (который, как предполагалось вначале, является кодом ASCII). Однако использовать операции отношения для сравнения строк не разрешается: позднее мы представим средства работы со строками.

Операции отношения можно использовать также при работе с числами с плавающей точкой. Однако при сравнениях этих чисел мы рекомендуем вам ограничиться лишь операциями < и >, так как ошибки округления могут привести к тому, что числа окажутся неравными, хотя по логике они должны быть равны. Рассмотрим подобную ситуацию на примере десятичных чисел. Очевидно, произведение 3 и 1/3 равно 1.0, но если мы представим 1/3 в виде 6-разрядной десятичной дроби, то произведение будет равно 0.999999, что не равняется в точности 1.

Каждое условное выражение проверяется: «истинно» ли оно или «ложно». При этом возникает интересный вопрос.

## Понятие «истина».

Мы можем ответить на этот вечный вопрос, по крайней мере, так, как он решен в языке Си. Напомним, во-первых, что выражение в Си всегда имеет значение. Это утверждение остается верным даже для условных выражений, как показывает пример, приведенный ниже. В нем определяются значения двух условных выражений, одно из которых оказывается истинным, а второе — ложным.

/\* истина и ложь \*/

main()

{

int tr, fa;

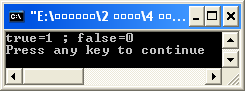
tr= (10 > 2); /\* отношение истинно \*/

fa=(10 == 2); /\* отношение ложно \*/

printf("true = %d; false = %d\n", tr, fa);

}

В данном примере значения двух условных выражений присваиваются двум переменным. Чтобы не запутать читателя, мы присвоили переменной true значение выражения, которое оказывается истинным, а переменной false — значение выражения, которое оказывается ложным. При выполнении программы получим следующий простой результат:



Вот как! Оказывается, в языке Си значение «истина» — это 1, а «ложь» — 0. Мы можем это легко проверить, выполнив программу, приведенную ниже.

/\* проверка истинности \*/

main()

{

if (1)

printf("1 соответствует истине\n");

else

printf("1 не соответствует истине\n");

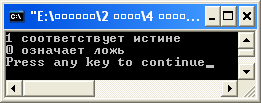
if(0)

printf("0 не означает ложь\n");

else

printf("0 означает ложь\n");

}



Мы скажем, что 1 должна рассматриваться как истинное утверждение, а 0 — как ложное. Если наше мнение верно, то в первом операторе if должна выполниться первая ветвь (ветвь if, а во втором операторе if — вторая (ветвь else). Попробуйте запустить эту программу, чтобы узнать, правы ли мы.

Итак, чему же все-таки соответствует истина? Мы можем использовать 1 и 0 в качестве проверочных значений для оператора if. Спрашивается, можем ли мы использовать другие числа. Если да, то что при этом происходит? Давайте проведем эксперимент.

/\* if — тест \*/

main()

{

if (200)

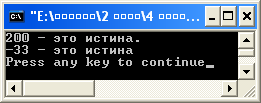
printf("200 — это истина.\n");

if (-33)

printf(" - 33 — это истина\n" );

}

Результаты выглядят так:



Очевидно, в языке Си числа 200 и-33 тоже рассматриваются как «истина». И действительно, все ненулевые величины принимаются в качестве «истины» и только 0 — как «ложь». В языке Си понятие истины оказывается чрезвычайно растяжимым!

Многие программисты обычно пользуются этим определением истинности. Например, строку

if(goats ! = 0)

можно заменить такой

if(goats)

поскольку выражение (goats ! = 0) и выражение (goats) оба примут значение 0, или «ложь», только в том случае, если значение переменной goats равно 0. Мы думаем, что смысл второй формы записи менее очевиден, чем первой. Однако в результате компиляции она оказывается более эффективной, так как при реализации программы требует меньшего числа машинных операций.

Осложнения с понятием «истина». Растяжимость понятия истина в языке Си может привести и к неприятностям. Рассмотрим следующую программу:

/\* занятость \*/

main()

{

int age = 20;

while (age ++ <= 65)

{

if ((age % 20) ==0) /\* делится ли возраст на 20? \*/

printf(" Вам %d. Поздравляем с повышением!\n" , age);

if (age = 65)

printf(" Вам %d. Преподносим Вам золотые часы.\n" , age);

}

}

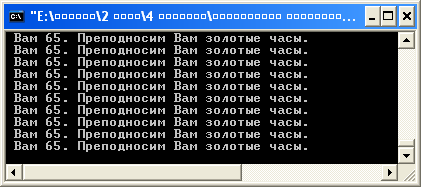
С первого взгляда вам может показаться, что результат работы программы будет выглядеть, например, так:

Вам 40.Поздравляем с повышением!

Вам 60.Поздравляем с повышением!

Вам 65.Преподносим Вам золотые часы.

На самом деле, однако, выход будет таким:



и т. д. — до бесконечности.

В чем дело? Это произошло не только потому, что мы плохо спроектировали программу, но и потому, что мы забыли свои собственные предостережения и использовали выражение

if (age = 65)

вместо

if (age = = 65)

Последствия ошибки оказались катастрофическими. Когда в процессе выполнения программа достигает указанного оператора, она видит выражение (аgе = 65). Это выражение, включающее в себя операцию присваивания, имеет значение, которое совпадает со значением переменной, расположенной слева от знака, т. е. с 65 (в данном случае). Поскольку 65 не равно нулю, выражение считается «истинным» и выполняется команда вывода на печать. Затем, когда в программе происходит передача управления на команду проверки условия в цикле while, значение переменной аде по-прежнему равно 65, что меньше или равно 65. Условие оказывается истинным, величина аgе увеличивается до 66 (ввиду наличия операции увеличения ++ в постфиксной форме), и тело цикла выполняется еще раз. Прекратится ли его выполнение на следующем шаге? Должно было бы, поскольку величина аде теперь больше, чем 65. Но когда программа опять попадает на наш ошибочный оператор if, переменная аде снова получит значение 65! В результате сообщение будет напечатано еще раз, затем тело цикла выполнится еще раз, и т.д. — до бесконечности. (Конечно, если вы в конце концов не захотите остановить программу.)

Подводя итоги, можно сказать, что операции отношения используются для образования условных выражений. Условное выражение имеет значение «1», когда оно истинно, и «0», если оно ложно. В операторах (таких, как while и if), где обычно используются условные выражения для задания проверяемых условий, могут применяться любые выражения, причем ненулевое значение является «истиной», а нуль — «ложью».

## Условные операторы.

В языке С имеется четыре основных условных оператора: if, if-else,? и switch. Перед тем, как обсудить их отдельно, необходимо отметить одно общее правило.

Большинство условных операторов можно использовать для выборочного выполнения или одной программной строки, или нескольких логически связанных строк (называемых блоком). Если условный оператор связан только с одной строкой исполняемого кода, эту строку заключать в фигурные скобки ({ }) не нужно. Если же условный оператор относится к блоку выполняемых операторов, то для связи блока с проверяемым условием фигурные скобки необходимы. По этой причине в операторе switch необходимы открывающая и закрывающая фигурные скобки.

### Оператор if.

Оператор if используется для условного выполнения фрагмента кода. Простейший вид оператора if следующий:

if (выражение)

действие\_если\_истина;

Обратите внимание на то, что выражение должно заключаться в круглые скобки. Для выполнения оператора if необходимо вычислить выражение и определить его значение: истина или ложь. Если выражение истинно, выполняется действие\_если\_истина и программа продолжается, начиная с оператора, следующего за этим действием. Если выражение ложно, действие\_если\_истина не выполняется, а выполняется оператор, следующий за невыполненным действием. Например: следующий фрагмент программы напечатает сообщение "Have a great day!" только в случае, если переменная ioutside\_temp больше или равна 72:

if(ioutside\_temp >= 72)

printf("Have a great day!");

Оператор if, связанный с блоком выполняемых операторов, выглядит следующим образом:

if (выражение)

{

действие\_если\_истина1;

действие\_если\_истина2;

действие\_если\_истинаЗ;

действие\_если\_истина4;

}

Синтаксис требует, чтобы все связанные операторы были заключены в фигурные скобки ({ }), и каждый оператор в блоке заканчивался точкой с запятой (;).

Начнем с очень простого примера. Мы уже видели, как нужно написать программу, подсчитывающую число символов в файле. Предположим, вместо символов мы хотим подсчитать строки. Это можно сделать путем счета числа символов «новая строка» в файле. Ниже приводится соответствующая программа:

/\* подсчет строк \*/

/\* X26.C \*/

#include <stdio.h>

void main(void)

{

int ch;

int linecount = 0;

printf("Exit <CTRL/Z>\n");

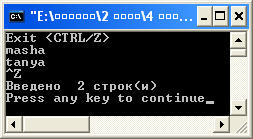
while((ch=getchar()) != EOF)

if(ch =='\n')

linecount++;

printf("Введено %d строк(и)\n",linecount);

}



«Сердцевиной» программы является оператор

if (ch = = ' \n')

linecount++;

Этот «оператор if» служит указанием компьютеру увеличить значение переменной linecount на 1, если только что прочитанный символ (содержимое переменной ch) представляет собой символ «новая строка». Знак == не является опечаткой; его смысл выражается словами «равняется». Не путайте эту операцию с операцией присваивания (=).

Что происходит в случае, когда значение переменной ch не является символом «новая строка»? Тогда в цикле while производится чтение следующего символа.

Оператор if, который мы только что применили в пporpaмме, считается одиночным оператором, начинающимся от ключевого слова if и завершающимся символом «точка с запятой». Вот почему мы не использовали фигурные скобки, чтобы отметить начало и конец тела цикла while.

Совсем несложно усовершенствовать программу так, чтобы она подсчитывала символы и строки одновременно; давайте попробуем это сделать.

/\* подсчет числа строк и символов \*/

#include <stdio.h>

void main ()

{

int ch;

int linecount = 0;

int charcount = 0;

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

{

charcount ++ ;

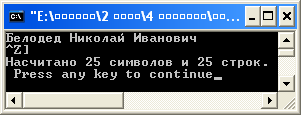
if(ch = '\n')

linecount ++ ;

}

printf(" Насчитано %d символов и %d строк. \n ", charcount, linecount);

}



Теперь в цикл while входят два оператора; поэтому мы использовали фигурные скобки, чтобы отметить начало и конец тела цикла.

Мы вызываем скомпилированную программу Icc и используем операцию переключения, чтобы подсчитать число символов и строк в файле с именем chow.

lcc < chow

Я насчитала: 8539 символов и 231 строки.

Следующий шаг в «развитии» этой программы — придание ей способности подсчитывать число слов. Это лишь немного труднее того, что мы делали до сих пор, но сначала нам нужно изучить ряд дополнительных вопросов, связанных с использованием операторов if.

Далее следует пример составного оператора if:

/\*07IF.C

Программа на С, иллюстрирующая оператор if\*/

#include "stdafx.h"

#include <stdio.h>

#include <conio.h>

main()

{

int inum\_As, inum\_Bs, inum\_Cs, ch;

float fGPA;

/\* Укажите число курсов, получивших оценку ... \*/

printf ("\nEnter number of courses receiving a grade of A: ") ;

scanf("%d",&inum\_As) ;

printf("\nEnter number of courses receiving a grade of B: ");

scanf ("%d", &inum\_Bs) ;

printf("\nEnter number of courses receiving a grade of C: ");

scanf("%d",&inum\_Cs);

fGPA = (inum\_As \* 4 + inum\_Bs \* 3 + inum\_Cs \* 2)/

(float) (inum\_As + inum\_Bs + inum\_Cs);

/\* Ваш общий коэффициент \*/

printf("\nYour overall GPA is: %5.2f\n",fGPA);

if(fGPA >= 3.5) {

printf ("\n CONGRATULATIONS !\n"); /\* Поздравления! \*/

printf("You are on the President's list."); /\* Вы — в президентском списке. \*/

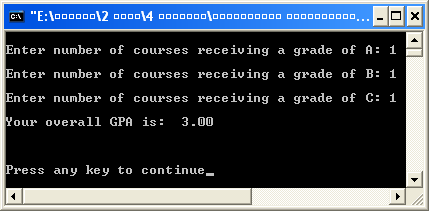
}

printf ("\n\nPress any key to finish\n"); /\* Конец работы \*/

ch = \_getch();

return (0);

}



В этом примере, если переменная fGPA больше или равна 3.5, после печати значения fGPA выводится приветственное сообщение. Вычисленное значение fGPA печатается вне зависимости от выполнения блока внутри оператора if.

### Оператор if-else.

Оператор if-else нужен для того, чтобы программа выполнила два разных действия в зависимости от истинности некоторого выражения. В простейшем случае оператор if-else выглядит следующим образом:

if (выражение)

действие\_если\_истина;

else

действие\_если\_ложь;

В этом операторе, если выражение истинно, выполняется действие\_если\_истина; если же выражение ложно, выполняется действие\_если\_ложъ. Рассмотрим пример:

if(ckeypressed == UP)

iy\_pixel\_coord++;

else

iy\_pixel\_coord--;

В этом примере текущая горизонтальная координата увеличивается или уменьшается в зависимости от значения, хранящегося в символьной переменной ckeypressed.

Любое из выражений — выражение\_если\_истина, выражение\_если\_ложь, или оба вместе — могут быть составным оператором или блоком, заключенным в фигурные скобки. Синтаксис этих трех вариантов очень прост:

if (выражение) {

действие\_если\_истина1;

действие\_если\_истина2;

действие\_если\_истина3;

}

else

действие\_если\_ложь;

if (выражение) {

действие\_если\_истина;

else {

действие\_если\_ложь1;

действие\_если\_ложь2;

действие\_если\_ложь3;

}

if (выражение) {

действие\_если\_истина1;

действие\_если\_истина2;

действие\_если\_истина3;

}

else {

действие\_если\_ложь1;

действие\_если\_ложь2;

действие\_если\_ложь3;

}

Необходимо только помнить, что где бы ни стоял блок выполняемых операторов, не нужно ставить точку с запятой после закрывающей фигурной скобки ( } ).

В следующей программе на С используется оператор if-else, в котором часть if является составным блоком:

/\*07CMPIF.C

Программа на С, иллюстрирующая использование составного оператора if-else\*/

#include "stdafx.h"

#include <stdio.h>

#include <conio.h>

main()

{

char c;

int ihow\_many, i, imore;

imore=1;

while(imore == 1)

{

/\* Введите название изделия \*/

printf("Please enter the product name: ");

if(scanf("%c",&c) != EOF) {

while(c != '\n')

{

printf("%c",c) ;

scanf("%c",&c);

}

/\* сколько закуплено? \*/

printf("s purchased? ");

scanf("%d",&ihow\_many);

scanf("%c",&c);

for(i = 1;i <= ihow\_many; i++)

printf ("\*");

printf("\n");

}

else

imore=0;

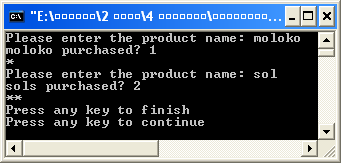
}

printf ("\n\nPress any key to finish\n");/\* Конец работы \*/

\_getch();

return(0);

}



Программа запрашивает у пользователя название изделия и, если пользователь не ввел символ ^Z(EOF), вводит по буквам название изделия и печатает эхо-отображение информации на следующей строке. Строка "s purchased" добавляется к названию изделия, и запрашивается количество проданных единиц. И наконец, в цикле for печатается соответствующее число звездочек (\*). Если бы пользователь ввел символ ^Z, то часть if оператора if-else игнорировалась бы и выполнилась бы часть else, где флаг imore устанавливается в ноль, в результате чего программа прекращается.

Раньше мы привели очень простую программу шифровки сообщений, которая заменяла каждый символ следующим по порядку в таблице ASCII. К сожалению, она заменила даже символ «новая строка», что привело к объединению нескольких строк в одну. Можно исключить эту проблему, написав программу, реализующую простой выбор: если символ — «новая строка», то оставить его без изменений, в противном случае преобразовать его. Вот как это можно запрограммировать на языке Си:

/\* код1 \*/

#include <stdio.h>

void main()

{

char ch;

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

{

if (ch = '\n' ) /\* оставить символ \*/

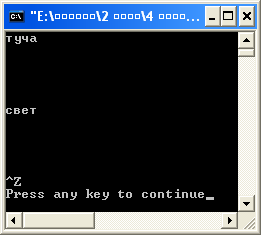
putchar(ch); /\* "новая строка" неизменным \*/

else

putchar(ch + 1); /\*заменить остальные символы \*/

}

}



В прошлый раз был использован файл, содержащий следующий текст:

Good spelling is an aid

to clear writing.

Его использование в качестве теста для нашей новой программы приводит к результату:

!!!!!Hppe!tgfmmjoh!jt!bo!bje

up!hppe!xsjujoh/

Ура! Она работает! Между прочим, совсем несложно написать и программу дешифровки. Скопируйте для этого программу код1, но только замените выражение (ch + 1) на (ch — 1).

Вы обратили внимание на общий вид оператора if-else. Он выглядит следующим образом:

if(выражение)

оператор

else

оператор

Если выражение истинно, то выполняется первый оператор, а если ложно, то выполняется оператор, следующий за ключевым словом else. Операторы могут быть простыми или составными. Правила языка Си не требуют отступа в строке, но это стало обычной практикой. Отступ позволяет с первого взгляда заметить те операторы, выполнение которых зависит от проверяемого условия.

Простая конструкция if позволяет нам выбирать: выполнить или нет некоторое действие; конструкция же if-else дает возможность выбрать одно из двух действий. Что делать, если нам требуется осуществить выбор из большого числа вариантов?

### Вложенные операторы if-else.

Когда используются вложенные операторы if, необходимо следить за тем, какому if соответствует конкретное действие else. Взгляните на следующий пример — сможете ли вы предсказать результат:

if(iout\_side\_temp < 50) /\* Если температура на улице ... \*/

if(iout\_side\_temp < 30) printf("Wear the down jacket!"); /\*Надеть куртку! \*/

else printf("Parka will do."); /\* Достаточно одеть парку. \*/

Листинг намеренно не выровнен, чтобы не было зрительных подсказок на вопрос: какой оператор к какому if относится. Что получится, если переменная iout\_side\_temp равна 55? Будет ли напечатано сообщение "Parka will do."?

Ответ — нет. В этом примере действие else связано со вторым выражением if. Это вызвано тем, что в С else связывается с ближайшим несвязанным if. Для облегчения отладки подобных фрагментов компилятор С написан так, что каждый else связывается с ближайшим if, еще не имеющим части else.

Конечно же, соответствующее оформление программы всегда проясняет ситуацию:

if(iout\_side\_temp < 50)

if(iout\_side\_temp < 30) printf("Wear the down jacket!");

else

printf("Parka will do.");

Возможен также следующий альтернативный вариант листинга:

if(iout\_side\_temp < 50)

if(iout\_side\_temp < 30)

printf("Wear the down jacket!");

else

printf("Parka will do.");

Любое написанное вами приложение много выиграет от использования в программе одного из этих двух стилей.

Сможете ли вы разобраться со следующим примером:

if(условное\_выражение1)

if(условное\_выражение2)

действие\_если\_условие2\_истина;

else

действие\_если\_условие1\_ложь;

Вы можете подумать, что это — еще один пример того, что обсуждалось выше. Это так. Но что, если вы действительно хотите, чтобы дей-ствие\_если\_условие1\_ложь было связано с условием 1, а не с условием2?. Ведь во всех предыдущих примерах действие else связывалось со вторым, то есть ближайшим, if. (Кстати: многие программисты тратили массу времени, отлаживая подобную программу. Они работали, следуя логике, описанной в предыдущем примере; к сожалению, компилятор не реагирует на "стиль" вашего мышления.)

Исправить ситуацию можно, если использовать фигурные скобки:

if(условное\_выражение1); {

if(условное\_выражение2);

действие\_если\_условие2\_истина ;

}

else

действие\_если\_ условие 1\_ложь;

Проблема решена; для этого условное\_выражение2 и связанное с ним действие\_если\_условие2 истина объединены в блок, который выполняется при истинном значении условного\_выражения1. Теперь понятно, что действие если\_условие1\_ложь связано с частью else условного выражения1.

На практике часто нам приходится осуществлять выбор более, чем из двух вариантов. Чтобы учесть это, мы можем расширить структуру if-else конструкцией else-if. Рассмотрим конкретный пример. Расценки, назначаемые за коммунальные услуги некоторыми компаниями зависят от количества потребляемой энергии. Приведем расценки, установленные за пользование электроэнергией:

Первые 240 кВт/ч: 0.05418 долл. за кВт/ч

Следующие 300 кВт/ч: 0.07047 долл. за кВт/ч

Свыше 540 кВт/ч: 0.09164 долл. за кВт/ч

Если вас занимает этот вопрос, мы могли бы подготовить программу, вычисляющую стоимость потребляемой энергии. Приведем пример программы, являющейся первым шагом, сделанным в этом направлении:

08_01

# include <stdio.h>

# include <math.h>

/\* счет за электроэнергию \*/

/\* вычисляет плату за электроэнергию \*/

#define RATE1 0.05418 /\* тариф за первые 240 кВт/ч \*/

#define RATE2 0.07047/\* тариф за следующие 300 кВт/ч \*/

#define RATE3 0.09164 /\* тариф за потребление свыше 540 кВт/ч \*/

#define BASE1 13.00 /\* плата за первые 240 кВт/ч \*/

#define BASE2 34.14 /\* плата за первые 540 кВт/ч \*/

#define BREAK1 240.0 /\* величина, определяющая первое условие ветвления \*/

#define BREAK2 540.0 /\* величина, определяющая второе условие ветвления \*/

void main()

{

float kwh; /\* количество использованных кВт/ч \*/

float bill; /\* плата \*/

printf("Количество КВт/ч. \n" );

scanf(" %f" , &kwh);

if (kwh < BREAK1)

bill = RATE1\*kwh;

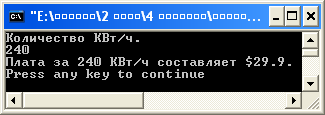
else if (kwh < BREAK2) /\* количество кВт/ч между 240 и 540 \*/

bill = BASE1 + RATE2\*kwh;

else bill = BASE2 + RATE3 \*kwh; /\* количество кВт/ч свыше 540 \*/

printf(" Плата за %.1f КВт/ч составляет $%1.2f.\n" , kwh, bill);

}



Для обозначения тарифов были использованы символические константы, которые поэтому оказались собранными в одном месте. Если электрическая компания изменит свои расценки (а это может случиться), то такое расположение констант не позволит нам забыть скорректировать какую-нибудь из них. Мы задали в символическом виде и константы, соответствующие граничным значениям потребляемой мощности. Они также подвержены изменениям. Управляющая логика программы реализуется путем простого выбора одной из трех расчетных формул в зависимости от значения переменной kwh, что иллюстрируется на рисунке.

08_02

Мы хотим подчеркнуть, что программа в процессе выполнения может достичь первого употребления else только в том случае, если величина переменной kwh больше или равна 240. Поэтому строка else if (kwh < BREAK2) эквивалентна требованию, чтобы значение kwh было заключено между 240 и 540; это мы и указали в комментарии к программе. Совершенно аналогично последнее употребление else может быть достигнуто, только если значение kwh больше или равно 540. И наконец, отметим, что константы ВASE1 и BASE2 представляют собой величину платы за первые 240 или 540 кВт/ч электроэнергии соответственно. Поэтому требуется только прибавить дополнительную плату за количество потребляемой электроэнергии, превышающее эти величины.

Фактически конструкция else-if является видоизмененным способом задания условного оператора, с которым мы познакомились раньше. Ядро нашей программы представляет собой другую форму записи следующей последовательности операторов:

if (kwh < BREAK1)

bill = RATE1\*kwh;

else

if (kwh < BREAK2)

bill = BASE1 + RATE2\*kwh;

else

bill = BASE2 + RATE3 \*kwh;

Отсюда видно, что программа состоит из оператора if-else, для которого часть else представляет собой другой оператор if-else.

Про второй оператор If-else говорят, что он «вложен» в первый. (Между прочим, вся структура if-else считается одним оператором. Вот почему мы не должны заключать вложенную конструкцию if-else в фигурные скобки.)

Эти две формы записи являются абсолютно эквивалентными. Единственное отличие — дополнительные пробелы и символы «новая строка», но они игнорируются компилятором. Тем не менее первая форма записи предпочтительнее, поскольку она более четко показывает, что мы осуществляем выбор из трех возможностей. Кроме того, она облегчает просмотр программы и понимание семантики каждого варианта. Применяйте форму записи, использующую вложение операторов там, где это необходимо — например когда требуется проверить значения двух разных величин или (в нашем случае) если бы была установлена 10%-ная дополнительная плата за потребление энергии свыше 540 кВт/ч только в летние месяцы.

В одном операторе можно использовать столько конструкций else-if, сколько нужно, что иллюстрируется приведенным ниже фрагментом:

if (score < 1000)

bonus = 0;

else if (score < 1500)

bonus = 1;

else if (score < 2000)

bonus = 2;

else if (score < 2500)

bonus = 4;

else

bonus=6;

(Этот фрагмент мог быть взят из игровой программы, где переменная bonus представляет собой количество дополнительных «фотонных бомб» или «питательных гранул», получаемых игроком для следующей партии.)

### Операторы if-else-if.

Комбинация операторов if-else-if часто используется для выполнения многочисленных последовательных сравнений. В общем виде они выглядят следующим образом:

if(выражение 1)

действие\_если\_условие1\_истина;

else if(выражение2)

действие\_если\_условие2\_истина;

else if(выражение3)

действие\_если\_условиеЗ\_истина;

Каждое действие может быть составным блоком в фигурных скобках (причем после закрывающей фигурной скобки точка с запятой не ставится). Такая управляющая логическая структура вычисляет каждое выражение до тех пор, пока не найдет истинное. Когда это происходит, все оставшиеся проверочные условия опускаются. В предыдущем примере, если ни одно из выражений не дает значения "истина", никаких действий не выполняется.

Взгляните на следующий пример и попробуйте определить результат:

if(выражение 1)

действие\_если\_условие1\_истина;

else if(выражение2)

действие\_если\_условие2\_истина;

else if(выражение3)

действие\_если\_условиеЗ\_истина;

else

действие\_по\_умолчанию;

В отличие от предыдущего примера, эта комбинация операторов if-else-if всегда выполняет какое-нибудь действие. Если ни одно из выражений не истинно, выполняется часть else, а именно действие\_по\_умолчанию. Следующий программный пример проверяет значение переменной econvert\_to для того, чтобы определить необходимый тип преобразования. Если запрошенное значение econvert\_to не соответствует ни одному из предусмотренных преобразований, то программа печатает соответствующее сообщение.

if(econvert\_to == YARDS) /\* преобразовывать в ярды \*/

fconverted\_value = length / 3;

else if(econvert\_to == INCHES) /\* ... в дюймы \*/

fconverted\_value = length \* 12;

else if(econvert\_to == CENTIMETERS) /\* ... в сантиметры \*/

fconverted\_value = length \* 12 \* 2.54;

else if(econvert\_to = METERS) /\* ... в метры \*/

fconverted\_value = (length \* 12 \* 2.54)/100;

else

printf("No conversion required"); /\* преобразование не требуется \*/

Когда у вас в программе имеется несколько конструкций if и else, каким образом компилятор решает, какому оператору if соответствует какой оператор else? Рассмотрим, например, фрагмент программы

if (number > 6)

if (number < 12)

printf (" Вы закончили!\n" );

else

printf(" Простите, вы потеряли ход!\n");

В каком случае фраза «Простите, вы потеряли ход!» будет напечатана? Когда значение переменной number меньше или равно 6, или когда оно больше 12? Другими словами, чему соответствует else: первому if или второму?

Ответ выглядит так: else соответствует второму if, т. е. при выполнении программы результаты будут такими:

|  |  |
| --- | --- |
| **Число** | **Результат** |
| 5 | Нет |
| 10 | Вы закончили! |
| 15 | Простите, вы потеряли ход! |

Существует правило, которое гласит, что else соответствует ближайшему if, кроме тех случаев, когда имеются фигурные скобки. Мы сознательно записали этот фрагмент так, как будто else соответствует первому if, но вспомните, что компилятор не обращает внимания на отступы в строках. Если мы действительно хотим, чтобы else соответствовал первому if, необходимо данный фрагмент оформить следующим образом:

if (number > 6)

{

if (number < 12)

printf(" Вы закончили!\п" );

}

else

printf(" Простите, вы потеряли ход!\n" );

Теперь результат может выглядеть так:

|  |  |
| --- | --- |
| **Число** | **Результат** |
| 5 | Простите, вы потеряли ход! |
| 10 | Вы закончили! |
| 15 | Нет |

Резюме: использование операторов if для организации выбора из нескольких вариантов.

**Общие замечания**

В каждой из последующих форм оператором может быть либо простой оператор, либо составной. Выражение «истинно» в обобщенном смысле, если его значение — ненулевая величина.

**форма записи 1:**

if (выражение)

оператор

Оператор выполняется, если выражение истинно.

**форма записи 2:**

if (выражение)

оператор1

else

оператор2

Если выражение истинно, выполняется оператор1, в противном случае — оператор 2.

**форма записи 3:**

if (выражение1)

оператор1

else if (выражение 2)

оператор2

else

операторЗ

Если выражение1 истинно, выполняется оператор1. Если выражение1 ложно, но выражение2 истинно, выполняется оператор2. В случае когда оба выражения ложны, выполняется операторЗ.

Пример

if (legs == 4) /\* legs — ноги \*/

printf(" Это, возможно, лошадь\n" );

else if (legs > 4)

printf(" Это не лошадь. \n" );

else /\* случай, когда legs < 4 \*/

{

legs++;

printf(" Теперь животное имеет еще одну ногу.\n");

}

### Условный оператор ?.

Условный оператор ? позволяет кратко записать условие проверки. Соответствующие действия выполняются в зависимости от вычисленного значения выражение\_условие: истина или ложь. Этот оператор можно использовать вместо эквивалентного оператора if-else. Условный оператор имеет следующий синтаксис:

Выражение\_условие ? действие\_если\_истина : действие\_если\_ложь;

Оператор ? иногда называют троичным оператором, так как он требует трех операндов. Рассмотрим такой оператор:

if(fvalue >= 0.0)

fvalue = fvalue;

else

fvalue = -fvalue;

Этот оператор можно записать при помощи условного оператора ?:

fvalue= (fvalue >= 0.0) ? fvalue : — fvalue;

Оба оператора определяют абсолютное значение fvalue. Приоритет условного оператора ниже любого другого, используемого в выражении; поэтому в данном примере круглые скобки не нужны. Тем не менее, скобки часто используются для лучшего восприятия программы.

В следующей программе на C++ оператор ? используется для сложного форматирования программного вывода:

// 07CONDIT.CPP

// Программа на C++, в которой используется условный оператор ?

//

#include "stdafx.h"

#include <iostream>

#include <conio.h>

#include <math.h> // для определения макроса abs

using namespace std;

main ()

{

float fbalance, fpayment;

// Введите размер вашей ссуды

cout <<"Enter your loan balance: ";

cin >> fbalance;

// Укажите, сколько выплачено по ссуде

cout << "\nEnter your loan payment amount: ";

cin >> fpayment;

// В настоящий момент вы

cout << "\n\nYou have ";

// переплатили / выплатили

cout << ((fpayment > fbalance) ? "overpaid by $" : "paid $");

cout << ((fpayment > fbalance) ? abs(fbalance - fpayment) : fpayment);

// no вашей ссуде в ... долларов

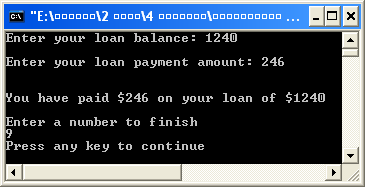
cout << " on your loan of $" << fbalance << ".";

cout << "\n\nEnter a number to finish\n";/\* Конец работы ввод любого числа \*/

cin >> fpayment;

return(0);

}



Первый условный оператор используется в программе в операторе cout для определения строки, выводимой на печать: "overpaid by $" или "paid $". В следующем условном операторе вычисляется и печатается соответствующее значение денежного эквивалента.

### Оператор switch.

Часто необходимо сравнить некоторую переменную или выражение с несколькими значениями. Для этого можно использовать либо вложенные операторы if-else-if, либо — оператор switch. Однако, будьте осторожны: в отличие от операторов-переключателей других языков высокого уровня (например, оператора case языка Паскаль), в С оператор switch имеет несколько особенностей. Синтаксис оператора switch следующий:

switch (общее\_выражение) {

case константа1:

операторы1;

break;

case константа2:

операторы2;

break;

…

…

…

case константа N:

операторыN;

break;

default: операторы;

}

Дополнительный оператор, на который надо обратить особое внимание, это — оператор break. Если бы этот пример был написан на Паскале, то при условии, что коснстанта1 равна конструкции общее\_ выражение, выполнились бы операторы1, и программа перешла бы к оператору, следующему за последним оператором case (после закрывающей фигурной скобки).

В С ситуация иная. В предыдущем примере, если удалить оператор break из раздела константа1, то получится, будто опять произошло совпадение с константой и далее будут выполняться операторы2. Именно оператор break заставляет программу пропустить оставшуюся часть операторов, входящих в switch. Рассмотрим несколько примеров.

Фрагмент программы с оператором if-else-if:

if(emove == SMALL\_CHANGE\_UP)

fycoord = 5;

else if(emove == SMALL\_CHANGE\_DOWN)

fycoord = -5;

else if(emove == LARGE\_CHANGE\_UP)

fycoord = 10;

else

fycoord = -10;

Можно переписать этот фрагмент, используя оператор switch:

switch(emove) {

case SMALL\_CHANGE\_UP:

fycoord = 5;

break;

case SMALL\_CHANGE\_DOWN:

fycoord = -5;

break;

case LARGE\_CHANGE\_UP:

fycoord = 10;

break;

default:

fycoord = -10;

}

В этом примере значение emove последовательно сравнивается с очередным значением case. Если происходит совпадение, переменная fycoord получает соответствующее значение. Затем выполняется оператор break, и оставшиеся операторы, входящие в switch, пропускаются. Если же совпадения не случается, выполняется присваивание по умолчанию (default: fycoord=-10). Так как это последняя опция в операторе switch, нет необходимости вставлять break. Ветвь default в операторе switch необязательна.

Правильно расположить break в операторе switch оказывается очень полезным. Взгляните на следующий пример:

/\*07SWITCH.C

Программа на С, иллюстрирующая возможность ветвлений и переходов

в операторе switch\*/

#include "stdafx.h"

main ()

{

char c='a';

int ivowelct=0, iconstantct=0;

switch(c) {

case 'a':

case 'A':

case 'e':

case 'E':

case 'i':

case 'I':

case 'o':

case 'O':

case 'u':

case 'U': ivowelct++;

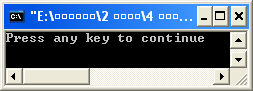
break;

default : iconstantct++;

}

return(0);

}



Эта программа на практике иллюстрирует две возможности оператора switch: наличие множества различных условий проверки, для которых выполняется одно и то же действие, и возможность пропуска операторов.

Некоторые другие языки высокого уровня имеют собственные операторы множественного выбора (операторы case — в Паскале и select — в PL/1), позволяющие включать в одну строку выбора несколько проверяемых значений и получать один и тот же результат. Однако, в С каждое проверяемое значение требует отдельного оператора case. Но обратите внимание, что в приведенном примере аналогичный эффект достигается путем удаления оператора break, в результате чего производится проверка всех гласных букв. Если переменная «с» содержит согласную букву, выполняются все проверки на гласную и происходит переход на оператор default.

В следующем примере на С показано, как использовать оператор switch для вызова соответствующей функции:

/\*07FNSWTH.C

Программа на С, иллюстрирующая действие оператора switch\*/

#include "stdafx.h"

#include <stdio.h>

#define QUIT 0

#define BLANK ' '

double fadd(float fx,float fy) ;

double fsub(float fx,float fy);

double fmul(float fx, float fy);

double fdiv(float fx,float fy);

main ()

{

float fx,fy;

char cblank, coperator = BLANK;

while (coperator != QUIT){

/\* Введите выражение (а (операция) b)\*/

printf("\nPlease enter an expression (a (operator) b): ");

scanf("%f%c%f", &fx, &coperator, &fy);

switch (coperator) { /\* ответ = \*/

case '+': printf("answer = %8.2f\n", fadd(fx,fy));

break;

case '-': printf("answer = %8.2f\n", fsub(fx,fy));

break;

case '\*': printf("answer = %8.2f\n", fmul(fx,fy));

break;

case '/': printf("answer = %8.2f\n", fdiv(fx,fy));

break;

case 'x': coperator = QUIT;

break; /\* Операция не реализована \*/

default : printf("\nOperator not implemented");

}

}

return(0) ;

}

double fadd(float fx,float fy)

{return(fx + fy);}

double fsub(float fx,float fy)

{return(fx - fy);}

double fmul(float fx,float fy)

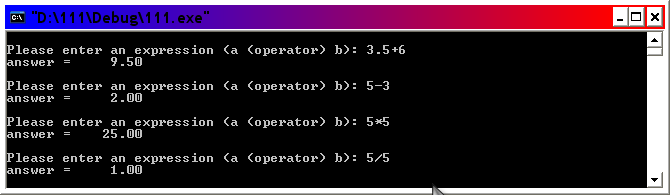
{return(fx \* fy);}

double fdiv(float fx,float fy)

{

return(fx / fy);

}



Использование оператора switch в этом примере очень эффективно, хотя вызов функций несколько усложнен. После того, как пользователь ввел некоторое выражение, например, 10+10 или 23\*15, переменная coperator (действие) анализируется в теле оператора switch и определяется вызываемая функция. Особенно интересна последняя опция, где переменная coperator сравнивается с "х", и оператор default.

Если пользователь вводит выражение с операцией "х", то переменной coperator присваивается значение QUIT, выполняется оператор break, a оператор по умолчанию (default) printf() пропускается. Однако, если пользователь ввел неизвестную операцию (например, %), то выполняется только оператор default, который печатает сообщение о том, что операция coperator не реализована.

Следующая программа на C++ иллюстрирует схожесть операторов switch в С и C++: //

// 07CALNDR.CPP

// Программа на C++, в которой оператор switch используется для

// печати календаря на год

//

#include "stdafx.h"

#include <iostream>

#include <conio.h>

//#include <iostream.h>

using namespace std;

main()

{

int jan\_1\_start\_day,num\_days\_per\_month, width,

month,date,leap\_year\_flag; // Укажите день недели 1 января

cout << "Please enter January 1's starting day;\n";

// 0 — указывает, что 1 января — воскресенье

cout << "\nA 0 indicates January 1 is on a Sunday,";

// 1 — указывает, что 1 января — понедельник

cout << "\nA 1 indicates January 1 is on a Monday,";

// 2 — указывает, что 1 января — вторник и т.д.

cout << "\nA 2 indicates January 1 is on a Tuesday, etc: ";

cin >> jan\_1\_start\_day;

// Укажите, на какой год вы хотите составить календарь:

cout << "\nEnter the year you want the calendar generated: ";

cin >> leap\_year\_flag;

// Календарь на год

cout << "\n\n The calendar for the year " << leap\_year\_flag; leap\_year\_flag=leap\_year\_flag % 4;

// признак високосного года (0)

cout.width(20);

for (month = 1; month <= 12; month++) {

switch(month) {

case 1:

cout << "\n\n\n" << " January" << "\n"; // Январь

num\_days\_per\_month = 31;

break;

case 2:

cout << "\n\n\n" << " February" << "\n"; // Февраль

num\_days\_per\_month = leap\_year\_flag ? 28 : 29;

break;

case 3:

cout << "\n\n\n" << " March " << "\n"; // Март

num\_days\_per\_month = 31;

break;

case 4:

cout << "\n\n\n" << " April " << "\n"; // Апрель

num\_days\_per\_month = 30;

break;

case 5:

cout << "\n\n\n" << " May " << "\n"; // Май

num\_days\_per\_month = 31;

break;

case 6:

cout << "\n\n\n" << " June " << "\n"; // Июнь

num\_days\_per\_month = 30;

break;

case 7:

cout << "\n\n\n" << " July " << "\n"; // Июль

num\_days\_per\_month = 31;

break;

case 8:

cout << "\n\n\n" << " August " << "\n"; // Август

num\_days\_per\_month = 31;

break;

case 9:

cout << "\n\n\n" << "September" << "\n"; // Сентябрь

num\_days\_per\_month = 30;

break;

case 10:

cout << "\n\n\n" << " October " << "\n"; // Октябрь

num\_days\_per\_month = 31;

break;

case 11:

cout << "\n\n\n" << "November " << "\n"; // Ноябрь

num\_days\_per\_month = 30;

break;

case 12:

cout << "\n\n\n" << "December " << "\n"; // Декабрь

num\_days\_per\_month = 31;

break;

}

cout.width(0) ;

// Bc Пн Вт Ср Чт Пт Сб

cout << "\nSun Mon Tue Wed Thu Fri Sat\n";

cout << "--- --- --- --- --- --- --- \n";

for ( date = 1; date <= 1 + jan\_1\_start\_day \* 5; date++ )

cout << " ";

for ( date = 1; date <= num\_days\_per\_month; date++ ) {

cout.width(2) ;

cout << date;

if ( ( date + jan\_1\_start\_day ) % 7 > 0 )

cout << " ";

else

cout << "\n ";

}

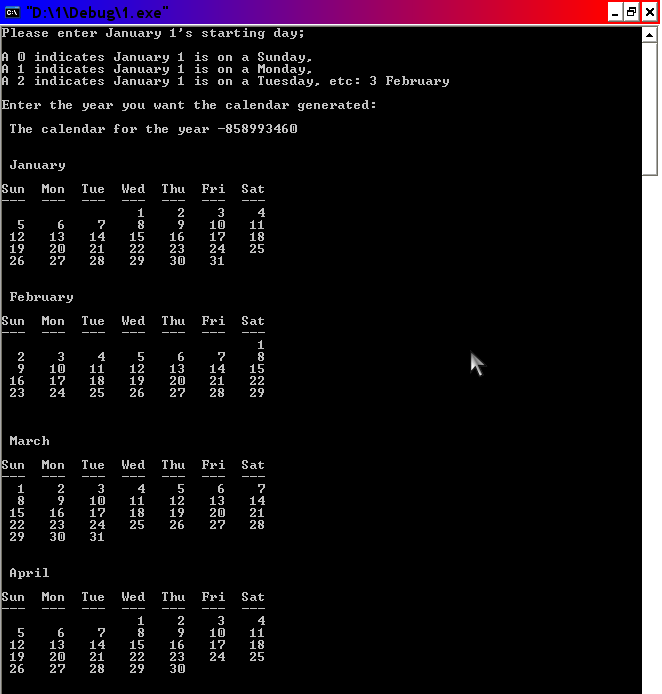
jan\_1\_start\_day=(jan\_1\_start\_day + num\_days\_per\_month) % 7;

}

cin >> date;

return(0);

}



Сначала в программе пользователь должен ввести целое число, представляющее собой день недели, соответствующий 1 января (0 — для понедельника, 1 — для вторника и так далее). Затем нужно ввести календарный год. После этого программа может напечатать заголовок календаря и использовать введенное значение года для создания флага високосного года leap\_year\_flag. При помощи операции деления по модулю (%) на 4 определяется остаток: ноль, если год — високосный, и ненулевое значение в противном случае.

Далее 12 раз выполняется цикл, в котором печатается название текущего месяца и переменной num\_days\_per\_month присваивается соответствующее значение: число дней в данном месяце. Все это выполняется при помощи оператора switch, в котором проверяется текущее целое значение месяца month.

Вслед за оператором switch, после печати названия месяца печатаются названия дней недели и пропускается некоторое количество пустых столбцов, в зависимости от того, на какой день недели пришелся первый день месяца.

И наконец, в последнем цикле for вычисляются и печатаются даты для каждого месяца. В последнем операторе программы вычисляется значение day\_code для следующего печатаемого месяца.

Операция условия и конструкция if-else облегчают написание программ, в которых осуществляется выбор между двумя вариантами. Однако иногда в программе необходимо произвести выбор одного из нескольких вариантов. Мы можем сделать это, используя конструкцию if-else if— ... —else, но во многих случаях оказывается более удобным применять оператор switch. Ниже приводится пример, иллюстрирующий его работу. Программа читает с терминала букву, затем выводит на печать название животного, начинающееся с этой буквы.

/\* животные \*/

main()

{

char ch;

printf(" Введите букву алфавита, а я укажу");

printf(" название животного,\n начинающееся с нее\n");

printf("Введите, пожалуйста, букву; для завершения работы введите #.\n");

while((ch = getchar()) != '#')

if( ch != '\n') /\* пропуск символа «новая строка» \*/

{

if (ch >= 'а' && ch <= 'я') /\* разрешены только строчные буквы \*/

switch (ch)

{

case 'a':

printf(" аргали, дикий горный азиатский баран \n");

break;

case ' б' :

printf(" бабирусса, дикая малайская свинья\n" );

break;

case ' в' :

printf(" выхухоль, водоплавающий крот\n");

break;

case ' г' :

printf(" гиббон, длиннорукая обезьяна\n");

break;

case 'д':

printf(" даман древесный\n"); break;

default:

printf("Это трудная задача!\n");

break;

}

else

printf(" Я распознаю только строчные буквы.\n");

printf(" Введите, пожалуйста, следующую букву или #.\n");

} /\* конец if, пропускающего символ «новая строка» \*/

} /\* конец цикла while \*/

}

Нам стало скучно продолжать, и мы остановились на букве д. Давайте теперь рассмотрим один пример выполнения программы перед тем, как обсудить использованные в ней новые средства языка.

Введите букву алфавита, а я укажу название животного, начинающееся с нее

Введите, пожалуйста, букву; для завершения работы введите #

а [возврат]

аргали, дикий горный азиатский баран

Введите, пожалуйста, следующую букву или #.

г [возврат]

гиббон, длиннорукая обезьяна

Введите, пожалуйста, следующую букву или #.

р [возврат]

Это трудная задача!

Введите, пожалуйста, следующую букву или #.

Т [возврат]

Я распознаю только строчные буквы.

Введите, пожалуйста, следующую букву или #.

# [возврат]

Этот пример служит иллюстрацией работы оператора switch. Вначале вычисляется выражение в скобках, расположенное за ключевым словом switch. В данном случае значением этого выражения будет символ, присвоенный переменной ch, который мы ввели перед этим. Затем программа просматривает список «меток» (в этом примере case 'а' :, case ' б' : и т. д.) до тех пор, пока не находит «метку», которая соответствует данному значению. Далее программа переходит к выполнению оператора, расположенного в этой строке. Что произойдет в случае, когда такой подходящей метки не найдется? Если существует строка с «меткой» case default:, то будет выполняться оператор, помеченный этой меткой. В противном случае произойдет переход к оператору, расположенному за оператором switch.

Что можно сказать по поводу оператора break? Его выполнение приводит к тому, что в программе происходит выход из оператора switch и осуществляется переход к следующему за ним оператору. При отсутствии оператора break будут выполнены все операторы, начиная с помеченного данной меткой и кончая оператором switch. Если удалить все операторы break из нашей программы, то, указав, например, букву г, получим следующий диалог:

Введите букву алфавита, а я укажу название животного, начинающееся с нее

Введите, пожалуйста, букву; для завершения работы введите # .

г [возврат]

гиббон, длиннорукая обезьяна

даман древесный

Это трудная задача!

Введите, пожалуйста, следующую букву или #.

# [возврат]

Мы видим, что выполнились все операторы, начиная от метки case'г' и кончая оператором switch. Если вы знакомы с языком Паскаль, то можете заметить, что оператор switch в Си похож на оператор case в Паскале. Важнейшее отличие состоит в том, что если вы хотите, чтобы в каждом конкретном случае выполнялся только помеченный оператор, то в операторе switch необходимо использовать операторы break.

Метки, имеющиеся в операторе switch, должны быть константами или константными выражениями (выражения, операнды которого константы) целого типа (включая тип char). Запрещается использовать в качестве метки переменную. Значением выражения в скобках должна быть величина целого типа (опять же, включая тип char). Ниже приводится общая структура оператора switch:

switch(цeлoe выражение)

{

case константа1:

операторы; (необязательные)

case константа2:

операторы; (необязательные)

…

case default: (необязательные)

операторы; (необязательные)

}

08_04

Когда мы хотим получить одинаковый результат при переходе к разным меткам, мы можем использовать метки без операторов. Например, фрагмент программы

case 'Е':

case 'е':

printf(" ехидна, муравьед колючий.\n");

break;

свидетельствует о том, что указание букв Е или е приводит к печати названия «ехидна ...». Если будет введена буква Е, то произойдет переход к соответствующей метке, но, поскольку там операторы отсутствуют, будут выполняться операторы, расположенные ниже, пока не встретится оператор break.

Наша программа имеет две небольшие особенности, о которых мы хотели бы упомянуть. Первая', поскольку мы собираемся использовать программу в диалоговом режиме, мы решили воспользоваться символом # вместо EOF в качестве признака прекращения ее работы. В работе компьютера могли бы возникнуть сложности, если бы он предложил нам ввести признак EOF или даже какой-нибудь управляющий символ, между тем как символ # вполне подходит для этой цели. Поскольку теперь отсутствует необходимость чтения символа EOF, мы не должны описывать в программе переменную ch типа int. Вторая: мы использовали оператор И, который позволяет игнорировать символы «новая строка» при вводе символов в программу. Это тоже некоторая плата за возможность диалоговой работы. Без этого оператора if каждый раз при нажатии клавиши [возврат] программе пришлось бы рассматривать данный признак как прочитанный символ.

Когда требуется использовать оператор switch, а когда конструкцию else-if? Часто у нас нет возможности выбора. Вы не можете применить оператор switch, когда выбор вариантов основывается на вычислении значения переменной или выражения типа float. Удобного способа воспользоваться оператором switch в случае, когда возможные значения переменной попадают в некоторый диапазон, также не существует. Проще написать, например, так:

if (integer < 1000 && integer > 2)

в то время как замена этой строки оператором switch приведет к необходимости ввести в программу метки для всех целых чисел от 3 до 999. Тем не менее, если у вас есть возможность применить оператор switch, ваша программа будет выполняться более эффективно.

### Совместное использование операторов if-else-if и switch.

В следующем программном примере перечисляемый тип (enum) используется для выполнения необходимых преобразований единиц измерения длины:

/\*07IFELSW.C

Программа на С, иллюстрирующая использование оператора if-else-if

в сочетании с различными операторами switch\*/

#include "stdafx.h"

#include <iostream>

#include <conio.h>

#include <stdio.h>

//#include <iostream.h>

using namespace std;

typedef enum conversion\_type {YARDS, INCHES, CENTIMETERS, METERS} C\_TYPE;

main()

{

int iuser\_response;

C\_TYPE C\_Tconversion;

int ilength=30;

float fmeasurement; /\* Укажите единицы измерения для преобразования : \*/

printf("\nPlease enter the measurement to be converted : ");

scanf("%f",&fmeasurement);

/\* Введите :

0— ярды

1— дюймы

2— сантиметры

3–метры\*/

printf("\nPlease enter : \

\n\t\t 0 for YARDS \

\n\t\t 1 for INCHES \

\n\t\t 2 for CENTIMETERS \

\n\t\t 3 for METERS \

\n\n\t\tYour response -->> ");

/\* Ваш ответ --> \*/

scanf("%d",&iuser\_response);

switch(iuser\_response) {

case 0 : C\_Tconversion = YARDS;

break;

case 1 : C\_Tconversion = INCHES;

break;

case 2 : C\_Tconversion = CENTIMETERS;

break;

default : C\_Tconversion = METERS;

}

if(C\_Tconversion == YARDS)

fmeasurement = ilength / 3;

else if(C\_Tconversion == INCHES)

fmeasurement = ilength \* 12;

else if(C\_Tconversion == CENTIMETERS)

fmeasurement = ilength \* 12 \* 2.54;

else if(C\_Tconversion = METERS)

fmeasurement = (ilength \* 12 \* 2.54)/100;

else

printf("No conversion required"); /\* Преобразование не требуется \*/

switch(C\_Tconversion) {

case YARDS : printf("\n\t\t %4.2f yards", fmeasurement);

break;

case INCHES : printf("\n\t\t %4.2f inches", fmeasurement);

break;

case CENTIMETERS : printf("\n\t\t %4.2f centimeters", fmeasurement);

break;

default : printf("\n\t\t %4.2f meters", fmeasurement);

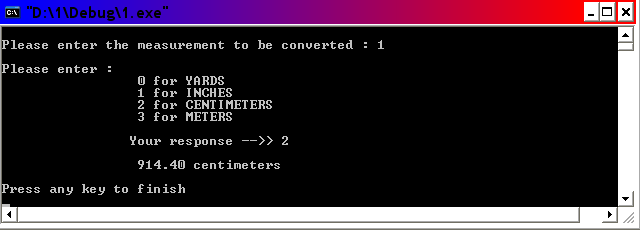
}

printf ("\n\nPress any key to finish\n");/\* Конец работы \*/

\_getch();

return(0);

}



В этом примере для указанного пересчета единиц длины используется перечисляемый тип данных. В стандартном С перечисляемые типы существуют только внутри самой программы (для обеспечения читабельности программы), и значения входящих в них переменных нельзя задавать или выводить непосредственно. Первый оператор switch в программе предназначен для преобразования введенного кода в соответствующий тип C\_Tconversion. Нужное преобразование выполняется при помощи вложенных операторов if-else-if. В последнем операторе switch печатается преобразованное значение и название единицы измерения. Конечно же, вложенные операторы if-else-if можно было бы записать при помощи оператора switch.

**Что вы должны были узнать.**

Как осуществить выбор из двух возможностей — выполнить оператор или пропустить его: с помощью оператора if.

Как осуществить выбор одного из двух вариантов: с помощью оператора if-else

Как осуществить выбор одного из нескольких вариантов: с помощью операторов else-if, switch

Операции отношения: > > = = = < = < ! =

Логические операции: && || !

Операция условия: :?

**Ключевое слово switch.**

*Общие замечания.* Управление в программе передается оператору, у которого в качестве метки используется значение некоторого выражения. Затем в процессе прохождения программы будут выполняться оставшиеся операторы, пока не произойдет новый переход.

Как выражения, так и метки должны иметь значения целого типа (включая тип char); метки должны быть константами или константными выражениями. Если некоторому значению выражения не соответствует никакая метка, управление передается оператору с меткой default (если такой имеется). В противном случае управление передается оператору, следующему за оператором switch.

*Форма.*

switch (выражение)

{

case метка1: оператор1

case метка2: оператор2

default: операторЗ

}

В операторе может присутствовать более чем 2 помеченных оператора, а наличие метки default является необязательным.

*Пример.*

switch (letter)

{

case 'a' :

case 'e' : printf(" %c — это гласная\n" , letter);

case 'с' :

case 'n' : printf(" Символ %c в наборе букв \"саnе\" n", letter);

default : printf(" Добрый день\n");

}

Если переменная letter имеет значение 'а' или 'е', будут выведены на печать все три сообщения, если же 'c' или 'n', то последние два. В случае остальных значений будет напечатано только последнее сообщение.

Изложенный здесь материал позволит вам писать гораздо более мощные и обладающие большими возможностями программы, чем раньше. В справедливости этого утверждения вы сможете убедиться, если сравните некоторые из примеров, приведенных здесь, с программами, рассмотренными ранее. Но вы изучили еще далеко не все. Вот почему вам придется одолеть еще немало материала.

**Вопросы и ответы.**

***Вопросы.***

1. Определите, какие выражения истинны, а какие ложны.

а. 100 > 3

б. 'а' > 'с'

в. 100 > 3 && ' а' > ' с'

г. 100 > 3 || 'а' > 'с'

д. !(100 > 3)

2. Запишите выражения, соответствующие следующим условиям:

а. Значение number равно или больше 1, но меньше 9.

б. Значение ch не равно q или k.

в. Значение number лежит между 1 и 9, но не равно 5,

г. Значение number не лежит между 1 и 9.

3. В программе, приведенной ниже, наряду с неоправданно сложными условными выражениями имеются и прямые ошибки. Уточните эту программу и исправьте в ней ошибки.

main() /\* 1 \*/

{

/\* 2 \*/

int weight, height; /\* вес в фунтах, рост в дюймах \*/

/\* 4 \*/

scanf(" %d, weight, height);/\* 5 \*/

if (weight < 100) /\* 6 \*/

if ( height > = 72) /\* 7 \*/

printf(" Для такого веса у вас слишком большой рост.\n");

else if (height < 72 && > 64) /\* 9 \*/

printf(" У вас большой рост для вашего веса\n");

else

if (weight > 300 && !(weight < = 300) /\* 11 \*/

if ( !(height > = 48) /\* 12 \*/

printf(" Для такого веса у вас слишком маленький рост.\n");

else /\* 14 \*/

printf(" У вас идеальный вес.\n"); /\* 15 \*/

/• 16 \*/

}

**Ответы.**

1. Выражения истинны в вопросах а и г.

2.

a. number > = 1 && number < 9

б. ch != 'q' && ch != 'k'

Замечание: выражение ch != 'q' || ch!= 'k' всегда будет иметь значение «истина», потому что если переменная ch равна q, то она не может равняться k, и второе условие оказывается выполненным: в результате все выражение «ИЛИ» будет истинным.

в. number > 1 && number < 9 && number != S

г. ! (number > 1 && number < 9)

или

number < = 1 II number > = 9

Замечание: сказать, что число НЕ лежит между 1 и 9 это то же самое, что сказать: число равно или меньше 1 ИЛИ равно или больше 9. Вторая форма несколько неуклюже звучит на словах, но проще записывается в виде выражения.

3. Строка 5 должна выглядеть так: scanf(" %d %d", &weight, &height); He забывайте указывать символы в качестве префиксов имен переменных в функции scanf(). Кроме того, данной строке должна предшествовать строка, предлагающая ввести данные.

Строка 9: подразумеваемое выражение должно выглядеть так: (height < 72 && height > 64). Однако первая часть этого выражения необязательна, поскольку величина height, если поток управления достигнет записи else-if, будет обязательно меньше 72. Поэтому более простое условие (height > 64) в данном случае служит той же цели.

Строка 11: избыточное условие; второе подвыражение (отрицание условия «величина weight меньше или равна 300») означает то же, что и первое. В действительности данное условие записывается так: (weight > 300). Но неприятности на этом не кончаются. Строка 11 относится к ошибочному оператору if! Очевидно, что эта часть else ассоциируется с оператором К, расположенным на строке 6, но, согласно правилу, связывающему ее с ближайшим отрицанием условия, содержащегося в if, она будет ассоциироваться с оператором if на строке 9. Поэтому условие, помещенное на строке 11, будет проверяться в том случае, когда величина weight меньше 100, а величина height меньше или равна 64. Это делает невозможным превышение переменной weight значения 300 при выполнении данного оператора.

Строки 7—9 должны быть заключены в фигурные скобки. Тогда строка 11 станет альтернативой оператору, расположенному на строке 6, а не на строке 9.

Строка 12: данное выражение необходимо упростить так: (height < 48)

Строка 14: это ключевое слово else относится к последнему оператору if, расположенному на строке 12. Операторы, помещенные на строках 12 и 13, необходимо заключить в фигурные скобки, тогда else будет относиться к оператору if на строке 11. Обратите внимание, что последнее сообщение будет напечатано для тех, чей вес заключен между 100 и 300 фунтами.