НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
Кафедра информатики и процессов управления (№17)

Дисциплина «Информатика» (основной уровень), 1-й курс, 1-й семестр.

Методические указания

Тематическое занятие 2

Логические выражения. Разветвляющиеся алгоритмы.

Содержание

Операции логических выражений

Операции отношения

Операции отношения выполняют сравнение двух операндов.

`==`	равно
`!=`	не равно (≠)
`>`	больше
`<`	меньше
`>=`	больше или равно
`<=`	меньше или равно

Логические операции

Логические операции и позволяют получать более сложные выражения.

!	отрицание «НЕ», NOT (*инверсия*)
&&	логическое «И», AND (*конъюнкция*)
||	логическое «ИЛИ», OR (*дизъюнкция*)

Результат выполнения логических операций задается *таблицей истинности*:

a	**b**	**! a**	**a && b**	**a || b**
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Приоритет операций

Таблица операций отношения и логических операций языка С в порядке приоритета:

Операции	***Ассоциирование***	***Приоритет***
()	→ слева направо	высокий
!	← справа налево	
< <= > >=	→ слева направо	
== !=	→ слева направо	
&&	→ ***слева направо***	
||	→ ***слева направо***	низкий

Пример. Следующие записи выражения ***«*** `x` ***не лежит в диапазоне (–2; 2) »*** эквивалентны:

```
!(x>-2 && x<2)      и      x<=-2 || x>=2
```

Вычисление логического выражения

В языке С не существует отдельного логического типа данных. По определению числовое значение логического выражения или сравнения равно `1`, если выражение *истинно*, и `0` – если *ложно*.

Например, в результате выполнения фрагмента программы

```
int a;
a = 10;
printf("%d",a>5);
```

на экран будет выведено значение `1`.

Вычисление выражения, содержащего логические операции `&&` и `||`, выполняется *слева направо* и прекращается, как только установлено гарантированное значение результата (истина или ложь). Например, при `a=10` вычисление выражения с операцией `&&`

```
a>12 && (b=15)>a
```

окончится уже на первом операнде (`a>12` – ложь, а значит результат всего выражения – ложь, независимо от второго операнда), и присваивание (`b=15`) не

будет выполнено. Вычисления выражений с логическими операциями происходят в этом порядке, даже не смотря на то, что скобки обладают более высоким приоритетом, чем `&&` и `||`.

Ветвления

Составной оператор

Составной оператор (блок операторов) – группа из произвольного числа операторов, которая ограничена *операторными скобками* – символами `{` и `}`. Составной оператор воспринимается как один оператор и используется там, где может стоять только один оператор, а требуется использовать несколько.

За составным оператором знак «точка с запятой» не ставится.

Условный оператор `if-else`

Алгоритмическая конструкция *ветвление* позволяет выбрать между несколькими вариантами действий в зависимости от заданного условия. Ветвление реализуется условным оператором и оператором выбора.

Условный оператор `if` имеет две синтаксических формы записи:

полная форма:

```
if (Выражение)
    ОператорИст
else
    ОператорЛож
```

сокращенная форма:

```
if (Выражение)
    ОператорИст
```

здесь: `ОператорИст` выполняется если `Выражение` истинно, `ОператорЛож` – если `Выражение` ложно. Ключевые слова `if`, `else` означают «**если**», «**иначе**» соответственно.

Выполнение условного оператора начинается с вычисления `Выражения`, которое считается истинным, если принимает ***ненулевое*** значение, и – ложным, если имеет нулевое значение. То есть для такой записи

```
if (a != 0) ...
```

предпочтительнее использовать более краткую форму

```
if (a) ...
```

поскольку в этом случае выполняется на одну операцию сравнения меньше, т.е. программа работает быстрее.

В такой конструкции `ОператорИст` (и `ОператорЛож`) может быть только одним оператором, блоком операторов или вообще отсутствовать (пустой оператор). Если требуется выполнить несколько операторов, то следует использовать составной оператор:

```
if (Выражение) {
    ОператорИст1
    ОператорИст2
    ...
    ОператорИстN
}
```

```
else {
    ОператорЛож1
    ОператорЛож2
    ...
    ОператорЛожМ
}
```

Пример. Сравнение двух чисел:

```
#include <stdio.h>
int main(void) {
   int a, b;
   printf("Input a=");    scanf("%d", &a);
   printf("Input b=");    scanf("%d", &b);
   if (a>b)
      printf("a>b\n");
   else
      printf("a<=b\n");
   return 0;
}
```

! Замечание: *Для наглядности строки кода рекомендуется набирать с отступом. Причем, ключевое слово* `else` *стараются располагать строго под соответствующим ему* `if`*.*

Тернарная операция условия

Конструкции с условным оператором `if` можно записать другим способом, используя условное выражение с трехместной (тернарной) операцией «`?:`». Такое выражение имеет следующий вид:

```
ВыражУсл ? ВыражИст : ВыражЛож
```

Вначале вычисляется выражение `ВыражУсл`. Если оно истинно (т.е. не равно нулю), то вычисляется `ВыражИст`, значение которого становится значением всего условного выражения. В противном случае, вычисляется `ВыражЛож`, и его значение становится значением всего выражения. Всегда вычисляется только одно из `ВыражИст` и `ВыражЛож`.

Например, следующий код с условным оператором

```
if (a > b)
   c = a;
else
   c = b;
```

можно записать в виде выражения

```
c = (a > b) ? a : b ;    /* c = max(a,b) */
```

Скобки в первом операнде ставить необязательно, но рекомендуется, поскольку это улучшают восприятие текста программы.

Если выражения `ВыражИст` и `ВыражЛож` имеют различные типы, то тип результата определяется общими правилами преобразования типов, рассмотренными ранее. Например, если `x` имеет тип `float`, а `n` – тип `int`, то выражение

```
(n > 0) ? x : n
```

имеет тип `float` независимо от положительности значения `n`.

В выражениях приоритет тернарной операции условия самый низкий из всех рассмотренных, но выше, чем у операции присваивания. Порядок ассоциирования – справа налево (←).

Вложенность условного оператора

Один оператор `if` может входить в состав другого оператора `if`. *Вложенный* оператор может иметь вид:

```
if (Выраж1)
   if (Выраж2)
      ОператорИст2
   else
      ОператорЛож2
else
   ОператорЛож1
```

Ключевое слово `else` всегда ассоциируется с ближайшим предыдущим оператором `if` без `else`. Поэтому, если у вложенного `if` отсутствует раздел `else`, то нужно использовать блок:

```
if (Выраж1) {
   if (Выраж2)
      ОператорИст2
}
else
   ОператорЛож1
```

! Замечание: *Следует **избегать** конструкций с **вложенностью** условного оператора **более трёх уровней** из-за сложности их анализа при отладке программы. За исключением следующей конструкции* `else-if`.

Конструкция `else-if`

Очень распространенный вариант вложенного условного оператора:

```
if (Выраж1)
   ОператорИст1
else if (Выраж2)
   ОператорИст2
else if (Выраж3)
   ОператорИст3
else
   ОператорЛож3
```

В программе сравнения двух чисел оператор `if` можно заменить:

```
if (a>b)
   printf("a>b\n");
else if (a<b)
   printf("a<b\n");
else
   printf("a=b\n");
```

Множественный выбор

Оператор `switch`

Для выбора одного из нескольких вариантов действий используется оператор множественного выбора:

```
switch (Выражение) {
   case КонстантВыраж1: Операторы1
   case КонстантВыраж2: Операторы2
   ...
   case КонстантВыражN: ОператорыN
   default: ОператорыИначе
}
```

Если значение `Выражения` совпадает со значением одного из **целочисленных** константных выражений `КонстантВыраж1`, `КонстантВыраж2`, `...`, `КонстантВыражN`, то выполняются операторы, идущие после соответствующей метки `case`. Если не найдено ни одного соответствия, то выполняются операторы, идущие после метки `default`.

Блоки `case` – это, по сути, всего лишь метки, и после выполнения операторов в одном из них, продолжается выполнение операторов в следующем блоке `case`, пока не будет предпринят принудительный выход из `switch`. Такой немедленный выход может быть сделан оператором `break`. Использование оператора `switch` демонстрируется на следующем примере.

Пример множественного выбора

Программа, которая расставляет окончания слова «штука» в соответствии с правилами русского языка, в зависимости от числа, введенного пользователем:

```
#include <stdio.h>
int main(void) {
   int k;
   char flex;
   printf("Введите количество деталей:");
   scanf("%d", &k);
   switch (k) {
      case 1:
         flex = 'а';
         break;
      case 2:
      case 3:
      case 4:
         flex = 'и';
         break;
      default:
         flex = ' ';
         break;
   }
   printf("Количество деталей: %d штук%c\n", k, flex);
   return 0;
}
```

Если, например, убрать первый оператор `break`, то при `k=1` переменной `flex` вначале будет присвоено значение `'а'`, а затем – значение `'и'`. После этого будет выполнен выход из оператора `switch` по второму `break`, и выводимое сообщение будет неверно.

Такого ***сквозного*** выполнения оператора `switch` следует избегать, за исключением использования нескольких меток для одной и той же операции (как в приведенном примере, когда `k=2`, `3` или `4`).

Кроме того, следует ставить оператор `break` даже в конце последнего блока, хотя в этом нет необходимости. Это – хороший стиль программирования, который может подстраховать от лишних неприятностей при добавлении еще одного блока `case`.

Кстати, данная программа будет правильно расставлять окончания только для неотрицательных целых чисел `k`, не превосходящих значения `20`. Чтобы эта программа корректно работала для любых неотрицательных целых чисел, в операторе `switch` необходимо изменить `Выражение`, например, так:

```
switch ( (k<=20) ? k : k%10 ) {
   ...  /* те же блоки case */
}
```

! Замечание: *Вообще, **сквозной** метод программирования **не способствует** устойчивости программы к изменениям, поскольку при ее доработке могут возникать ошибки и побочные эффекты.*

Русский язык и локализация

В приведенном выше примере необходимо выводить русские буквы, для этого нужно переключиться на кодировку кириллицы.

Национальные стандарты могут быть подключены в ходе локализации с помощью заголовочного файла стандартной библиотеки `<locale.h>`. Используя описанную в этом заголовочном файле функцию `setlocale()` можно изменить текущую кодировку и правила форматирования чисел.

```
#include <stdio.h>
#include <locale.h>
int main(void) {
   setlocale(LC_ALL,"");
   printf("Вывод русских букв.\nРазделитель целой и дробной ");
   printf("части числа – запятая: %f\n",3.141592);
   return 0;
}
```

Упражнения

Упражнение 2.1

Составить программу, которая запрашивает у пользователя два целых числа и выводит на экран наибольшее из них. В случае если числа равны, программа не выводит ничего.

Упражнение 2.2

Составить программу, которая запрашивает у пользователя целое число и, если это число положительное и четное, выводит на экран соответствующее сообщение.

Упражнение 2.3

Составить программу, которая запрашивает у пользователя два целых числа. Если одно из чисел делит другое нацело (без остатка), то программа выводит на экран результат целочисленного деления, в противном случае программа выводит их сумму.