2.3. Классификация операторов. Условные операторы

Краткая характеристика операторов языка Си

Операторы языка Си можно разделить на три группы: операторы-декларации (рассмотрены ранее), операторы преобразования объектов и операторы управления процессом выполнения алгоритма.

Программирование процесса преобразования объектов производится посредством записи операторов (инструкций).

Простейший вид операторов – ***выражение***, заканчивающееся символом «;» (точка с запятой). Выполнение такого оператора заключается в вычислении некоторого выражения.

Простые операторы: оператор присваивания (выполнение операций присваивания), оператор вызова функции (выполнение операции вызова функции), пустой оператор «;» – частный случай выражения. Пустой оператор используют тогда, когда по синтаксису оператор требуется, а по смыслу – нет (например, смотри бесконечный оператор цикла *for* в разд. 7.4).

Примеры операторов *«выражение»*:

i++; – выполняется операция инкремента (увеличение на 1);

x+*y*; – выполняется операция сложения (результат будет утерян);

a = *b*–*c*; – выполняется операция вычитания с одновременным присваиванием.

Операторы языка Си записываются в свободном формате с использованием разделителей между ключевыми словами. Любой оператор может помечаться меткой – идентификатор и символ «:» (двоеточие). Область действия метки – функция, где эта метка определена.

К ***управляющим операторам*** относятся: операторы условного и безусловного переходов, оператор выбора альтернатив (переключатель), операторы организации циклов и передачи управления (перехода).

Каждый из управляющих операторов имеет конкретную лексическую конструкцию, образуемую из ключевых слов языка Си, выражений и символов-разделителей.

Допускается вложенность операторов. В случае необходимости можно использовать составной оператор – блок, состоящий из любой последовательности операторов, заключенных в фигурные скобки – { и }, после закрывающей скобки символ «;» не ставится.

Условные операторы

Условный оператор ***if*** используется для разветвления процесса выполнения кода программы на два направления.

В языке Си имеется две разновидности условного оператора: простой и полный. Синтаксис ***простого*** оператора:

if **(*выражение*)** ***оператор*;**

выражение – логическое или арифметическое выражение, вычисляемое перед проверкой, и, если выражение истинно (не равно нулю), то выполняется ***оператор***, иначе он игнорируется; ***оператор*** – простой или составной (блок) оператор языка Си. Если в случае истинности выражения необходимо выполнить несколько операторов (более одного), их необходимо заключить в фигурные скобки.

Структурная схема простого оператора приведена на рис. 6.1.

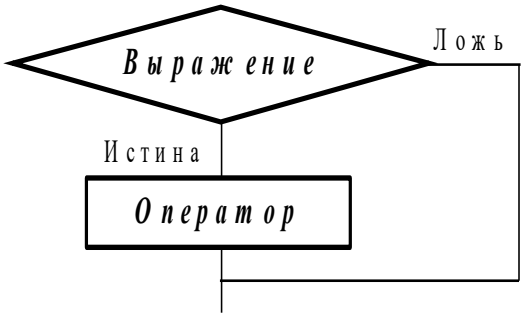

Рис. 6.1

Примеры записи условного оператора *if*:

```
if (x > 0) x = 0;
if (i != 1)   j++,  s = 1;     – используем операцию «запятая»;
if (i != 1) {
                j++;  s = 1;   – последовательность операций (блок);
}
if (getch() != 27) k = 0;   – если нажата любая клавиша кроме “Esc”.
if (!x) exit (1);       или    if (x == 0) exit(1);
```

if (i>0 && i<n) k++; – если нужно проверить несколько условий, то их объединяют знаками логических операций и заключают в круглые скобки (для улучшения читаемости программы можно ставить круглые скобки и там где они необязательны);

if (a++) b++; – необязательно в качестве выражения использовать логические выражения.

Синтаксис ***полного*** оператора условного выполнения:

***if* (*выражение*) *оператор* 1 ;**
else *оператор* 2 ;

Если ***выражение*** не равно нулю (истина), то выполняется ***оператор* 1**, иначе – ***оператор* 2**. Операторы 1 и 2 могут быть простыми или составными (блоками).

Наличие символа **«;»** перед словом ***else*** в языке Си обязательно.

Структурная схема такого оператора приведена на рис. 6.2.

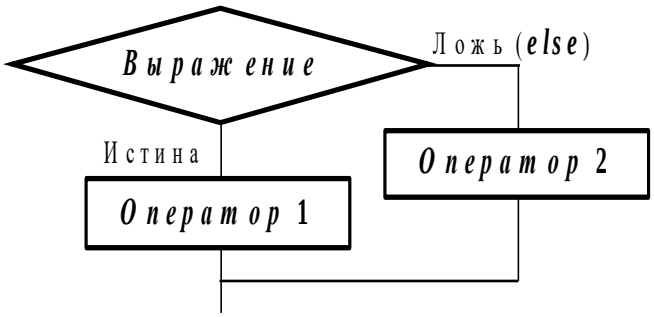

Рис. 6.2

Примеры записи:

```
if (x > 0) j = k+10;
        else m = i+10;

if ( x>0 &&  k!=0 ) {
                j = x/k;
                x += 10;
}
else m = k*i + 10;
```

Операторы 1 и 2 могут быть любыми операторами, в том числе и условными. Тогда, если есть вложенная последовательность операторов *if – else,* то фраза *else* связывается с ближайшим к ней предыдущим *if*, не содержащим ветвь *else*. Например:

```
if (n > 0)
   if(a > b)  z = a;
        else   z = b;
```

Здесь ветвь *else* связана со вторым *if* (a > b). Если же необходимо связать фразу *else* с внешним *if*, то используются операторные скобки:

```
if(n > 0) {
        if(a > b)  z = a;
        }
  else   z = b;
```

В следующей цепочке операторов *if – else – if* выражения просматриваются последовательно:

```
if (выражение 1)  оператор 1;
   else
        if (выражение 2)  оператор 2;
           else
                if (выражение 3)  оператор 3;
                   else   оператор 4 ;
```

Если какое-то *выражение* оказывается истинным, то выполняется относящийся к нему *оператор* и этим вся цепочка заканчивается. Каждый оператор может быть

либо отдельным оператором, либо группой операторов в фигурных скобках. ***Оператор* 4** будет выполняться только тогда, когда ни одно из проверяемых условий не подходит. Иногда при этом не нужно предпринимать никаких явных действий, тогда последний *else* может быть опущен или его можно использовать для контроля, чтобы зафиксировать «невозможное» условие (своеобразная экономия на проверке условий).

Пример:

```
if (x < 0) printf("\n X отрицательное \n");
        else if(x==0) printf ("\n X равно нулю \n");
              else printf("\n X положительное \n");
```

Замечание. Наиболее распространенной ошибкой при создании условных операторов является использование в выражении операции присваивания «=» вместо операции сравнения на равенство операндов «==» (два знака равно). Например, в следующем операторе синтаксической ошибки нет:

***if* (*x* = 5) *a*++;**

но значение ***a*** будет увеличено на единицу независимо от значения переменной ***x***, т.к. результатом операции присваивания $x = 5$ в круглых скобках является значение $5 \neq 0$ – истина.

Условная операция «? :»

Условная операция – ***тернарная***, т.к. в ней участвуют три операнда. Формат написания условной операции следующий:

Выражение 1 **?** *выражение* 2 **:** *выражение* 3**;**

если *выражение* 1 (условие) отлично от нуля (истинно), то результатом операции является значение *выражения* 2, в противном случае – значение *выражения* 3. Каждый раз вычисляется только одно из выражений 2 или 3.

На рис. 6.3 приведена схема вычисления результата, которая аналогична схеме полного оператора *if* (см. рис. 6.2):

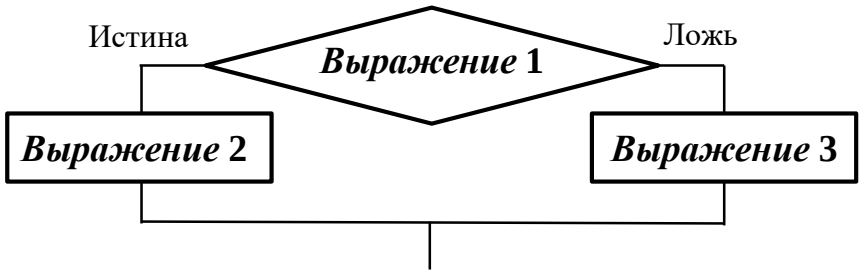

Рис. 6.3

Условное вычисление применимо к арифметическим операндам и операндам-указателям.

Рассмотрим участок программы для нахождения максимального значения ***z*** из двух чисел ***a*** и ***b***, используя оператор *if* и условную операцию.

1. Запишем оператор *if* :

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

2. Используя условную операцию, получим

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

Условную операцию можно использовать так же, как и любое другое выражение. Если выражения 2 и 3 имеют разные типы, то тип результата определяется по правилам преобразования. Например, если *f* имеет тип *double*, а *n* – *int*, то результатом операции

```
(n > 0) ? f : n;
```

по правилам преобразования типов будет *double*, независимо от того, положительно *n* или нет.

Использование условных выражений позволяет во многих случаях значительно упростить программу. Например:

```
int a, x;
   ...
x = (a < 0) ? –a : a;
printf("\n Значение %d  %s нулевое !", x, (x ? "не" : " ") );
```

Оператор выбора альтернатив (переключатель)

Оператор *switch* (переключатель) предназначен для разветвления процесса вычислений на несколько направлений.

Общий вид оператора:

```
switch ( выражение ) {
    case  константа1:      список операторов 1
    case  константа2:      список операторов 2
                ...
    case  константаN:      список операторов N
    default:  список операторов N+1          – необязательная ветвь;
    }
```

Выполнение оператора начинается с вычисления ***выражения***, значение которого должно быть целого или символьного типа. Это значение сравнивается со значениями ***констант*** и используется для выбора ветви, которую нужно выполнить.

В данной конструкции ***константы*** фактически выполняют роль меток. Если значение выражения совпало с одной из перечисленных констант, то управление передается в соответствующую ветвь. После этого, если выход из переключателя в данной ветви явно не указан, последовательно выполняются все остальные ветви.

Все константы должны иметь разные значения, но быть одного и того же типа. Несколько меток могут следовать подряд, и тогда переход в указанную

ветвь будет происходить при совпадении хотя бы одной из них. Порядок следования ветвей не регламентируется.

В случае несовпадения значения выражения ни с одной из констант выбора происходит переход на метку ***default*** либо, при ее отсутствии, к оператору, следующему за оператором *switch*.

Управляющий оператор ***break*** (разрыв) выполняет выход из оператора *switch*. Если по совпадению с каждой константой должна быть выполнена одна и только одна ветвь, схема оператора switch следующая:

```
switch (выражение) {
                case константа1: операторы 1;  break;
                case константа2: операторы 2;  break;
                        ...
                case константаN: операторы N;  break;
                default:  операторы (N+1);  break;
        }
```

Структурная схема рассмотренной конструкции (с использованием оператора *break*) приведена на рис. 6.4.

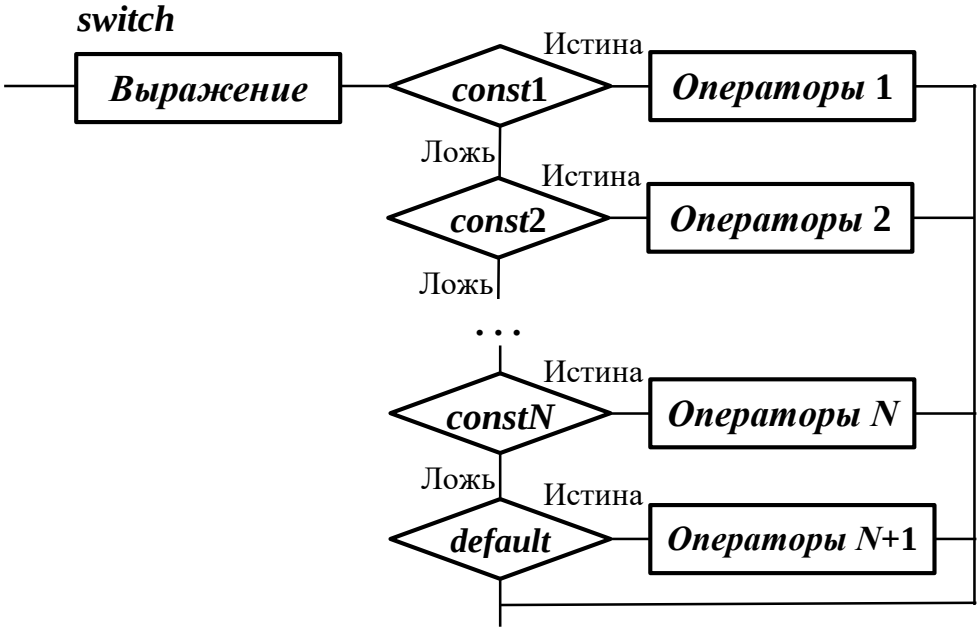

Рис. 6.4

Пример оператора *switch* с использованием оператора *break*:

```
void main(void)    {
        int i = 2;
        switch(i)      {
                case 1:  puts ( "Случай 1. "); break;
                case 2:  puts ( "Случай 2. "); break;
                case 3:  puts ( "Случай 3. "); break;
                default: puts ( "Случай default. "); break;
```

```
        }
}
```

Результатом выполнения данной программы будет:

Случай 2.

Аналогичный пример без использования оператора *break* (схема общего вида такой конструкции приведена рис. 6.5):

```
void main() {
        int i = 2;
        switch(i)      {
                case 1: puts ( "Случай 1. ");
                case 2: puts ( "Случай 2. ");
                case 3: puts ( "Случай 3. ");
                default: puts ( "Случай default. ");
                }
}
```

В данном случае результат будет следующим:

Случай 2.
Случай 3.
Случай default.

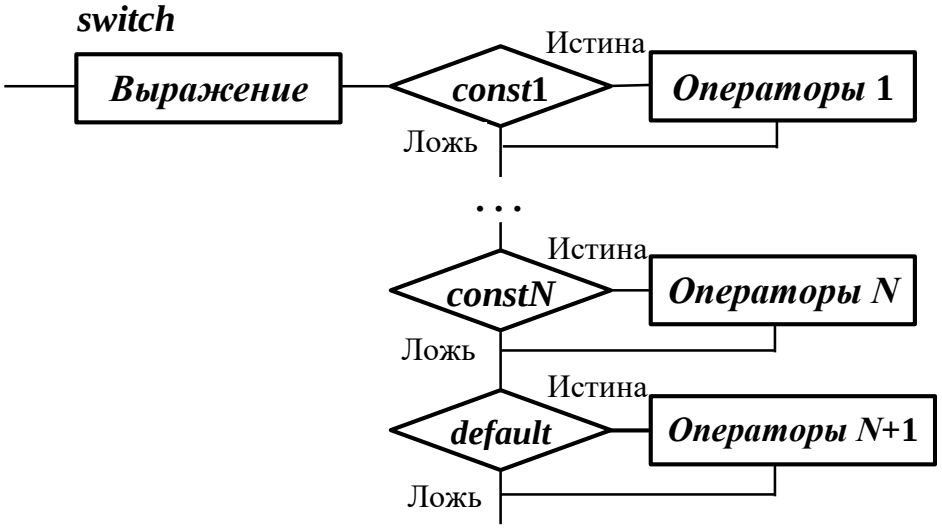

Рис. 6.5

Пример реализации простейшего калькулятора на четыре действия с контролем правильности ввода символа нужной операции. Ввод данных осуществляется следующим образом: операнд 1, символ нужной операции, операнд 2.

Текст программы может быть следующим:

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

```
void main(void)
{
        double a, b, c;
        char s;
        m1:  fflush(stdin);                  // Очистка буфера ввода stdin
        printf("\n Введите операнд 1, символ операции, операнд 2:");
        scanf("%lf%c%lf", &a, &s, &b);
        switch(s)  {
                case '+':       c = a+b;       break;
                case '–':       c = a–b;       break;
                case '*':       c = a*b;       break;
                case '/':       c = a/b;       break;
                default: printf("\n Ошибка, повторите ввод! "); goto m1;
                }
          printf("\n a %c b = %lf ", s, c);
          printf("\n Продолжим? (Y/y) ");
          s = getch();
          if ( (s=='Y') || (s=='y') ) goto m1;
          printf("\n Good bye! ");
}
```

После запуска программы на экран выводится подсказка, нужно набрать соответствующие значения без пробелов, например, как показано ниже, и нажать клавишу *Enter*:

Введите операнд 1, символ операции, операнд 2: 2.4+3.6

На экран будет выведен результат и дальнейший диалог:

a + b = 6.000000

Продолжим? (Y/y)

Введя символ **y (Y),** вернемся в начало функции и на экране вновь появится:

Введите операнд 1, символ операции, операнд 2:

Если ошибочно ввести – **2r3 ,** появятся следующие сообщения:

Ошибка, повторите ввод!

*Введите операнд 1, символ операции, операнд 2: 2 * 3*

*a*b = 6.000000*

Continue? (Y/y)

Нажимаем любую клавишу, кроме *y* или *Y* – следует сообщение

Good bye!

Программа закончена.