3. Управляющие конструкции

Владимир Верстов

Б. Керниган, Д. Ритчи. Язык программирования C

Оператор

- … — это выражение с точкой с запятой в конце

- Точка с запятой - элемент оператора, его завершающая часть

- Примеры:

```
x = 0;
i++;
printf("%d %d\n", x, i);
```

Блок

- … — это группа выражений, заключенных в фигурные скобки, `{` и `}`

- Блок синтаксически эквивалентен одному оператору

- Типовые и очевидные примеры блоков: функции, конструкции `if`, `else`, `while` и `for`

Оператор `if-else`

- Выражает процесс принятия решений
- `else` необязательно
- Если `выражение` истинно, то выполняется `оператор1`
- Если выражение ложно, то выполняется `оператор2`

```
if (выражение)
  оператор1
else
  оператор2
```

Оператор `else-if`

- Последовательность операторов `if` описывает процесс принятия многовариантного решения
- `Выражения` вычисляются по порядку
- Если какое-либо `выражение` истинно, то выполняется соответствующий `оператор`
- `else` обрабатывает вариант ничто из перечисленного и является опциональным

```
if (выражение1)
  оператор1
else if (выражение2)
  оператор2
else if (выражение3)
  оператор3
else if (выражение4)
  оператор4
else
  оператор5
```

Двоичный поиск

```
/* Алгоритма поиска числа в упорядоченном
   по возрастанию массиве.
   -1 означает, что число не найдено */
int binsearch(int x, int v[], int n) {
    int low, high, mid;
    low = 0;
    high = n - 1 ;
    while (low <= high) {
        mid = (low + high) / 2;
        if (x < v[mid])
            high = mid - 1;
        else if (x > v[mid])
            low = mid + 1 ;
        else
            return mid;
    }
    return -1;
}
```

Оператор `switch`

- Выбор одного из вариантов в зависимости от того, с какой из `констант` совпадает результат некоторого `выражения`
- Блок `default` не обязателен
- Все выражения после `case` должны быть различны

```
switch (выражение) {
  case константное-выражение1:
    операторы
  case константное-выражение2:
    операторы
  case константное-выражение3:
    операторы
  default:
    операторы
}
```

```c
#include <stdio.h>
/* подсчет символов пустого пространства, цифр и остальных */
int main() {
    int c, i, nwhite, nother, ndigit[10];
    nwhite = nother = 0;
    for (i = 0; i < 10; i++)
        ndigit[i] = 0;
    while ((c = getchar()) != EOF) {
        switch (c) {
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
            ndigit[c - '0']++;
            break;
        case ' ':
        case '\n':
        case '\t':
            nwhite++;
            break;
        default:
            nother++;
            break;
        }
    }
    printf ("digits =");
    for (i = 0; i < 10; i++)
        printf (" %d", ndigit[i]);
    printf (", white = %d, other = %d\n", nwhite, nother);
    return 0;
}
```

Оператор `switch`

- Оператор `break` означает немедленных выход из оператора `switch`

- Если `case` не закачивается `break`, то последующие `case` выполняются последовательно

Циклы while и for

```
while (выражение)
  оператор
```

```
for (выражение1; выражение2; выражение3)
  оператор

выражение1;
while (выражение2)
  оператор
  выражение3;
```

Бесконечный цикл

```
while (true) {
    // ...
}

for (;;) {
    // ...
}
```

Преобразование строки в число

```
пропустить символы пустого пространства, если такие есть
обработать знак, если он есть
прочитать целую часть и преобразовать в число
```

Преобразование строки в число

```
#include <ctype.h>

int atoi(char s[])
{
    int i, n, sign;
    for (i = 0; isspace(s[i]); i++)
        ;
    sign = (s[i] == '-') ? -1: 1;
    if (s[i] == '+' || s[i] == '-')
        i++;
    for (n = 0; isdigit(s[i]); i++)
        n = 10 * n + (s[i] - '0' );
    return sign * n;
}
```

Реверс строки

```c
#include <string.h>

void reverse(char s[]) {
    int c, i, j;
    for (i = 0, j = strlen(s) - 1; i < j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

Реверс строки

- `strlen` - функция для определения длины строки из библиотеки `string.h`

- запятая не является знаком операций и не гарантирует порядок их выполнения

- запятую следует использовать как можно реже и осторожно

Цикл do-while

```
do
   оператор
while (выражение)
```

- Циклы `for` и `while` - циклы с предусловием
- Цикл `do-while` - цикл с постусловием
- Тело цикла `do-while` выполниться как минимум 1 раз
- Тело циклов `for` и `while` может ни раз не выполнится
- Цикл `do-while` применяется значительно реже циклов `for` и `while`

Преобразование цикла в строку

```
void itoa(int n, char s[]) {
    int i, sign;
    if ((sign = n) < 0)
        n = -n;
    i = 0;
    do {
        s[i++] = n % 10 + '0';
    } while ((n /= 10) > 0);
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}
```

Оператор `break`

- Оператор `break` - принудительный выход из цикла или оператора `switch`

```
/* удаление символов пустого пространства из конца строки */
int trim(char s[]) {
   int n;
   for (n = strlen(s) - 1; n >= 0; n--)
      if (s[n] != ' ' && s[n] != '\t' && s[n] != '\n')
         break;
   s[n + 1] = '\0';
   return n;
}
```

Оператор `continue`

- Оператор `continue` передает управление на следующую итерацию циклов

- Оператор `continue` используется реже, чем оператор `break`

- Пример:

```
for (i = 0 ; i < n; i++) {
    if (a[i] < 0)
        continue;
    // ...
}
```

Оператор `goto`

- источник потенциальных проблем
- никогда не бывает необходим
- почти всегда легче обойтись без него
- практически не используется на практике

Оператор `goto`

```
for (...)
    for (...) {
        ...
        if (disaster) /* все плохо */
            goto error; /* перейти к обработке ошибки */
}
...
error:
    решить проблему
```