Операторы передачи управления

Оператор goto

Оператор безусловного перехода goto имеет формат:

goto метка;

В теле функции должна присутствовать ровно одна конструкция вида:

метка: оператор;

Оператор goto передает управление на помеченный оператор.

Оператор break

Оператор break используется внутри операторов цикла, if или switch для обеспечения перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится break.

Пример. Программа вычисляет значение гиперболического синуса вещественного аргумента x с заданной точностью eps с помощью разложения в бесконечный ряд.

$$sh\ x = x + x^3/3! + x^5/5! + x^7/7! + \dots$$

Вычисление заканчивается, когда абсолютная величина очередного члена ряда, прибавляемого к сумме, станет меньше заданной точности.

```
#include "stdafx.h"
#include <iostream>
#include <math.h>
#include <locale.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale( LC_ALL, "Russian" );
    const int MaxIter = 500; //ограничитель количества итераций
    double x, eps;
    cout << "\nВведите  аргумент и точность: ";
    cin >> x >> eps;
    bool ok = true;            //признак успешного вычисления
    double y = x, ch = x;      // сумма и первый член ряда
    for (int n = 0; fabs(ch) > eps; n++)  {
      ch *= x*x /(2*n + 2)/(2*n + 3); //очередной член ряда
        y += ch;
        if (n > MaxIter){
            ok = false; break;}
    }
    if (ok) cout << "\nЗначение функции: " << y << endl;
    else cout << "\nРяд расходится";
    system("PAUSE");
    return 0;
}
```

Оператор continue

Оператор перехода к следующей итерации цикла continue пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации.

Оператор return

Оператор возврата из функции return завершает выполнение функции и передает управление в точку ее вызова. Вид оператора:

```
return [ выражение ];
```

Выражение должно иметь скалярный тип. Если тип возвращаемого функцией значения описан как void, выражение должно отсутствовать.

Указатели

Указатели предназначены для храненения адресов областей памяти. В C++ различаю три вида указателей – указатели на объект, на функцию и на void.

```
int i;                          // целая переменная
const int ci = 1;               // целая константа
int * pi;                       // указатель на целую переменную
const int * pci;                // указатель на целую константу
int * const cp = &i;            // указатель-константа на целую переменную
const int * const cpc = &ci;    // указатель-константа на целую константу
```

Существуют следующие способы инициализации указателя:

1. Присваивание указателю адреса существующего объекта:
   - с помощью операции получения адреса:

     ```
     int a = 5;      // целая переменная
     int* p = &a;    // в указатель записывается адрес a
     int* p (&a);    // то же самое другим способом
     ```

   - с помощью значения другого инициализированного указателя:

     ```
     int* r = p;
     ```

   - с помощью имени массива или функции, которые трактуются как адрес

     ```
     int b[10];                  // массив
     int* t = b;                 // присваивание адреса начала массива
     void f(int a ){ /* … */ }   // определение функции
     void (*pf)(int);            // указатель на функцию
     pf = f;                     // присваивание адреса функции
     ```

2. *Присваивание указателю адреса области памяти в явном виде:*

   ```
   char* vp = (char *)0xB8000000;
   ```

3. *Присваивание пустого значения:*

   ```
   int* suxx = NULL;     int* rulez = 0;
   ```

4. *Выделение участка динамической памяти и присваивание ее адреса указателю:*

- с помощью операции new:

```
int* n = new int;                    // 1
int* m = new int (10);               // 2
int* q = new int [10];               // 3
```

- с помощью функции malloc[1]:

```
int* u = (int *)malloc(sizeof(int));     // 4
```

Операция разадресации (разыменования) (*)

Предназначена для доступа к величине, адрес которой хранится в указателе

```
char a;               // переменная типа char
char * p = new char;  /* выделение памяти под указатель и под динамическую
                         переменную типа char */
*p = '!0'; a = *p;    // присваивание значения обеим переменным
```

Операция получения адреса (&)

```
#include <stdio.h>
int main(){
   unsigned long int A = 0Xcc77ffaa;
   unsigned short int* pint = (unsigned short int*) &A;
   unsigned char* pchar = (unsigned char *) &A;
   printf(" | %x | %x | %x |", A, *pint, *pchar);
   return 0;
}
```

IBM PC-совместимый компьютер выведет на экран строку :

```
| cc77ffaa | ffaa | aa |
```

Ссылки

Ссылка представляет собой *синоним имени*, указанного при инициализации ссылки. Ссылку можно рассматривать как указатель, который всегда разыменовывается. Формат объявления ссылки:

```
тип & имя;
```

где тип — это тип величины, на которую указывает ссылка, & — оператор ссылки, означающий, что следующее за ним имя является именем переменной ссылочного типа, например:

```
int kol;
int& pal = kol;          // ссылка pal – альтернативное имя для kol
const char& CR = '\n';   // ссылка на константу
```

Массивы

Массив – конечная именованная последовательность однотипных величин

```
float a[10];   // описание массива из 10 вещественных чисел
```

Инициализация массива

```
int b[5] = {3, 2, 1}; // b[0]=3, b[1]=2, b[2]=1, b[3]=0, b[4]=0
```

Для доступа к элементу массива после его имени указывается номер элемента (*индекс*) в квадратных скобках. В следующем примере подсчитывается сумма элементов массива.

```
#include <iostream.h>
int main(){
   const int n = 10;
   int i, sum;
   int marks[n] = {3, 4, 5, 4, 4};
   for (i = 0, sum = 0; i<n; i++) sum += marks[i];
   cout << "Сумма элементов: " << sum;
   return 0;
}
```

Программа сортировки массива методом выбора

```
#include <iostream.h>
int main(){
const int n = 20;                    // количество элементов массива
int b[n];                            // описание массива
int i;
for (i = 0; i<n; i++) cin >> b[i];   // ввод массива
for (i = 0; i<n-1; i++){             // n-1 раз ищем наименьший элемент
   // принимаем за наименьший первый из рассматриваемых элементов:
   int imin = i;
   // поиск номера минимального элемента из неупорядоченных:
   for (int j = i + 1; j<n; j++)
   // если нашли меньший элемент, запоминаем его номер:
      if (b[j] < b[imin]) imin = j;
   int a = b[i];      // обмен элементов
   b[i] = b[imin];    // с номерами
   b[imin] = a;       // i и imin
}
// вывод упорядоченного массива:
for (i = 0; i<n; i++)cout << b[i] << ' ';
return 0; }
```

Динамические масивы

```
int n = 100;
float *p = new float [n];
```

```
int n = 100;
float *q = (float *) malloc(n * sizeof(float));
```