Введение в *C++*

Операции и операторы

Основные операции

Sizeof

формат: sizeof (<имя-типа>)
sizeof <имя-объекта>

примеры: sizeof(void*);
double* a;
sizeof a;

Арифметические

бинарные: +, -, *, /, %
унарные: +, -

примеры: unsigned n, m;
n % m;

операции возведения в степень в языке нет!

Отношения

бинарные: >, >=, <, <=, ==, !=
унарная: !

примеры: a==0 ⇔ !a

Логические связки

&& и || выражения с логическими связками вычисляются слева направо и вычисления останавливаются, когда результат становится ясен

Преобразования типа (кастинг)

формат: (<имя-типа>) <выражение>

примеры: long a, b;
(char*) (a+b);

Основные операции

Инкремент и декремент

формат: ++ инкремент -- декремент *имеют префиксную и постфиксную формы*

примеры:

```
int x, n=10;
x = ++n;    /* x = 11;  n = 11; */   x = --n;    /* x =9;   n = 9; */
x = n++;    /* x = 10;  n = 11; */   x = n--;    /* x = 10; n = 9; */
```

обработка массива *Pascal:*

```
s[n] := p[m];    n := n +1;    m := m +1;
```

обработка массива *Си:*

```
s[n++] = p[m++];
```

Побитовые

бинарные: &, |, ^, <<, >>

унарная: ~ инверсия

примеры:

таблица истинности:

x1	x2	&	|	^
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

```
int x = 07;
x << 3;        сдвиг влево
x >> 1;        сдвиг вправо
x = n & 017;   маскирование
x = n | 017;   установка битов
x = n ^ 017;   сброс единиц
~x;            инверсия битов
```

Основные операции

Присваивания

n = n + 1; ⇔ n++; i = i + 2; ⇔ i += 2; j = j * 3; ⇔ j *= 3;

Пусть @ - бинарная операция, тогда <e1> = <e1> @ <e2> ⇔ <e1> @= <e2>

например: a = a >> 2; ⇔ a >>= 2;

Косвенная адресация, получение адреса

Примеры: char x; ptr = &x; int a[10];

char *ptr; *ptr ⇔ x; a[i] ⇔ *(a+i)

*(&x) ⇔ x;

Условная операция

формат: <e1> ? <e2> : <e3> /* если <e1> истинно, тогда результат есть <e2> иначе <e3> */

пример: if (a > b) z = a;

else z = b; ⇔ z = (a>b) ? a : b;

Операция «запятая»

формат: <e1>, <e2>, <e3>, … , <en>

выражения вычисляются слева направо, результат операции есть значение последнего вычисленного выражения

Основные операторы

Условный оператор

формат: if (<выражение>)
<оператор-1>
[else
<оператор-2>]

примеры:

```
if (a)
    if (b)          a=1
        z = a;      b=0
    else            z=?
        z = b;
```

```
if ( g==0 )  …  ⇔  if ( !g )  …

If ( h!=0 )  …  ⇔  if ( h )  …
```

множественный выбор:

```
If ( выражение-1 )          <оператор-1>
else if (выражение-2 )      <оператор-2>
else if (выражение-3 )      <оператор-3>
……….
else if (выражение-n-1 )        <оператор-n-1>
else                        <оператор-n>
```

Если истинно (выражение-i), то выполняется <оператор-i> и вся работа заканчивается

Если ни одно из (выражений-i) не истинно, то выполняется <оператор-n>

Основные операторы

Операторы цикла

формат: while (выражение) <оператор> цикл с предусловием

do <оператор> while (выражение) с постусловием, используется редко

for (выражение-1 ; выражение-2 ; выражение-3) <оператор>

⇔

<выражение-1>;
while (выражение-2) {
<оператор>
<выражение-3>; }

оператор break вызывает немедленный выход из охватывающего его цикла

оператор continue завершает текущую итерацию и переходит к проверке условного выражения в операторах «while» и «do» и к вычислению (выражения-3) в операторе «for»

примеры:

```
for ( ; ; )   { …. }          бесконечный цикл
for ( i=0; i<n ; i++ )  ……    обработка массива в прямом направлении
for ( i=n; i--; )   ……        обработка массива в обратном направлении
int main() {              цикл почти всегда можно
    int c;                (но не всегда нужно) заменить рекурсией
    putchar ( c = getchar ( ) );
    if ( c != EOF ) main();  что делает эта функция main ?
}
```

Основные операторы

Операторы цикла. Примеры

Обработка только положительных элементов массива

```
for ( i=o; i<n; i++ ) {
    if ( a[ i ] < 0 ) continue;
    <обработка положительных элементов>
}
```

Нужен ли оператор go to ?

```
for ( … ) {                  например, для аварийного выхода
    for ( … ) {              из глубоко вложенных циклов?
        while ( … ) {
      if ( … ) goto error;   нет, не нужен, т.к. в таких ситуациях
} } }                        применяются объекты-исключения
```

Реверсирование строки на месте

```
reverse (char* s) {
int  c, n, m;                          два индекса бегут
for ( n=o, m=strlen(s) -1; n<m; n++, m--) {     навстречу друг другу
    c = s[n];    s[n] = s[m];   s[m] = c;  }     переставляем символы
}
```

Основные операторы

Переключатели

формат:

```
switch ( <целое-выражение> ) {
case <константа-1> : <оператор-1>
case <константа-2> : <оператор-2>
.....
case <константа-k> : <оператор-k>
default:             <оператор-n>   }
```

пример:

```
Подсчет цифр, пробелов и прочего
int digits, whites, others, c;
digits = whites = others = 0;
while ( ( c = getchar() ) != EOF )
    switch ( c ) {
        case '0' :
        case '1' :
         ..............
        case '9' :   digits++; break;     это цифра
        case ' ' :   whites++; break;     это пробел
        default :    others++;            это прочее
    }
```

Основные операторы

Перечислимые типы

формат: enum <имя-типа> { <список-перечисления> }
enum <имя-типа> <описания-переменных>
enum <имя-типа> { < список-перечисления > } <описания-переменных>

где:

{ <список-перечисления> } есть { <имя-1> [= <конст.-выражение-1>],
<имя-2> [= <конст.-выражение-2>],
.....
<имя-n> [= <конст.-выражение-n>] }

пример:

Манипуляции с цветами спектра
enum spectrum { Red, Orange, Yellow = 10, Green, Blue, Violet }; описали тип
enum spectrum color; описали переменную
color = Green + 2 * Orange; вычислили цвет
Чему равен color? Посчитаем: 11 + 2 = 13 (Violet)

Использование перечислимых типов не обязательно,
но повышает надежность и улучшает читабельность кода