Операторы часть 2

Унарная операция присваивания:
вместо записи **x = x # 1;**
где **#** – символы, обозначающие операцию ***инкремента*** (+), либо ***декремента*** (–),
x – переменная **арифметического типа** или типа **указатель**, рекомендуется использовать запись:
##x; – префиксную
x##; – постфиксную
Если унарная операция используется
1. **в чистом виде**, то различий между постфиксной и префиксной формами нет.
2. **в выражении**, то в ***префиксной форме*** (##x) сначала значение переменной изменится на 1, а затем новое значение используется в выражении;

в постфиксной форме (x##) сначала значение переменной используется в выражении, а после вычисления выражения значение переменной изменяется на 1.
Пример использования **постфиксной** операции:

```
int x = 5, y;
y = x++;
```

Пример использования **префиксной** операции:

```
int x = 5, y;
y = ++x;
```

Примеры:

```
1. int b = 7, n = 1, c;
       c = b * ++n;
 //n = n+1,  c = b*n   ↔ c = 14 n=2 b=7

2.     int b = 7, n = 1, c;
       c = b * n++;
// c = b*n,  n = n+1   ↔ c = 7 n=2 b=7

3. int b = 6, n , a = 3;
       n = (--a)*10 - a * (b++);
// a = 2   n = 8   b = 7
```

НЕЯВНОЕ ПРЕОБРАЗОВАНИЕ ТИПОВ ОПЕРАНДОВ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ
Неявные преобразования всегда производится от более ***«маленького» типа к более «большому»*** по следующие правила:
1. операнды типов **char**, **short** и **enum** всегда преобразуются в **int**;
2. операнды типов **unsigned char** и **unsigned short** всегда преобразуются в **unsigned int**;
3. операнды типа **float** всегда преобразуются в тип **double.**
4. В арифметическом выражении **тип результата выражения** определяется самым "большим" типов среди всех образующих выражение операндов. К этому типу преобразуются все остальные операнды.

- unsigned int x = 5;
- …(x + 3.14159)…

5. Преобразование типа при вычислениях арифметических выражений **применяется к копиям значений образующих выражение**.

6. **Информация теряется** если результат арифметического выражения **не может быть представлен в типе операнда после преобразования**.

- unsigned int x = 5;
- x = x + 3.14159;

Результатом 1/3 будет «0», чтобы избежать такого рода ошибок необходимо явно изменять тип хотя бы одного операнда, т.е. записывать, например: 1.0/ 3

```
int x = 5, y = 2;
double z;
z = x / y;
printf("%d / %d = %.2lf\n", x,y,z);
```

Операторы организации циклов

Цикл – это одно из фундаментальных понятий программирования. Под циклом понимается организованное *повторение некоторой последовательности операторов*.
Для организации циклов используются специальные операторы:

– оператор цикла с предусловием **while**;
– оператор цикла с постусловием **do – while**;
– оператор цикла с предусловием и коррекцией **for.**

Любой цикл состоит из кода цикла, т.е. тех операторов, которые выполняются несколько раз, начальных установок, модификации параметра цикла и проверки условия продолжения выполнения цикла.
Один проход цикла называется итерацией. Проверка условия выполняется на каждой итерации либо до кода цикла (с предусловием), либо после кода цикла (с постусловием).

Оператор с предусловием while
*Общий вид записи***:**

```
выражение1;
while (выражение-условие) {
    код цикла
    выражение2;
}
```

Выражение1 – задает начальные условия для цикла (инициализация, т.е. начальная установка).
Выражение - условие **–** определяет условие выполнения цикла, если оно не равно 0, цикл выполняется**.**
Выражение2 **–** задает изменение параметра цикла или других переменных (шаг или модификация параметра).
Код цикла может включать любое количество управляющих операторов, связанных с конструкцией *while*, взятых в фигурные скобки (блок), если их более одного. Среди этих операторов могут быть *continue* – переход к следующей итерации цикла и *break* – выход из цикла.

Оператор цикла с постусловием do – while
*Общий вид записи***:**

```
do{
    код цикла
    шаг цикла;
} while (выражение-условие);
```

Выражение - условие – определяет условие выполнения цикла, если оно не равно 0, цикл выполняется.
Шаг цикла– задает изменение параметра цикла или других переменных .
При работе данного оператора сначала выполняется *код цикла*, а затем проверяется *выражение-условие.*
Выполнение кода цикла происходит до тех пор, пока **выражение-условие не примет значение ложь**.
При использовании оператора цикла с постусловием тело цикла выполняется *хотя бы один раз*.

Оператор цикла с предусловием for

Общий вид оператора:

```
for (выражение 1; выражение-условие; выражение 2)
                код цикла;
```

Выражение 1 – задает начальные условия для цикла (инициализация).
Выражение - условие – определяет условие выполнения цикла, если оно не равно 0, цикл выполняется, а затем вычисляется значение выражения2.
Выражение 2 – задает изменение параметра цикла или других переменных (шаг).
выражение1 и **выражение2** могут состоять из нескольких выражений, разделенных запятыми.
Цикл продолжается до тех пор, пока **выражение-условие** не станет равно 0.
Любое выражение может отсутствовать, но разделяющие их « ; » должны быть обязательно.

Примеры:

1. Уменьшение параметра:
   for (n=10; n>0; n--){ операторы; }

2. Проверка условия отличного от того, которое налагается на число итераций:
   for (num=1; num*num*num<216; num++) { операторы; }

3. Шаг с помощью умножения:
   for (d=100.0; d<150.0;d*=1.1) { операторы; }

4. Шаг с помощью арифметического выражения:
   for (x=1; y<=75; y=5*(x++)+10) { операторы; }

5. Использование несколько инициализирующих или корректирующих выражений:
   for (x=1, y=0 ; x<10 ; x++, y += x) { операторы; }

6. Бесконечный цикл
   for (; ;) { операторы; }

Операторы передачи управления

Формально к операторам передачи управления относятся:
- оператор безусловного перехода **goto**;
- оператор перехода к следующему шагу (итерации) цикла **continue**;
- выход из цикла или оператора **switch** – **break**;
- оператор возврата из функции **return**.

Оператор безусловного перехода goto
В языке Си предусмотрен оператор **goto**, хотя в большинстве случаев можно обойтись без него.

Общий вид оператора

goto метка;

Он предназначен для передачи управления на оператор, помеченный меткой. Метка представляет собой идентификатор, оформленный по всем правилам идентификации переменных с символом «двоеточие» после него, например, пустой помеченный оператор:

метка : ;

Область действия метки – функция, где эта метка определена.
Программа с **goto** может быть написана без него за счет повторения некоторых проверок и введения дополнительных переменных.
Наиболее характерный случай использования оператора **goto** – выполнение прерывания (выхода) во вложенной структуре при возникновении грубых неисправимых ошибок во входных данных. И в этом случае необходимо выйти из двух (или более) циклов, где нельзя использовать непосредственно оператор **break**, т.к. он прерывает только самый внутренний цикл:

```
for (...)
   for (...) {  ...
         if ( ошибка )  goto Error;
     }
        ...
Error :    – операторы для устранения ошибки;
```

Если программа обработки ошибок сложная, а ошибки могут возникать в нескольких местах, то такая организация оказывается удобной.

Оператор continue
Этот оператор может использоваться во всех типах циклов. Наличие оператора **continue** вызывает пропуск "оставшейся" части итерации и переход к началу следующей, т.е. досрочное завершение текущего шага и переход к следующему шагу.

В циклах **while** и **do** это означает непосредственный переход к проверочной части. В цикле **for** управление передается на шаг коррекции, т.е. модификации выражения2.
Что выполняют следующие фрагменты кода?
1.

```
for( int x = 1; x <= 10; x++)
{
      if(x == 5)  continue;
      printf("%d ",x);
}
```

2.

```
int x = 1;
while( x <= 10) {
      if( x == 5)       continue;
      printf("%d ", x);
      x++;
}
```

Оператор break
Оператор ***break*** **–** пропускает оставшуюся часть структуры **switch** и производит альтернативный выход из самого внутреннего цикла, то есть переходит к первому оператору, следующему за текущим оператором цикла. Заметим, что «покинуть» одновременно несколько вложенных друг в друга циклов при помощи **break** не удается.

Найти сумму чисел, числа вводятся с клавиатуры до тех пор, пока не будет введено 100 чисел или 0.

```
for(s = 0, i = 0; i < 100; ++i){
      scanf("%d", &x);
      if( !x ) break;// если ввели 0, то суммирование
                     // заканчивается.
      s += x;
}
```

Оператор return
Оператор ***return*** – оператор возврата из функции. Он всегда завершает выполнение функции и передает управление в точку ее вызова.

Вид оператора:
return [выражение];