Control Flow II

Arash Rafiey

September 5, 2017

for statement

```
for (initialization; condition; increment_decrement)
   statement; // Loop body .
```

For example, to sum **factorials** from 1 to 5:

```
const unsigned long long int maxValue = 5;
unsigned long long sum, value, factorial = 1;

// There can be several initialize statements.
for (value = 1, sum = 0; value <= maxValue; ++value)
{
   // Watch for overflow for big numbers.
   factorial *= value;
   sum += factorial;
}

// Here sum = 153, value = 6, factorial = 120.
```

while statement

```
while (expression)
    statement;
```

Execution order: If the expression evaluates to true, the statement is executed. The loop **starts over** unless:

1. either expression becomes false
2. or break or similar statement stops the loop.

Finally, the execution resumes after the "statement".

These two are equivalent:

1.
   ```
   for (initialization; condition; increment_decrement)
       statement;
   ```

2.
   ```
   initialization;
   while (condition) {
       statement;
       increment_decrement;
   }
   ```

Example for while

For example, to sum factorials from 1 to 5:

```
int maxValue = 5;
unsigned long int sum=0, value=1, factorial=1;
while (value <= maxValue) {
   factorial *= value;
   sum += factorial;
   value ++;
}
// Here sum = 153, value = 6, factorial = 120.
```

Computing x^y

```
void main() {
unsigned long int Total-Val = 1;
int x, y ;
printf(”enter two non-negative integer numbers x,y \n ”);
scanf( “%d %d”,&x, &y);
int i=1;
while (i <= y) {
   Total-Val *= x;
   i++;
}
}
```

do-while statement

```
do
   statement;
while (condition); //Note the semicolon.
```

Unlike while and for, the do-while evaluates the condition **after** each passing through the loop body.
That is, the "statement" is always executed **at least once**.
Then the condition is evaluated. If it is true, the statement is run again, and so on.
When the condition becomes false, the **loop terminates**.

Recommended usage rules:

1. When the loop must be run at least once, do-while is faster.
2. When there is an initialization, use for, otherwise while.

Example of do-while statement

For example, to reverse a number:

```
int n=213, rev=0; // n is the number to be reversed.
do {
   rev = rev *10;
   rev = rev + n%10;
   n = n/10;
} while (n!=0);
```

Break and continue

The break statement is used to **exit** immediately from **innermost**:

1. for
2. while
3. do-while

Break and continue

The break statement is used to **exit** immediately from **innermost**:

1. for
2. while
3. do-while

In loops, continue passes the control to the next iteration.

```
int i, x=25, length=10;
for (i = 2; i < length; i++) {
  if ( ! x % i ) // (x % i==0 )
    break; // Stop when zero is encountered.
  if ( x/i < 3)
    continue; // Skip negative elements.
}
printf(“ %d \n”, i);
```

Goto and labels

goto is often related to **“spaghetti“ code**.

Rarely there is a situation when goto makes perfect sense: Breaking out of **many loops** since break exits from innermost loop.

Label follows the rules for identifier names. It is followed by a colon **:** and can be attached to the beginning of any statement.

The **scope** of a **label** is the function where label is defined.

```
bool found = false;
for (i = 0; i < 13; ++i)
   for (j = 0; j < 27 ; ++j)
      if (i % 5 == j % 3) {
         found = true;
         goto FoundMatch;
      }
FoundMatch: if (found) {
   printf(“ %d %d \n”, i,j);
}
```

Infinite loop

Infinite loop implementations:

1.
   ```
   for (;;) {
       statement
   }
   ```

2.
   ```
   while (1)
       statement
   ```

3.
   ```
   do
       statement
   while (123)// Any non zero value will fit.
   ```

4.
   ```
   SOME_LABEL:
   statement
   goto SOME_LABEL;
   ```

Infinite loop is typically broken by break, return or similar.

Example: read some values from input and calculate their average, if an input value is zero then terminate

```
int i=0, value,
float sum=0.0;
while (1) {
   scanf(”%d”, &value);
   sum+=value;
   i++;
   if ( ! value )
     break;
}
printf(“ %f \n”, sum/ i);
```

Quiz

1) Write a program to read a number *n* from input and print out the following

1,2,3,..........,n-1,n
1,2,3,........,n-2,n-1
1,2,3,.....,n-3,n-2
.
.
.
1,2
1

Solution to number 1

```
# include < stdio.h >
int main() {
int i,j,n;
printf(”enter an integer \n”);
scanf(“%d”,&n);
for (i = 1; i <= n; i + +) {
   for (j = 1; j <= i; j + +)
     printf(” %d ”,j);
   printf(” \n ”);
}
}
```

2) Write a program to read a positive integer n from user and
a) prints out the number of digits of n .
b) how many of these digits are even (0,2,4,6,8) .

For example if n=3567
Then the out put of your program is :
a) 4
b) 1 (6 is even)

Solution to number 2

```
# include < stdio.h >
int main() {
int i,n;
int count=0;
int count-even=0;
printf(”enter an integer \n”);
scanf(“%d”,&n);
while ( n > 0) {
   count++;
   if ( n % 2 ==0 )
     count-even++;
   n= n/10;
}
printf(”number of digits, number of even digits %d %d \n”, count,
count-even);
}
```