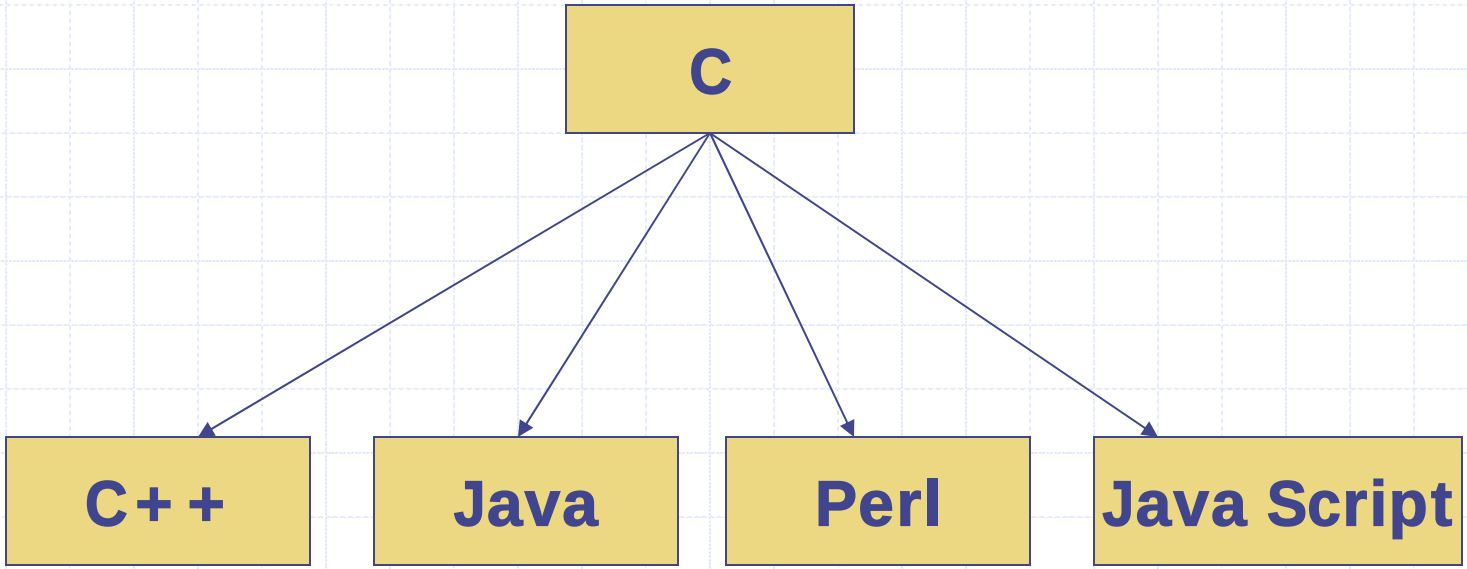
Repetition Statements in C / C++

JIM ENGEL

The Scope

- C++ is a superset of C
- The repetition statements such as **for**, **while** and **do/while** are all constructs of the original C language and thus essentially carry over into C++.
- There are a couple of minor but potentially hazardous differences, which will be pointed out toward the end.

The Wellspring



Looping

- Execution of a group of statements in a **"loop"** is a fundamental computer operation.
- Loops can be executed without the repetition statements by means of comparisons, statement labels and the dreaded goto statement.

```
int j = 0, sum = 0 max = 10;
LOOP:                                   // a “bad style” example
        sum + =J;
        if ( ++j < = max )
                goto LOOP;
```

Inconvenient & error prone !

The “for” Loop

General form:

for (***initializing_expr; test_expr; increment_expr*** **)**

{

loop_statements**;**

}

Example:

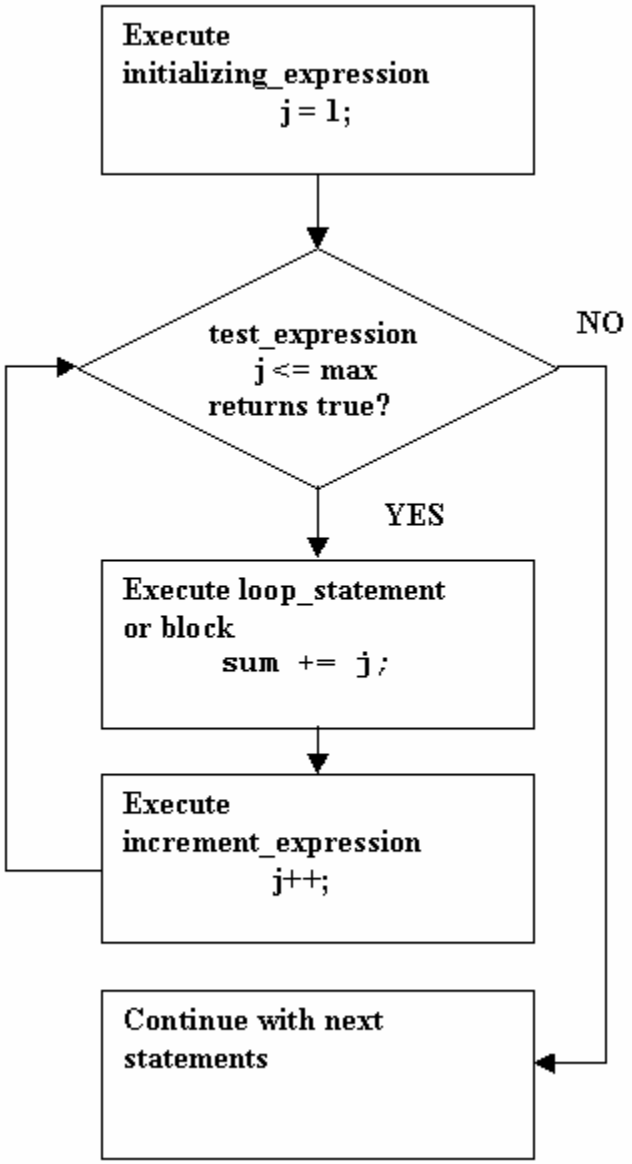
```
int  j, max =12,  sum = 0;
for ( j = 1; j <= max; j++)  {
sum += j;
}
```

for *is a reserved word.*

```
for (    initializing_expr;
         test_expr;
         increment_expr )
    {
         loop_statements;
    }
```

Our Example:

```
int j, max =12, sum = 0;
for (j = 1; j <= max; j++)
{
     sum += j;
}
```



Alternate Example:

```
int  j, max,  sum;
for ( j = 1,max= 12,sum= 0; j <= max; j++)
   sum += j;
```

- Expressions separated by comma operators are evaluated left to right. Type and value of result is type and value of the rightmost operand.

- Note: If there is only one loop statement, the curly brackets are optional.

More about the for loop

Any of the fields in the for loop can be left blank:

Fragment One

```
j = 1;
for (   ; j <= max; j++)  {
    sum += j;
}
```

Fragment Two

```
j = 1;
for (   ; j <= max;    )  {
    sum += j;
   ++j;
}
```

An infinite loop !

```
j = 1;
for (   ;   ;    )  {    //  runs forever !
sum += j;
     ++j;
}
```

This one works:

```
j = 1;
for (   ; TRUE  ;    )  { // a reminder.
    sum += j;
    if ( ++j > max )
            break;              //  The break statement
}
```

The break statement is covered in detail later...

The "while" Loop

General form:

while (*test_expression* **)** {

    loop_statements;

}

Again, ***while*** *is a reserved word.*

Notice that these are equivalent :

```
initializing_expression;
while (   test_expression   )
{
      loop_statements;
      increment_expression;
}
```

```
for ( initializing_expression;  test_expression;  increment_expression )
{
       loop_statements;
}
```

The while loop is just a convenient simplification of the general case of the for loop.

Use of Auto-increment / Auto-decrement

```
int count = 20;
 while ( count-- > 0)
 {
     hexpull ( target, ib++, 2 );
     target += 2;
     *target++ = ' ';
 }
```

Auto-increment:

++count	**means add one and then use**
count++	**means use and then add one**

Auto-decrement:

--count	**means subtract one and then use**
count--	**means use and then subtract one**

Notice that this will execute for the 0 case, perhaps

```
while ( --count >= 0 )
```

would be more clear.

Often the comparison is omitted:

```
int count = 20;
while ( count--  )                          {
     hexpull ( target, ib++, 2 );
     target += 2;
     *target++ = ' ';
}
```

The "Do/While" Loop

General form:

```
do
{
    loop statements;
}   while ( condition) ;
```

Primarily used where the loop must execute once.

Example:

```
char iar[1024];
do {
        readline ( iar, 100, finp);
} while ( *iar == '*' ) ;
```

Both **do** *and* **while** *are reserved words*

Do/while is a less frequently used mechanism, in one large program there are:

200	while loops
4	do while loops
~250	for loops

Nested Loop Example

```
#include "stdafx.h"
#include "stdio.h"
int main(int argc, char* argv[])
{
   int value [4][5]          //   generally bad practice!
      ={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},
        {16,17,18,19,20}};
   int row, col;

   printf ("Our array values:\n");
   for ( row = 0; row < 4; ++row )      {
      for ( col = 0; col < 5; ++col )   {
         printf ("%4d", value[row][col]);
      }
      printf ("\n");                          //new line
   }
   printf ("\nFinished\n\n");
   return 0;
}
```

The Actual Results:

```
Our array values:
    1   2   3   4   5
    6   7   8   9  10
   11  12  13  14  15
   16  17  18  19  20

Finished
```

Breaking out!

Although the normal loop exit occurs when the condition test comes up false, there are methods to break out:

- break statements.
- goto statements (Not usually the best method, but sometimes appropriate.)

When the break statement executes, the current loop execution is suspended and control passes to the next executable statement beyond the block or statement.

Very important:

In a nested loop situation, only the inner most loop is broken out of.

(Exiting from deeply nested loops on an error condition is by tradition one of the few accepted uses of the goto statement.)

Break Statements

Example:

```
char  iar[1024];
int   return_value;
do {
    return_value  = readline ( iar, 100, finp);
     if ( return_value != 0)
          break;
}   while ( *iar == '*' ) ;

if (return_value != 0) {
    bprint ("Looks like a file read error!");
}
```

The Continue Statement

- On occasion it is convenient to complete a loop iteration in the midst of the code, often because an unusual situation has occurred.

- The **continue** statement is similar to the break, but it completes the iteration, that is, skips to the end of the loop block, and then goes on to the next iteration.

Continue example

```
for ( j = 0; j < n; j++  )   {
  if (  ary[j]  < 0 )
        continue;

     ……  /* Process positive elements */
}
```

Loops with "Other Types"

Most loops use integers for the loop iteration variables,but other types are also sometimes appropriate. For example, this fragment simply prints out the alphabet:

```
char ch;
for ( ch = ‘A’; ch <= ‘Z’;  ++ch  )   {
   printf ("%c", ch );
}
```

Alpha-Numeric Values

```
char ch;
for ( ch = 'A'; ch <= 'Z';  ++ch  )   {
   printf ("%4d %c\n", ch, ch );
}
```

Output:

```
  65 A
  66 B
  67 C
  …….
  87 W
  88 X
  89 Y
  90 Z
```

C++ vs. C

```
for ( int i=0; i <  50; )  // invalid for "C",
                           // valid for C++
```

Warning: when the index i is declared this way, it may not exist once the loop is complete.

Final Comments

- You can jump out of a loop, but do not jump into a loop.
- There is not much temptation here, since the goto is the only obvious way to do this. (This is a big advantage of the structured programming block structure, it was much easier to make a mistake in an older language such as Fortran.)

Advice

- Loops, and especially nested loops, can become very long, extend over many pages of code. This can be very difficult to follow.
- Try and keep loops to a page or less by creating appropriate functions.
- Be very careful with you indentation