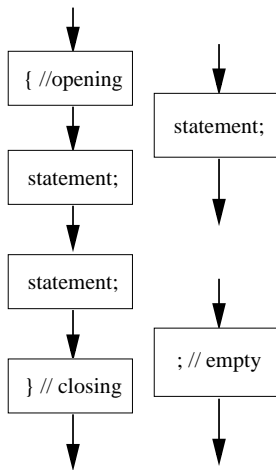


C++ Looping Structures

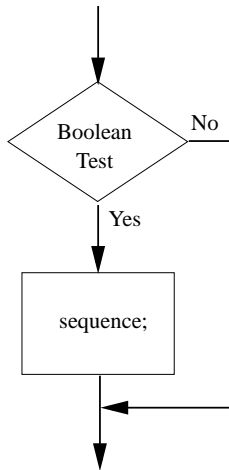
July 6, 2010

Flowchart Notation

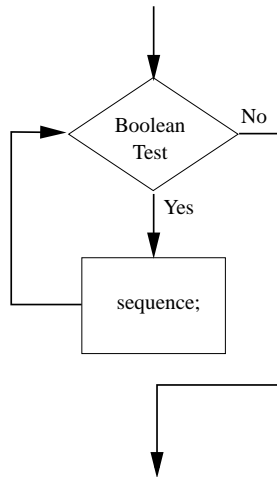
Sequence



Selection

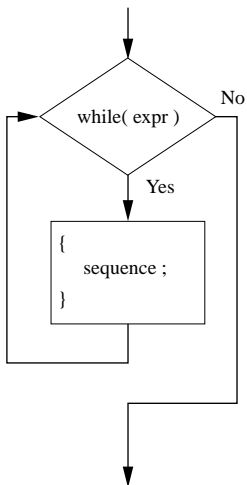


Repetition



Repetition Statements

The while Loop



```
1 // input is an empty string ,
2 // ... not a garbage value
3 string input;
4 const string THE_END( "end" );
5 // count number of one-letter "words"
6 // we've read,
7 int one_letter(0);
8
9 while( input != THE_END ) {
10     cout << "Enter 'end' to stop:" << endl;
11     cin >> input;
12     if( input.size() == 1 ) one_letter++;
13 }
14
15 cout << "Stopped! after reading"
16     << one_letter << " one-letter words."
17     << endl;
```

The while Loop

```
1 // input is an empty string ,
2 //   ... not a garbage value
3 string input;
4 const string THE_END( "end" );
5 // count number of one-letter "words"
6 // we've read,
7 int one_letter(0);
8
9 while( input != THE_END ) {
10     cout << "Enter 'end' to stop:" << endl;
11     cin >> input;
12     if( input.size() == 1 ) one_letter++;
13 }
14
15 cout << "Stopped!_after_reading_"
16     << one_letter << "_one-letter_words."
17     << endl;
```

<<Interactive Program>>

RUN

EDIT

while_example.cxx

The while Loop

```
1 // input is an empty string ,
2 //   ... not a garbage value
3 string input;
4 const string THE_END( "end" );
5 // count number of one-letter "words"
6 // we've read,
7 int one_letter(0);
8
9 while( input != THE_END ) {
10     cout << "Enter 'end' to stop:" << endl;
11     cin >> input;
12     if( input.size() == 1 ) one_letter++;
13 }
14
15 cout << "Stopped! after reading "
16     << one_letter << " one-letter words."
17     << endl;
```

<<Interactive Program>>

RUN

EDIT

while_example.cxx

one_letter is called a **flag variable**.

- ▶ Flag variables are declared and **initialized before** a loop.
- ▶ ... **updated within a loop** (perhaps conditionally by `if()`).
- ▶ ... **inspected after** a loop completes.

The while Loop

```
1 // input is an empty string ,
2 // ... not a garbage value
3 string input;
4 const string THE_END( "end" );
5 // count number of one-letter "words"
6 // we've read,
7 int one_letter(0);
8
9 while( input != THE_END ) {
10     cout << "Enter 'end' to stop:" << endl;
11     cin >> input;
12     if( input.size() == 1 ) one_letter++;
13 }
14
15 cout << "Stopped! after reading "
16     << one_letter << " one-letter words."
17     << endl;
```

<<Interactive Program>>

RUN

EDIT

while_example.cxx

1. What is the initial value of input?
2. What condition must be met for the **looping sequence** to execute?
3. How is input changed in the looping sequence? What happens if it **doesn't** change?

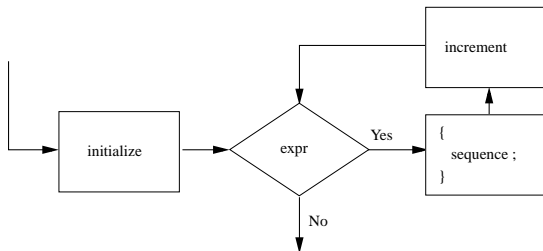
The for Loop

Recall the three important parameters governing a while loop's execution:

1. What is the initial value of the **looping variable**?
2. What condition must be met (usually dependent on the looping variable) for the **looping sequence** to execute?
3. How is the **looping variable** changed in the looping sequence?

A for loop puts these three critical pieces of information in **one place**, at the top of the loop, where they are easily written, verified, and commented.

The for Loop



```

1 int i;
2 for( i='A'; i<='Z'; i++ ) {
3     cout << char(i);
4     if( !( i % 12 ) ) {
5         cout << endl;
6     }
7 }
8 cout << "i_is_" << char(i) << endl;
  
```

```

ABCDEF
GHIJKLMNOP
QRSTUVWXYZ
i_is_ [
  
```

RUN

EDIT

for_all_caps.cxx

The `i` is called the **loop counter** or **looping variable**.

The for Loop

```
1 for( int i='z'; i>='a'; i-- ) {  
2     cout << char(i);  
3     if( !( i % 12 ) ) {  
4         cout << endl;  
5     }  
6 }  
7 cout << endl;
```

```
zyx  
wvutsrqponml  
kjihgfedcba
```

RUN

EDIT

for_all_lower.cxx

Note that the looping counter may be declared *inside the for loop*.

This is the preferred style.

The for Loop

```
1 for( int l('z'), u('A'); \
2     l>='a' && u<'Z' ; l-=2, u+=1 ) {
3     cout << char(l) << char(u);
4     if( !( u % 12 )) {
5         cout << endl;
6     }
7 }
8 cout << endl;
```

```
zAxBvCtDrEpFnGlH
jIhJfKdLbM
```

RUN

EDIT

for_half_letters.cxx

The for Loop

```
1 for( int l('z'), u('A'); \
2     l>='a' && u<'Z' ; l-=2, u+=1 ) {
3     cout << char(l) << char(u);
4     if( !( u % 12 )) {
5         cout << endl;
6     }
7 }
8 cout << endl;
```

```
zAxBvCtDrEpFnGlH
jIhJfKdLbM
```

RUN

EDIT

for_half_letters.cxx

Now Try Lecture Question 3

The for Loop's Sibling

What Happens Here?

```
1 for( int i=0; i < 0; i++ ) {  
2     cout << "In_the_for()_loop." << endl;  
3 }  
4 cout << "Out_of_the_for()_loop." << endl;
```

The for Loop's Sibling

What Happens Here?

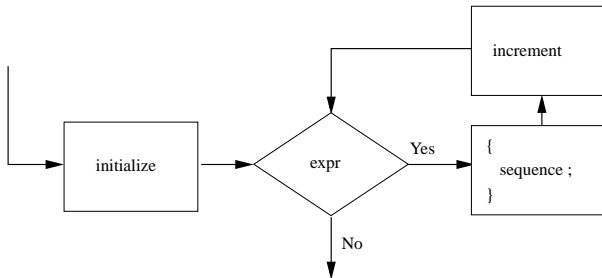
```
1 for( int i=0; i < 0; i++ ) {  
2     cout << "In_the_for()_loop." << endl;  
3 }  
4 cout << "Out_of_the_for()_loop." << endl;
```

Out of the for() loop.

RUN

EDIT

for_initial_eval.cxx



The for Loop's Sibling

for() \iff while()

```
1 for( int i=0; i < 0; i++ ) {  
2     cout << "In_the_for()_loop." << endl;  
3 }  
4 cout << "Out_of_the_for()_loop." << endl;
```

```
1 int i=0;  
2 while( i < 0 ) {  
3     cout << "In_the_for()_loop." << endl;  
4     i++;  
5 }  
6 cout << "Out_of_the_for()_loop." << endl;
```

Rewriting Loops

```
1 char input('a');
2 cout << "Enter a lowercase character:_" << flush;
3
4 for( cin >> input; input <= 'q'; input++ ) {
5     cout << input;
6     if( !( input % 10 ) ) {
7         cout << endl;
8     }
9 }
10
11 cout << endl << "Past looping structure._" << endl;
```

1. Rewrite this for loop as a while() loop.

Rewriting Loops

```
1 char input('a');
2 cout << "Enter a lowercase character: " << flush;
3
4 for( cin >> input; input <= 'q'; input++ ) {
5     cout << input;
6     if( !( input % 10 ) ) {
7         cout << endl;
8     }
9 }
10
11 cout << endl << "Past looping structure." << endl;
```

As a while() loop:

```
1 char input('a');
2 cout << "Enter a lowercase character: " << flush;
3 cin >> input;
4
5 while( input <= 'q' ) {
6     cout << input;
7     if( !( input % 10 ) ) {
8         cout << endl;
9     }
10    input++;
11 }
12
13 cout << endl << "Past looping structure." << endl;
```

1. Rewrite this for loop as a while() loop.

The `break`; Keyword

`break`; Jumps to the first statement *outside* of the innermost loop and continues on.

In general, you can think of *breaking out* of the current loop .

break; Example

Where does the innermost loop begin and end?

```
1 int i;
2 char console_read;
3 for( ; ; ) {
4     cout << "Top_of_oo_for-loop." << endl;
5     while( true ) {
6         cout << "Continue_loop_with_a_"
7             << "positive_integer:_" << flush;
8         cin >> i;
9         if( i <= 0 ) {
10            // this is the only way out.
11            break;
12        }
13        cout << "Round_and_round_we_go!"
14            << endl;
15    }
16
17    cout << "Out_of_the_while_loop:_"
18    cout << "'Q'_to_quit:_" << flush;
19    cin >> console_read;
20    if( console_read == 'Q' ) {
21        break;
22    }
23 }
```

<<Interactive Program>>

RUN

EDIT

break_example.cxx

Now Try Lecture Question 4.

User Input Loops

```
1 double input(0);
2
3 while( true ) {
4     cout << "Enter a decimal within "
5         << "(0,1]: " << endl;
6     cin >> input;
7     if( input>0 && input<=1 ) break;
8 }
9
10 cout << "Thanks , " << input <<
11     " is within (0,1]. " << endl;
12 /**
13  * note the spacing for readability in
14  * if predicate
15  */
```

Here is the general pattern used when needing to check user input against required values or ranges.

You should know this pattern!

User Input Loops

```
1 double input(0);
2
3 while( true ) {
4     cout << "Enter a decimal within "
5         << "(0,1):" << endl;
6     cin >> input;
7     if( input>0 && input<=1 ) break;
8 }
9
10 cout << "Thanks , " << input <<
11     " is within (0,1]." << endl;
12 /**
13  * note the spacing for readability in
14  * if predicate
15  */
```

<<Interactive Program>>

RUN

EDIT

while_prompts.cxx

Now Try Lecture Question 5

Looping Patterns

There are three fundamental *looping patterns* in programming (not just C++).

Counted Loops Loop for a pre-determined number of times.

Conditional Loops Loop until a particular *condition* occurs.

Sentinel Loops Loop until a particular *value* is encountered.

Counted Loops

Think of for-loops First!

```
1 int i;
2 for( i='A'; i<='Z'; i++ ) {
3     cout << char(i);
4     if( !( i % 12 ) ) {
5         cout << endl;
6     }
7 }
8 cout << "i_ is_" << char(i) << endl;
```

Counted Loops The number of loop iterations are known *before* the first iteration begins.

Sentinel Loops

```
1 // input is an empty string ,
2 // ... not a garbage value
3 string input;
4 const string THE_END( "end" );
5 // count number of one-letter "words"
6 // we've read,
7 int one_letter(0);
8
9 while( input != THE_END ) {
10     cout << "Enter 'end' to stop:" << endl;
11     cin >> input;
12     if( input.size() == 1 ) one_letter++;
13 }
14
15 cout << "Stopped! after reading "
16     << one_letter << " one-letter words ."
17     << endl;
```

Waiting for a string *value* of “end”.

Conditional Loops

Waiting for a range of variable values.

```

1 double input(0);
2
3 while( true ) {
4     cout << "Enter a decimal within "
5         << "(0,1]: " << endl;
6     cin >> input;
7     if( input>0 && input<=1 ) break;
8 }
9
10 cout << "Thanks , " << input <<
11     " is within (0,1]. " << endl;
12 /**
13  * note the spacing for readability in
14  * if predicate
15  */

```

These are *conditions*, there is no particular “value” to them.

We still test them by choosing special values (endpoints) and reducing the condition to a Boolean value.

finis