CS2313 Computer Programming

LT5 – Loop Statements

Outlines

- **while** and **do-while** loop
- **for** loop

Outcomes

- Be able to express repeating task with a **loop** statement in C++.

- Clearly define the loop **condition** and loop **body**.

Syntax Summary

- Keywords
  - `while`, `do`, `for`.

Motivations

Suppose that you need to print a string (e.g., "Welcome to C++!") a hundred times. It would be tedious to have to write the following statement a hundred times:

```
cout  << "Welcome to C++!" << endl;
```

So, how do you solve this problem?

Opening Problem

Problem:

100 times

```
cout << "Welcome to C++!" << endl;
cout << "Welcome to C++!" << endl;
cout << "Welcome to C++!" << endl;
cout << "Welcome to C++!" << endl;
cout << "Welcome to C++!" << endl;

…
…
…
cout << "Welcome to C++!" << endl;
cout << "Welcome to C++!" << endl;
cout << "Welcome to C++!" << endl;
cout << "Welcome to C++!" << endl;
cout << "Welcome to C++!" << endl;
```

Introducing while Loops

```
int count = 0;
while (count < 100)
{
  cout << "Welcome to C++!\n";
  count++;
}
```

Loop

- Beside **sequential** execution and **conditional (branch)** execution, **looping** is another common control flow in programming.

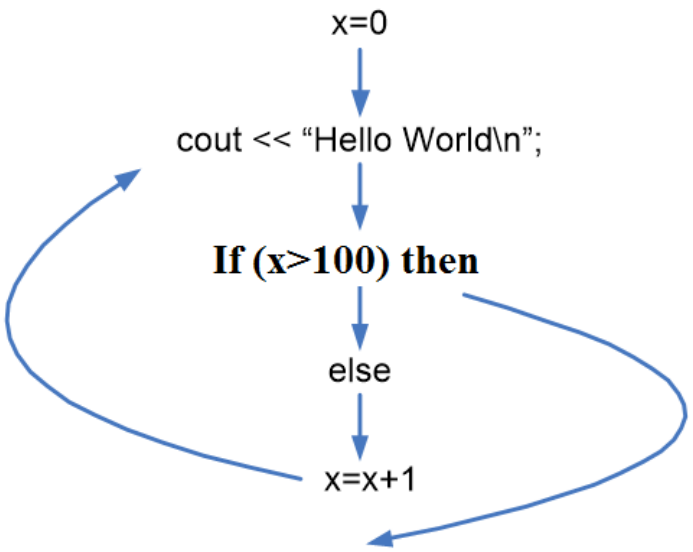
- When the execution enters a **loop**, it will execute a block of code **repeatedly** until the **exit loop condition** is met.

Loop

- In general , a program loop consists:
  - Loop **Initialization** statements.
  - Looping **Body**.
  - **Exit** Loop condition.
  - **Post** loop statements.

Loop

- Set x=0.
- Print out “Hello World\n”.
- If x is greater than 100, exit the loop.
- Add 1 to x.
- Loop back.



`while` Statement

- General form of **`while`** statement (*repetition* statement)

```
while (logical expression) {
    statements
}
```

- Semantics:
  - `logical expression` is evaluated first.
  - If `true`, `statements` in the loop body are executed.
  - Then `logical expression` is evaluated again and if still evaluated to `true`, the loop repeats; otherwise, the loop terminates.

Example of `while` Statement

```
#include <iostream>
using namespace std;

void main() {
   int     cnt = 0, n;
   float   max, x;

   cout << "The maximum value will be computed.\n";
   cout << "How many numbers do you wish to enter?  ";
   cin >> n;

   while (n <= 0)  /* ensure a positive number is entered */
   {
      cout << "\nERROR: Positive integer required.\n\n";
      cout << "How many numbers do you wish to enter?  ";
      cin >> n;
   }
/* To be continued in next page */
```

Example of `while` Statement

```
    cout << "\nEnter " << n << " real numbers:  ";
    cin >> x;  /* read 1st number */
    max = x;
    /* pick the largest number in while-loop */
    while (++cnt < n) {
       cin >> x;  /* read rest numbers */
       if (max < x)
          max = x;
    }
    cout << "\nMaximum value:  " << max << endl;
}
```

do Statement

- General form of **do** statement (*repetition* statement)

```
do {
    statements
} while (expression);
```

- Semantics:
  - `statements` are executed first; thus the loop body is executed at least once.
  - Then `logical expression` is evaluated and if `true`, the loop repeats; otherwise, the loop terminates.

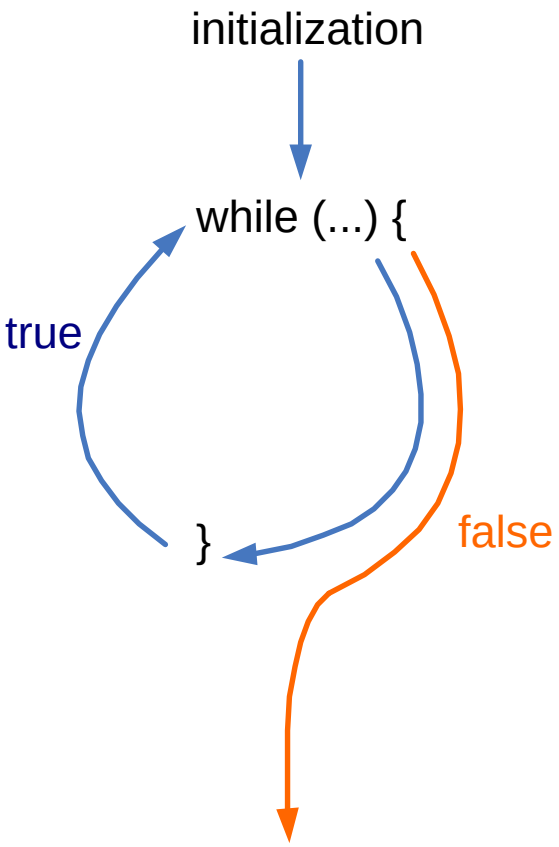
Example of do Statement

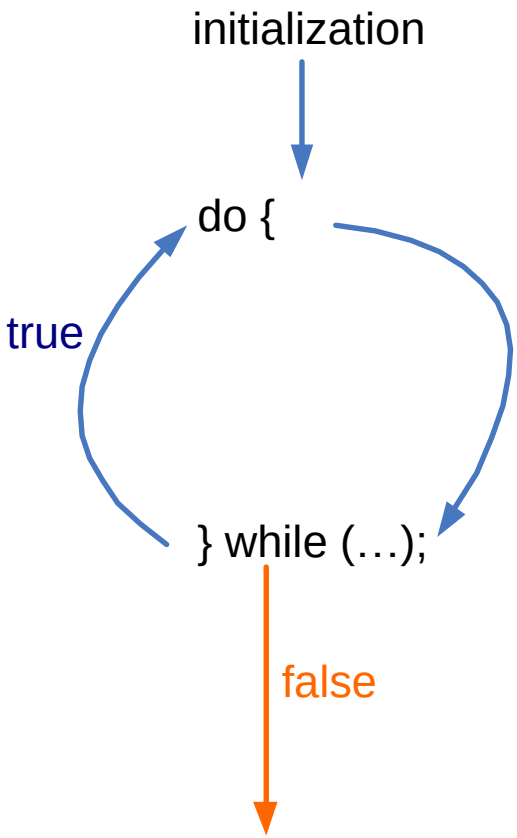
```
int error;  /* no Boolean type support */
int n;
...
...
do {
    cout << "Input a positive integer: ";
    cin >> n;
    if (error = (n <= 0))
     cout << "\nError: negative input!\n";
} while (error);
...
```

```
if (error = (n <= 0))
```

```
error = (n <= 0);
if (error)
```

`while` vs `do-while`





`for`-loop Statement

```
for (expr1 ; expr2; expr3) {
        statements;
}
```

The loop body **`statements`** are executed as long as **`expr2`** is true. When **`expr2`** becomes false, the loop ends.

`expr1`: Executed before entering the loop. Often used for variable **initialization**.

`expr3`: For each iteration, **`expr3`** is executed after executing loop body. Often used to update the counter variables.

`for`-loop Statement

`expr1` and **`expr3`** can contain multiple statements. Each statement is separated by a comma '`,`':

Example:

```
for(i=0, j=0; i<10; i++, j++) {
   ……
}
```

`expr1`: `i=0, j=0`

`expr2`: `i<10`

`expr3`: `i++, j++`

Example of `for` Statement

```
#include <iostream>
using namespace std;

int main() {
    int i;
    for (i = 0; i <10; i++)
        cout << i << endl;

    return 0;
}
```

```
#include <iostream>
using namespace std;

int main() {
    int i;
    for (i = 0; i <10; i++){
        if (i%2==0)
            cout << i << endl;
    }
    return 0;
}
```

Nested `for`-loop

```
#include<iostream>
using namespace std;
void main(){
    int i,j;
    for (i=0;i<3; i++){
      cout << "Outer for:\n";
      for (j=0;j<2; j++){
        cout << "Inner for:";
        cout << "i=" << i << ", j=" << j << endl;
      }
      cout << endl;
    }
}
```

```
Outer for:
Inner for:i=0, j=0
Inner for:i=0, j=1

Outer for:
Inner for:i=1, j=0
Inner for:i=1, j=1

Outer for:
Inner for:i=2, j=0
Inner for:i=2, j=1
```

The outer loop is executed 3 times.
For each iteration of the outer loop, the inner loop is executed 2 times.

Exercise

- Write a program to generate a multiplication of `n` rows and `m` columns (where n and m is input by the user). Assume `1<n` and `m<=9`.
- E.g. when `n=4`, `m=3`, the following table is generated:

```
1       2       3
2       4       6
3       6       9
4       8       12
```

Answer

```
#include <iostream>
using namespace std;

int main()
{
   int m, n;
   cout << "enter m, n:" << endl;
   cin >> m >> n;

   for (int i = 1; i <= n; i++)
   {
       for (int j = 1; j <= m; j++)
       {
          cout << i*j<<" ";
       }

       cout << endl;
    }
   return 0;
}
```

`break` Statement

- the **`break`** statement causes an **exit** from the innermost enclosing loop or `switch` statement (last week).

```
int n = 0;
while (1) {
  cin >> n;
  if (n < 0)
     break;  /* exit loop if x is negative */
  cout << sqrt(n) << endl;
}
/* If break is executed, jumps to here */
```

`continue` Statement

- `continue` statement causes the current iteration of a loop to **stop** and the next iteration to begin immediately.

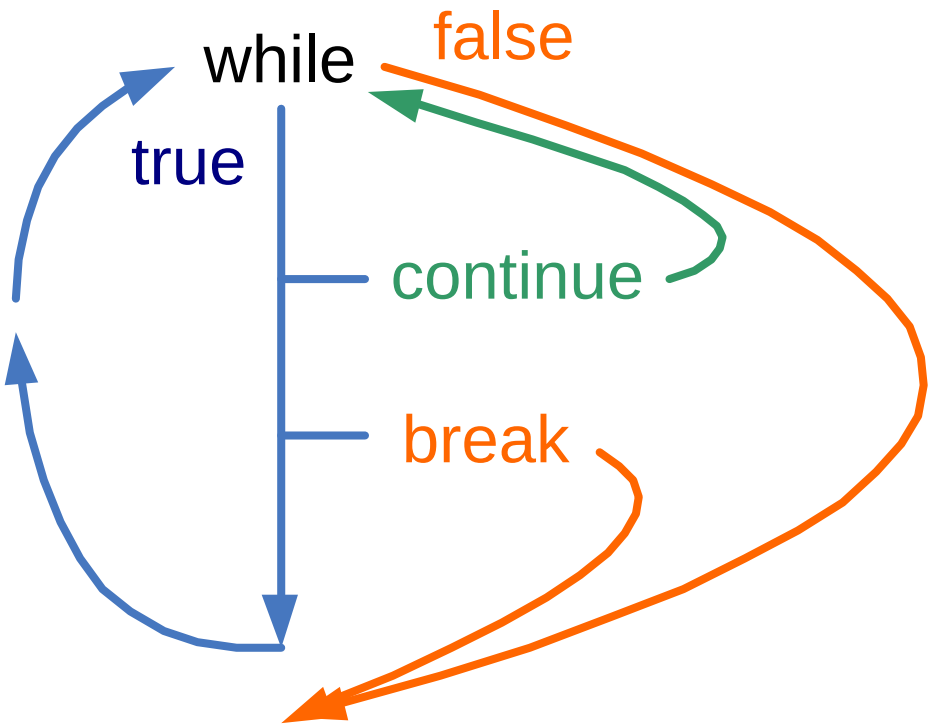
- It can be applied in a `while`, `do-while` or `for` statement.

Example of `continue` Statement

```
/* read in and sum 10 not-too-small nos. */
   int cnt = 0;
   float x = 0, sum = 0;

   while (cnt < 10) {
     cin >> x;
     if (x > -0.01 && x < 0.01)
        continue; /* discard small values */
     ++cnt;
     sum += x;
     /* continue transfer control here */
   }
```

`continue` and `break`



Comma Operator (,)

- It has the **lowest** precedence of all operators in C++ and is evaluated **from left to right**.
- General form is

```
expr1, expr2
```

- Sometimes used in `for` statements to allow **multiple initializations** and **multiple processing** of **counter variables**.
- The comma operator is **rarely** used.

Examples of Comma Operator

```
sum=0;
for (j=1; j<=10; j++)
  sum += j;
```

and

```
for (sum=0,j=1; j<=10; j++)
  sum += j;
```

and

```
for (sum=0,j=1; j<=10; sum += j, j++)
  ;
```

are equivalent

Common Programming Errors

- Mix up of assignment **=** and test of equality **==** .
- Mix up of the comma operator **,** and the semi-colon **;** .
- Unaware of extra semi-colons, e.g.,

```
sum=0;
for (j=1; j<=10; j++)
   sum += j;
```

is not the same as

```
sum=0;
for (j=1; j<=10; j++);
  sum += j;
```

Common Programming Errors

- Fail to ensure that the termination condition of a loop is **reachable**.
- Misuse of relational operators.

```
int k=8;
if (2 < k < 7)
   cout >> "true"; /* print true */
else
   cout >> "false";
```

- Use `(2 < k && k < 7)` for the correct answer!

Further Remarks

- Don't use a variable of any **floating** point data type to control a loop because real numbers are represented in their **approximate values** internally.

- **Infinite** loop can be made:

```
while (1)
{
   ……
}
```

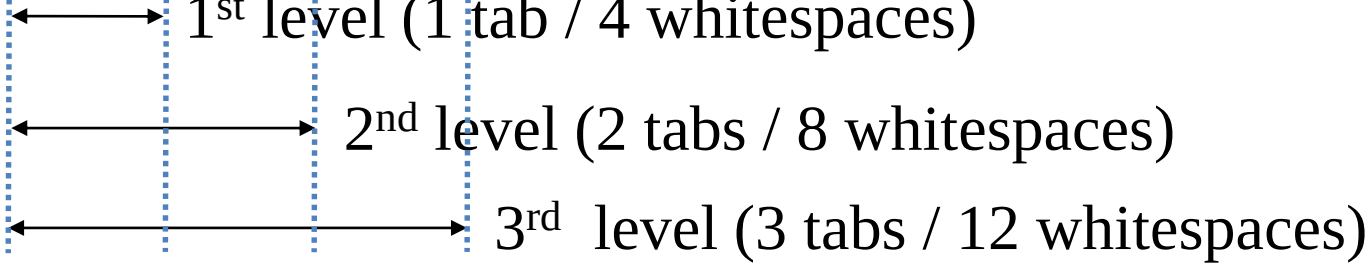
```
for ( ; ; )
{
    ……
}
```

Programming Style

- Follow the normal rules of **English** (e.g. putting a space after a comma).
- Put one **space** on each side of a binary operator (e.g. `=` `>` `<`) to enhance readability.
- Indent code in a **consistent** fashion to indicate the **flow of control** (use the `tab/whitespace` key).
  - Note the multiple levels of indentation.

Indentation

```
void main(){
    int i;
    for (i = 0; i < 100; i++) {
        if (i>3)
            cout << i;
    }
}
```



Formatting Programs

- It's highly recommended to code properly as you write program to reflect the **structure of the program**.
  - Improve **readability** and increase the **ease for debugging** the program.
  - In assignment, **marks** will be allocated for **indentation**.

- To indent in visual studio, you may press the `tab` key.

- You may select multiple lines of statement and press `tab` to indent the selected statements.

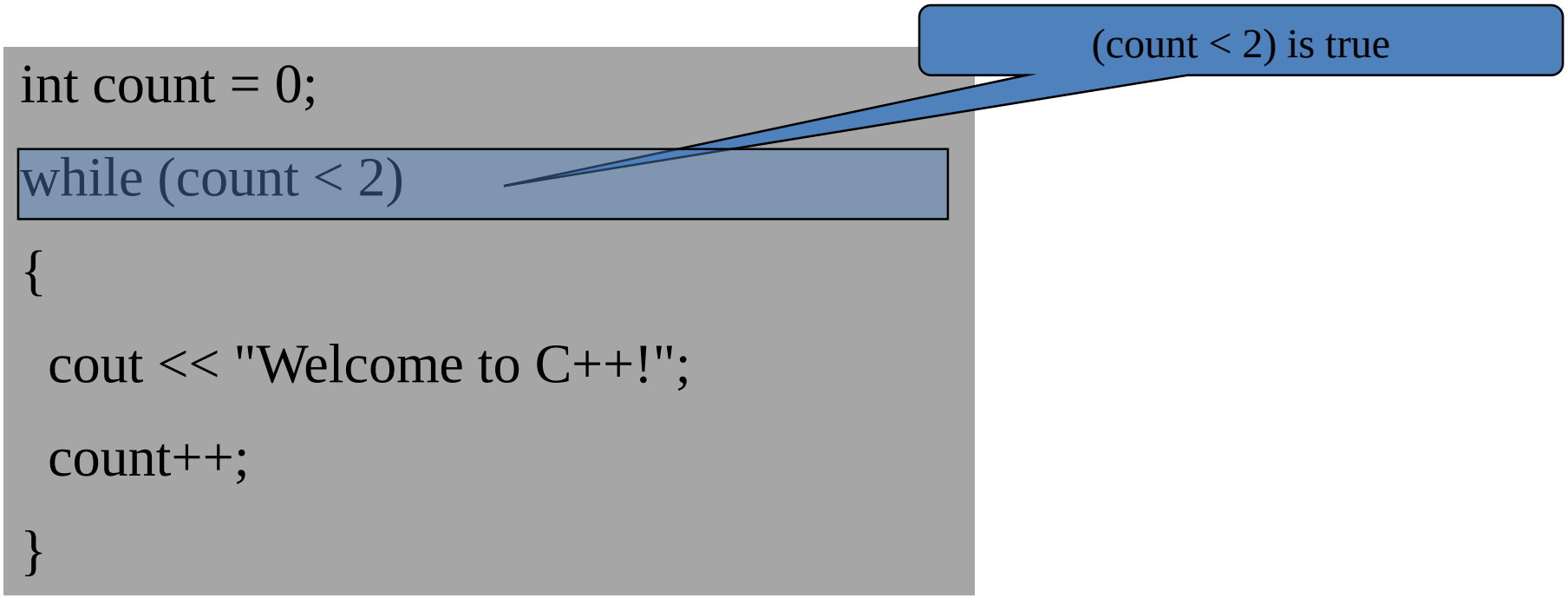
- To move back one level to the left, press `shift+tab`.

Trace while Loop

```
int count = 0;
while (count < 2)
{
  cout << "Welcome to C++!";
  count++;
}
```

Initialize count

Trace while Loop, cont.

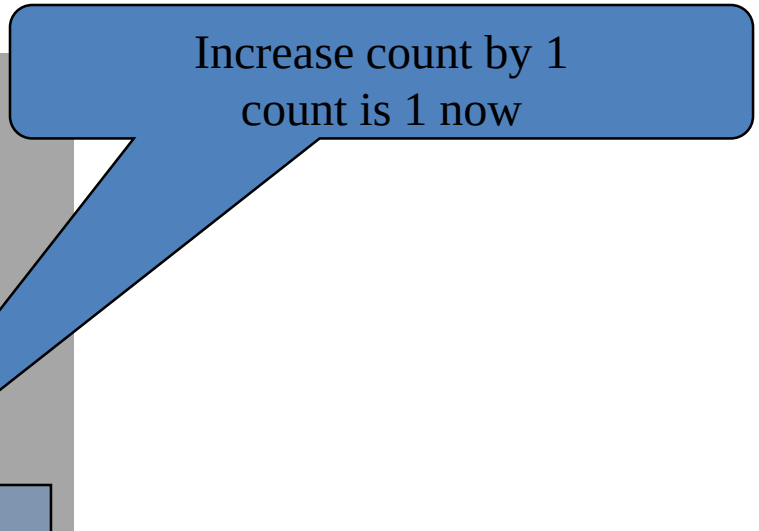


Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
  cout << "Welcome to C++!";
  count++;
}
```

Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
  cout << "Welcome to C++!";
  count++;
}
```



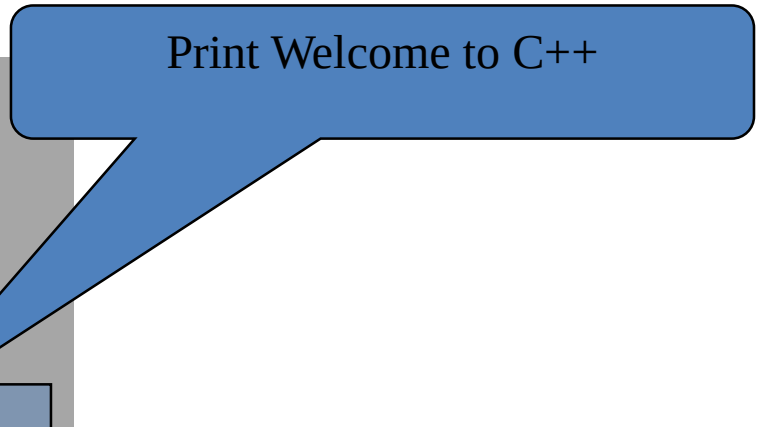
Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
  cout << "Welcome to C++!";
  count++;
}
```

(count < 2) is still true since count is 1

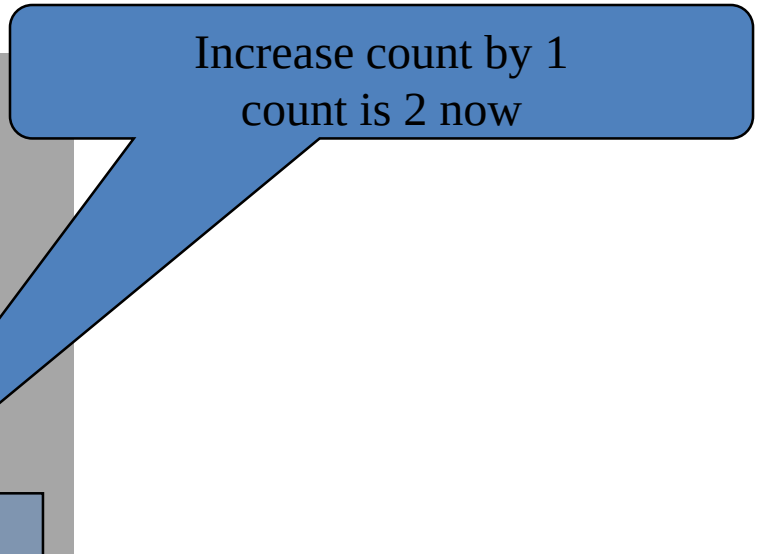
Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
cout << "Welcome to C++!";
  count++;
}
```



Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
  cout << "Welcome to C++!";
  count++;
}
```



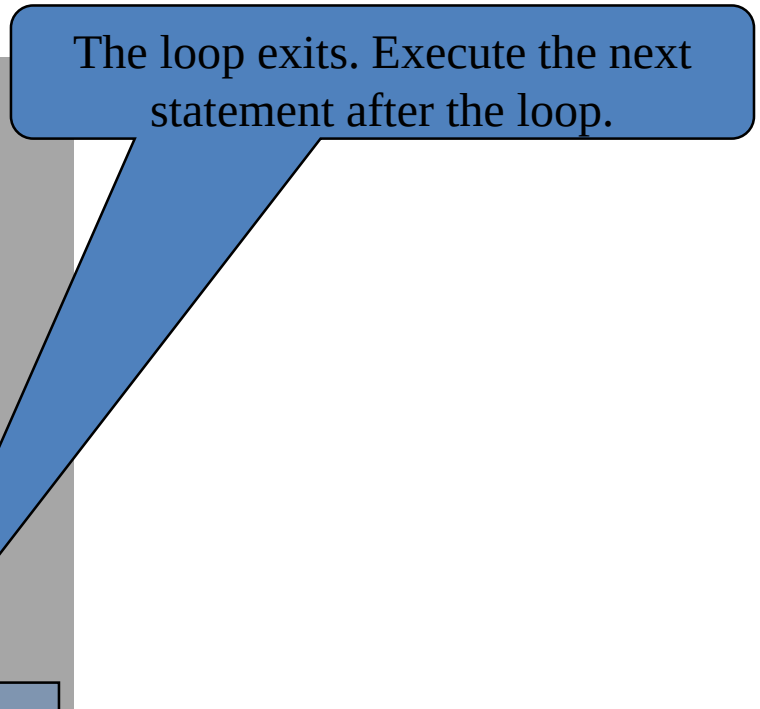
Trace while Loop, cont.

```
int count = 0;
while (count < 2)
{
 cout << "Welcome to C++!";
  count++;
}
```

(count < 2) is false since count is 2 now

Trace while Loop

```
int count = 0;
while (count < 2)
{
  cout << "Welcome to C++!";
  count++;
}
```



Trace for Loop

```
int i;
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

Declare i

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

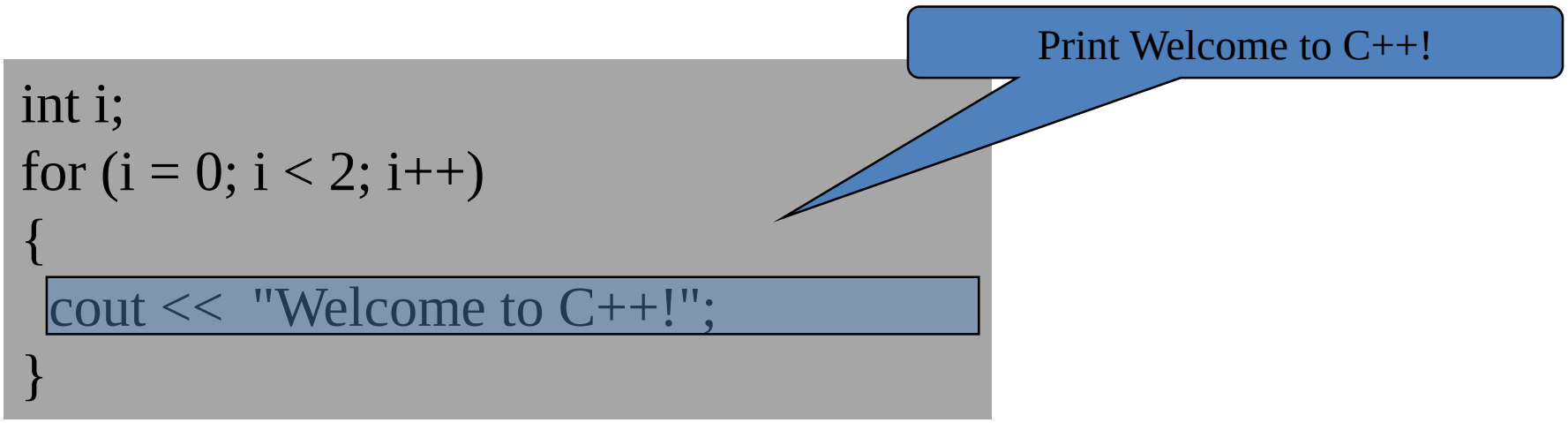
Execute initializer
i is now 0

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

(i < 2) is true
since i is 0

Trace for Loop, cont.



Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

Execute adjustment statement
i now is 1

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

(i < 2) is still true
since i is 1

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

Execute adjustment statement
i now is 2

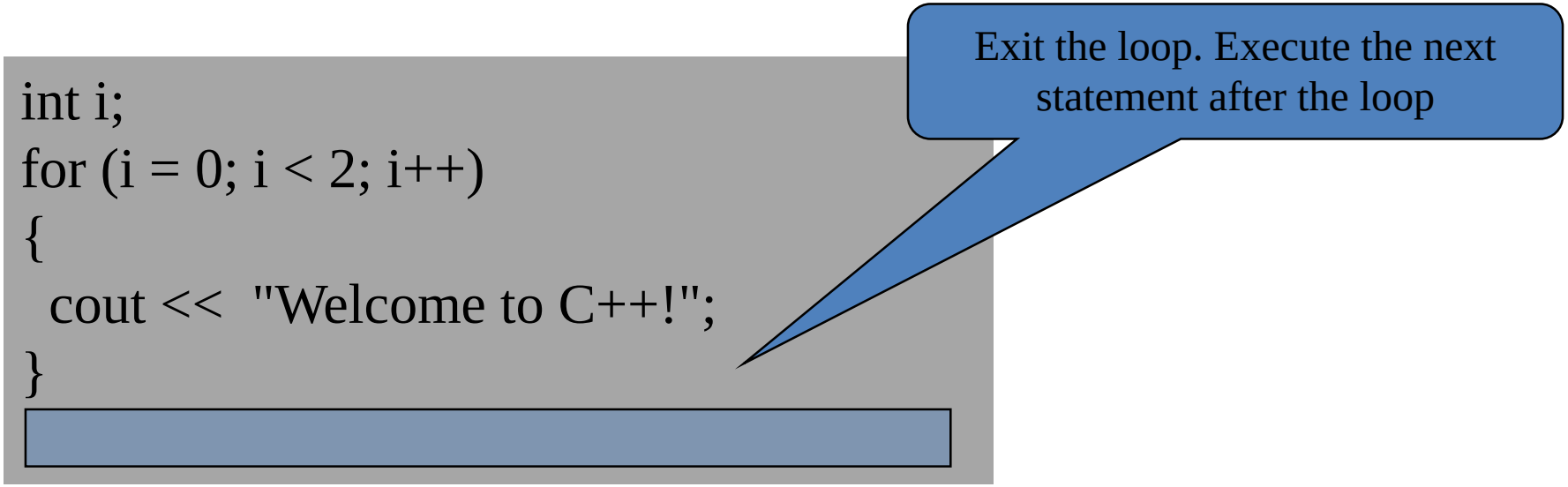
Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```

(i < 2) is false
since i is 2

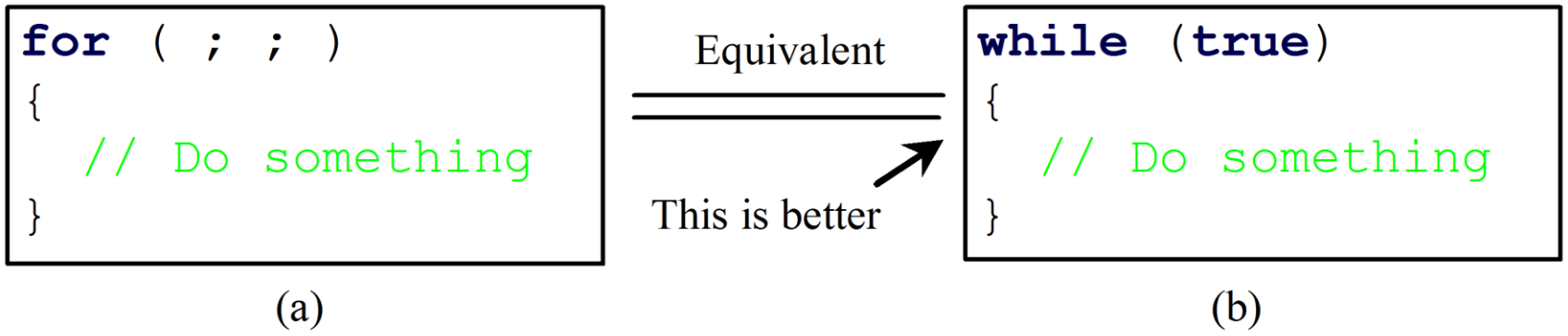
Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
  cout <<  "Welcome to C++!";
}
```



Note

If the loop-continuation-condition in a for loop is omitted, it is implicitly true. Thus the statement given below in (a), which is an infinite loop, is correct. Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:



Summary

- In C++, repeating task could be expressed with loop statements:
  - `while(…){…}`
  - `do {…} while(…);`
  - `for (…;…;…){…}`

- A complete looping structure may consist of :
  - Loop initialization statements.
  - Loop Body.
  - Exit condition.
  - Post Loop statements.