Computer Science I

CS 135

6. Repetition: While and For Loops

René Doursat

Department of Computer Science & Engineering
University of Nevada, Reno

Spring 2006

Computer Science I

CS 135

0. Course Presentation
1. Introduction to Programming
2. Functions I: Passing by Value
3. File Input/Output
4. Predefined Functions
5. If and Switch Controls
6. **While and For Loops**
7. **Functions II: Passing by Reference**
8. **1-D and 2-D Arrays**

Computer Science I

CS 135

6. Repetition: While and For Loops

a. **Repetition Structures**

b. **While Loops**

c. **Do/While Loops**

d. **For Loops**

Computer Science I

CS 135

6. Repetition: While and For Loops

a. Repetition Structures

- ✓ Reminder: three types of control structures
- ✓ Why is repetition needed?
- ✓ While, do/while and for loops

b. While Loops

c. Do/While Loops

d. For Loops

6.a Repetition Structures

Reminder: three types of control structures

- Sequence, selection and repetition structures

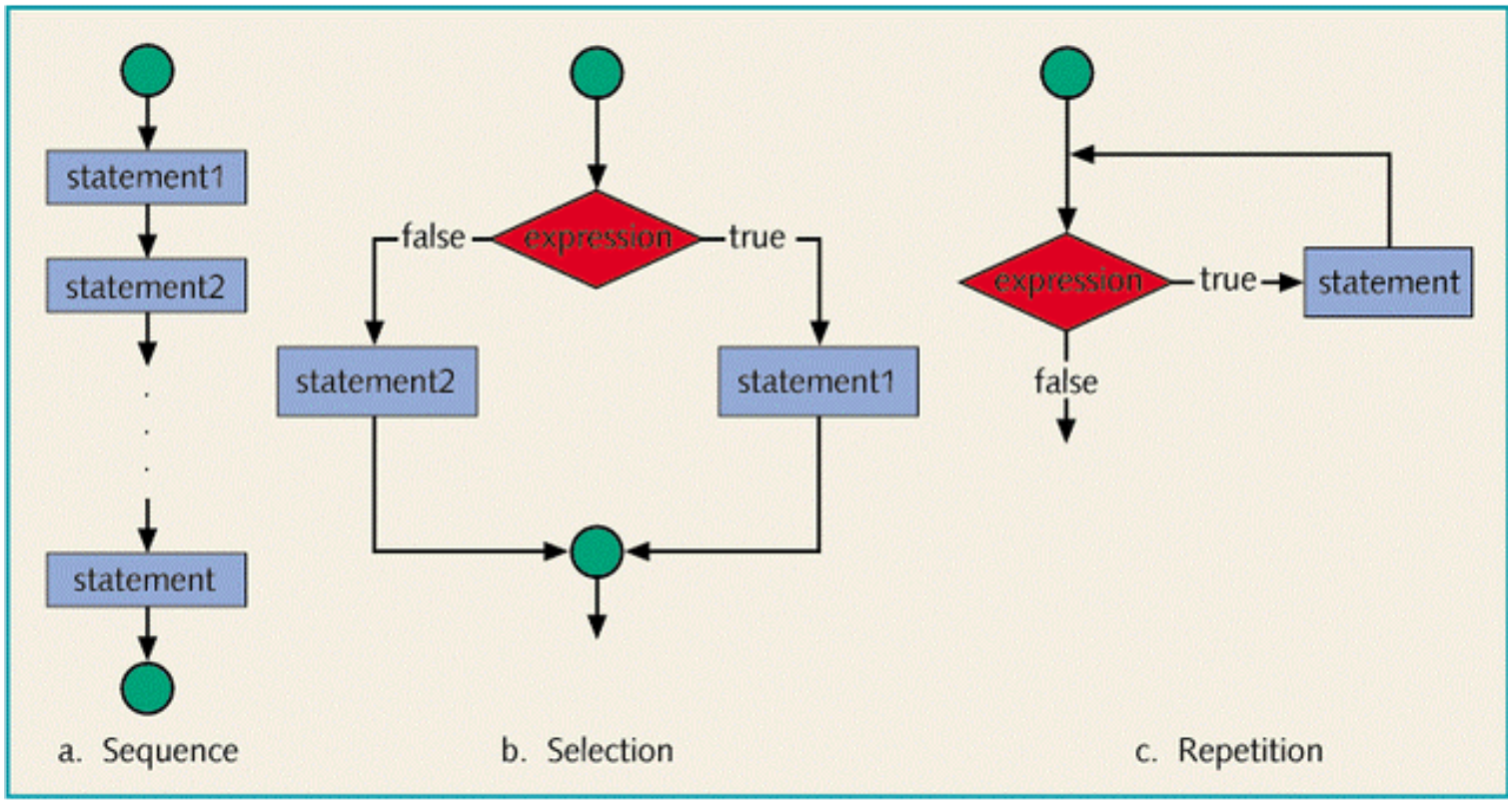


Figure 4-1 Flow of execution

6.a Repetition Structures

Reminder: three types of control structures

➢ Structure theorem

- ✓ it is possible to write any computer program by using only three basic control structures that are easily represented in pseudocode:
  - sequence structures
  - selection structures
  - repetition structures

introduce branching ("jumps") in the sequential logic

➢ Sequence structures

- ✓ straightforward execution of one processing step after another
- ✓ sequence of pseudocode statements: do this, do that, then this, then that, etc.

6.a Repetition Structures

Reminder: three types of control structures

- Selection structures
  - ✓ condition and choice between two actions, depending on whether the condition is true or false
  - ✓ represented by the pseudocode keywords IF, THEN, ELSE, and ENDIF
- Repetition structures
  - ✓ block of statements to be executed repeatedly, as long as a condition is true
  - ✓ represented by the pseudocode keywords WHILE and ENDWHILE (or DOWHILE and ENDDO)

6.a Repetition Structures

Why is repetition needed?

- A computer can repeat a group of actions
  → *repetition structures*
  - ✓ examples:
    - calculate 100 student grades
    - pour water in the saucepan until it is full
    - cook the pasta until it is “al dente”
  - ✓ pseudocode example:

```
WHILE water_level < pan_height
    Add 1 tablespoon to water_volume
    water_level = water_volume / pan_surface
ENDWHILE
```

6.a Repetition Structures

Why is repetition needed?

- Repetition allows to efficiently use variables
  - ✓ for example, repetition allows to input, add, and average multiple numbers using a limited number of variables
  - ✓ adding four numbers without a loop (the old-fashioned way):
    - declare a variable for each number, input all the numbers and add all the variables together
  - ✓ adding four numbers with a loop (the high-tech way):
    - → create a loop that iteratively reads a number and adds it to a variable holding the sum of the numbers

6.a Repetition Structures

Why is repetition needed?

```
void main()
{
   // declare variables
   int num1, num2, num3, num4, sum;

   // prompt user for 4 numbers
   cout << "Enter number: ";
   cin >> num1;
   cout << "Enter number: ";
   cin >> num2;
   cout << "Enter number: ";
   cin >> num3;
   cout << "Enter number: ";
   cin >> num4;

   // calculate sum
   sum = num1 + num2 + num3 + num4;

   // display result
   cout << "The sum is: ";
   cout << sum << endl;
}
```

```
void main()
{
   // declare variables
   int num, sum = 0;

   // prompt and increment sum 4 times
   cout << "Enter number: ";
   cin >> num; sum += num;

   cout << "Enter number: ";
   cin >> num; sum += num;

   cout << "Enter number: ";
   cin >> num; sum += num;

   cout << "Enter number: ";
   cin >> num; sum += num;

   // display result
   cout << "The sum is: ";
   cout << sum << endl;
}
```

Adding four numbers by repeating four times the same task

6.a Repetition Structures

Why is repetition needed?

```
void main()
{
   // declare variables
   int num, sum = 0;

   // prompt and increment sum 4 times
   cout << "Enter number: ";
   cin >> num; sum += num;

   cout << "Enter number: ";
   cin >> num; sum += num;

   cout << "Enter number: ";
   cin >> num; sum += num;

   cout << "Enter number: ";
   cin >> num; sum += num;

   // display result
   cout << "The sum is: ";
   cout << sum << endl;
}
```

```
void main()
{
   // declare variables
   int num, sum = 0;
   int counter = 0;

   // prompt and increment sum 4 times
   while (counter < 4) {
       cout << "Enter number: ";
       cin >> num; sum += num;

       counter++;
   }

   // display result
   cout << "The sum is: ";
   cout << sum << endl;
}
```

Repeating four times the same task with a loop

6.a Repetition Structures

Why is repetition needed?

```
void main()
{
   // declare variables
   int num, sum = 0;
   int counter = 0;

   // prompt and increment sum 4 times
   while (counter < 4) {
        cout << "Enter number: ";
        cin >> num; sum += num;

        counter++;
   }

   // display result
   cout << "The sum is: ";
   cout << sum << endl;
}
```

```
void main()
{
   // declare variables
   int num, sum = 0;
   int counter = 0;
   int num_count;

   // prompt user for number of inputs
   cin >> num_count;

   // increment sum num_count times
   while (counter < num_count) {
        cout << "Enter number: ";
        cin >> num; sum += num;

        counter++;
   }

   // display result
   cout << "The sum is: ";
   cout << sum << endl;
}
```

Repeating the same task with a loop a variable number of times

6.a Repetition Structures

Why is repetition needed?

- Benefits of repetition
  - ✓ you can vary the number of inputs: instead of just four, it can be any user-specified variable
  - ✓ this number can be very large: you can repeat 10 times, 100 times or 10,000 times the same task without changing the code
    - example: calculating students' grades for the whole class using the same formula
  - ✓ you can also repeat an action while a specific condition is valid (or until its contrary has been met)
    - example: reading lines from a file until the end of the file
    - example: asking the user for their choice, until it is a valid selection

6.a Repetition Structures

While, do/while and for loops

- Three types of loops
  - ✓ **`while`** loops
    - (a) evaluate the logical expression first; (b) if true, execute the body, otherwise exit; (c) go to (a)
  - ✓ **`do … while`** loops
    - (a) execute the body first; (b) then, evaluate the logical expression; (c) if true, go to (a)
  - ✓ **`for`** loops
    - **`for`** loops are a specialized form of **`while`** loops that simplifies the writing of counter-controlled loops

Computer Science I

CS 135

6. Repetition: While and For Loops

a. Repetition Structures

- ✓ Reminder: three types of control structures
- ✓ Why is repetition needed?
- ✓ While, do/while and for loops

b. While Loops

c. Do/While Loops

d. For Loops

Computer Science I

CS 135

6. Repetition: While and For Loops

a. Repetition Structures

b. While Loops

- ✓ General form of a while loop
- ✓ Case 1: counter-controlled while loops
- ✓ Case 2: sentinel-controlled while loops
- ✓ Case 3: flag-controlled while loops

c. Do/While Loops

d. For Loops

6.b While Loops

General form of a while loop

- General flowchart of a while loop
  - ✓ (a) evaluate the logical expression first; (b) if true, execute the body, otherwise exit; (c) go to (a)

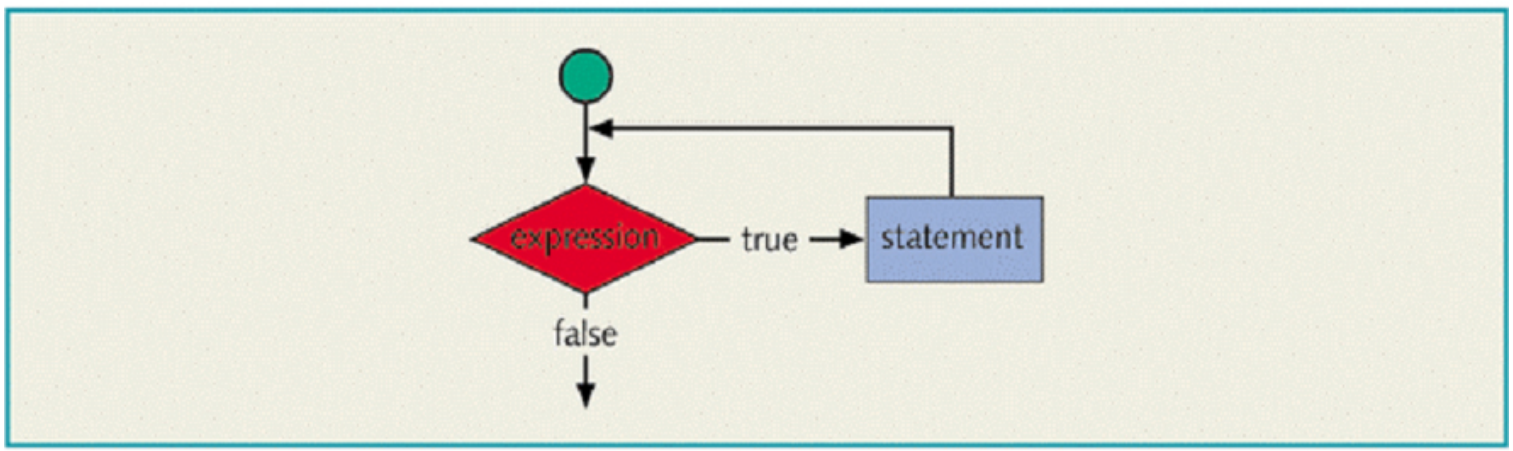


Figure 5-1 `while` loop

6.b While Loops

General form of a while loop

- Syntax of a while loop

```
while (expression)
    statement
```

```
while (!end_of_file)
    read_line();
```

- ✓ **`while`** is a reserved keyword
- ✓ *expression* is a logical expression (or variable, or function)
  - *expression* provides an entry condition into the loop
- ✓ *statement* follows *expression* and can be any C++ statement
  - *statement* executes if *expression* initially evaluates to true
  - then *expression* is reevaluated and *statement* executes again multiple times as long as the *expression* is still true
  - when *expression* becomes false, *statement* is bypassed and the program goes to the next statement directly

6.b While Loops

General form of a while loop

- While loop with compound statement
  - ✓ the body of a while loop can contain multiple C statements
  - ✓ a block of statements is called a "compound statement" and must be surrounded with curly braces { }

```
while (expression) {
    statement1
    statement2
    statement3
}
```

```
while (!end_of_file) {
    read_line();
    display_line();
    ...
}
```

6.b While Loops

Case 1: counter-controlled while loops

- A counter-controlled loop repeats statements a fixed number of times
  - ✓ if you know exactly how many pieces of data need to be processed, the while loop can be a counter-controlled loop

```
int n = 17;                 // set number of times

int counter = 0;            // init loop variable
while (counter < n) {       // test loop variable
      ...
      counter++;            // update loop variable
      ...
}
```

6.b While Loops

Case 2: sentinel-controlled while loops

- A sentinel-controlled loop repeats statements until some value is reached
  - ✓ you do not know how many pieces of data need to be processed, but you will know the last one when you see it

```
cin >> var;                  // init loop variable
while (var != sentinel) { // test loop variable
      ...
      cin >> var;            // update loop variable
      ...
}
```

6.b While Loops

Case 3: flag-controlled while loops

- A flag-controlled loop repeats statements until a boolean flag becomes false
  - ✓ here too, you do not know how many pieces of data need to be processed; this time, you carry over the value of a complex expression into a flag that will stop the loop

```
found = false;              // init loop variable
while (!found) {            // test loop variable
      ...
      if (expression)
          found = true; // update loop variable
      ...
}
```

Computer Science I

CS 135

6. Repetition: While and For Loops

a. Repetition Structures

b. While Loops

- ✓ General form of a while loop
- ✓ Case 1: counter-controlled while loops
- ✓ Case 2: sentinel-controlled while loops
- ✓ Case 3: flag-controlled while loops

c. Do/While Loops

d. For Loops

Computer Science I

CS 135

6. Repetition: While and For Loops

a. Repetition Structures

b. While Loops

c. **Do/While Loops**

- ✓ General form of a do/while loop

d. **For Loops**

6.c Do/While Loops

General form of a while loop

- General flowchart of a do/while loop
  - ✓ (a) execute the body first; (b) then, evaluate the logical expression; (c) if true, go to (a)

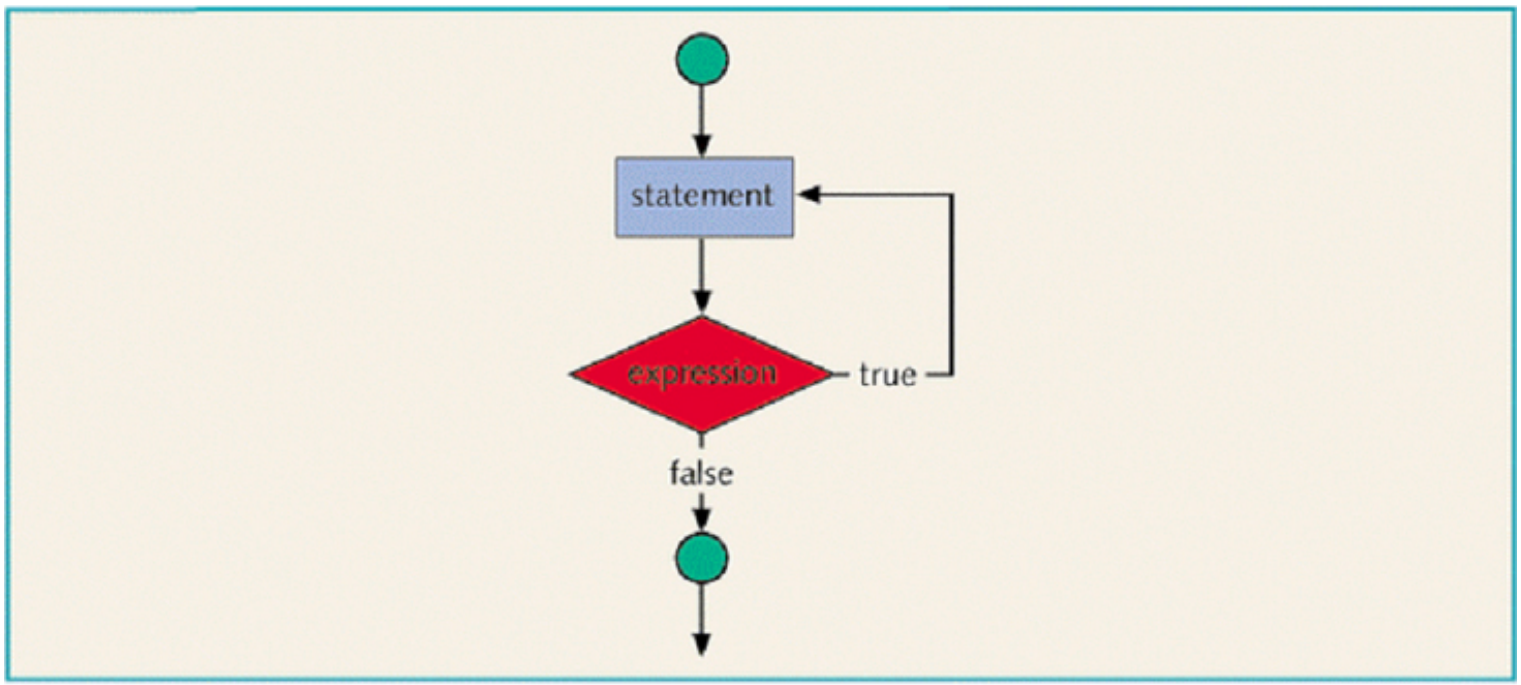


Figure 5-3 `do...while` loop

6.c Do/While Loops

General form of a while loop

- General syntax of a do/while loop

```
do {
    statement1
    statement2
    statement3
} while (express.);
```

```
do {
    cin >> ans;
    switch (ans) {
    ... }
} while (ans != 'q');
```

- ✓ **do** is a reserved keyword
- ✓ don't forget the semicolon **;** after the while expression!
- ✓ the statements execute first, then the expression is evaluated
- ✓ if expression evaluates to true, statements execute again, etc.
- ✓ a do/while loop always iterates at least once

Computer Science I

CS 135

6. Repetition: While and For Loops

a. Repetition Structures

b. While Loops

c. **Do/While Loops**

- ✓ General form of a do/while loop

d. **For Loops**

Computer Science I

CS 135

6. Repetition: While and For Loops

a. Repetition Structures

b. While Loops

c. Do/While Loops

d. For Loops

- ✓ General form of a for loop
- ✓ Examples of for loops
- ✓ Nested for loops

6.d For Loops

General form of a for loop

- General flowchart of a for loop
  - ✓ compared to a while loop, a for loop additionally contains an **initial statement** and an **update statement**

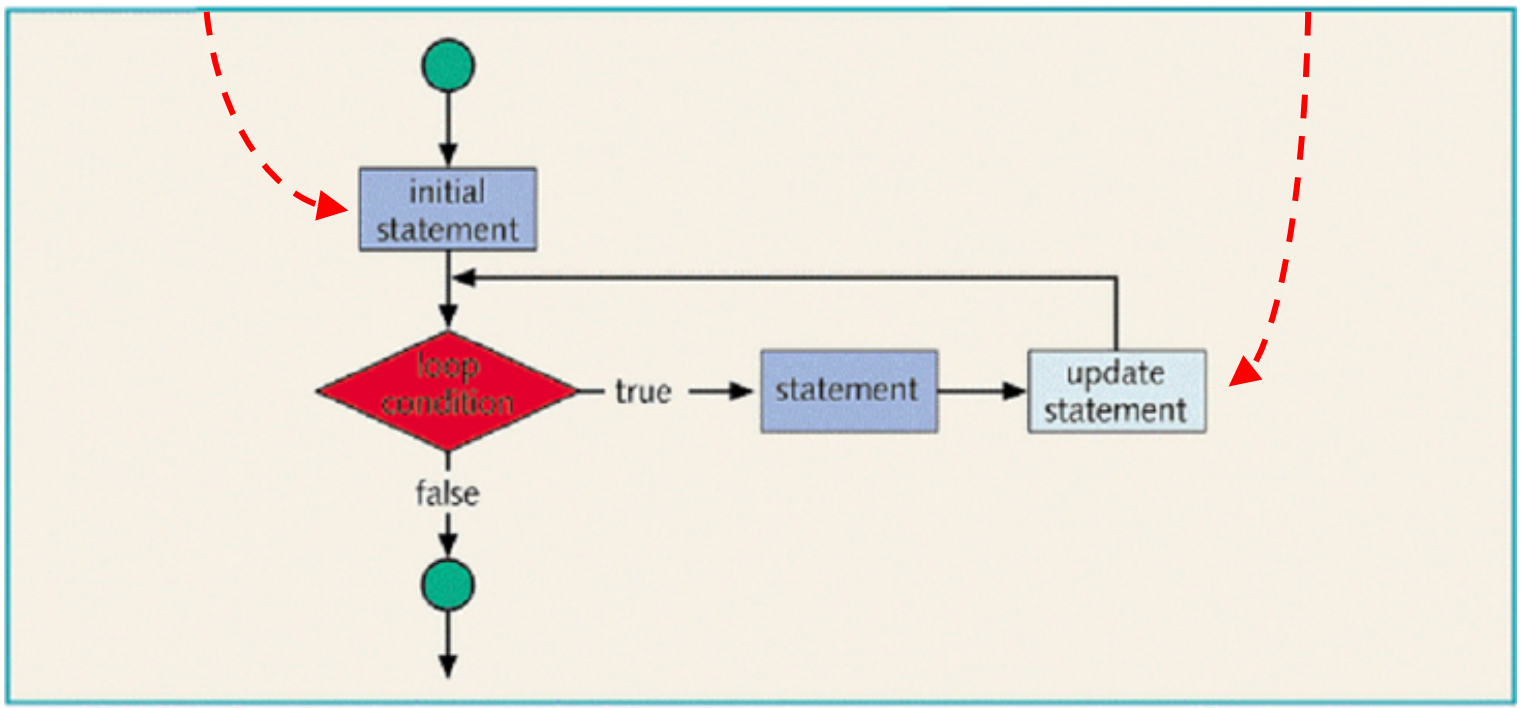
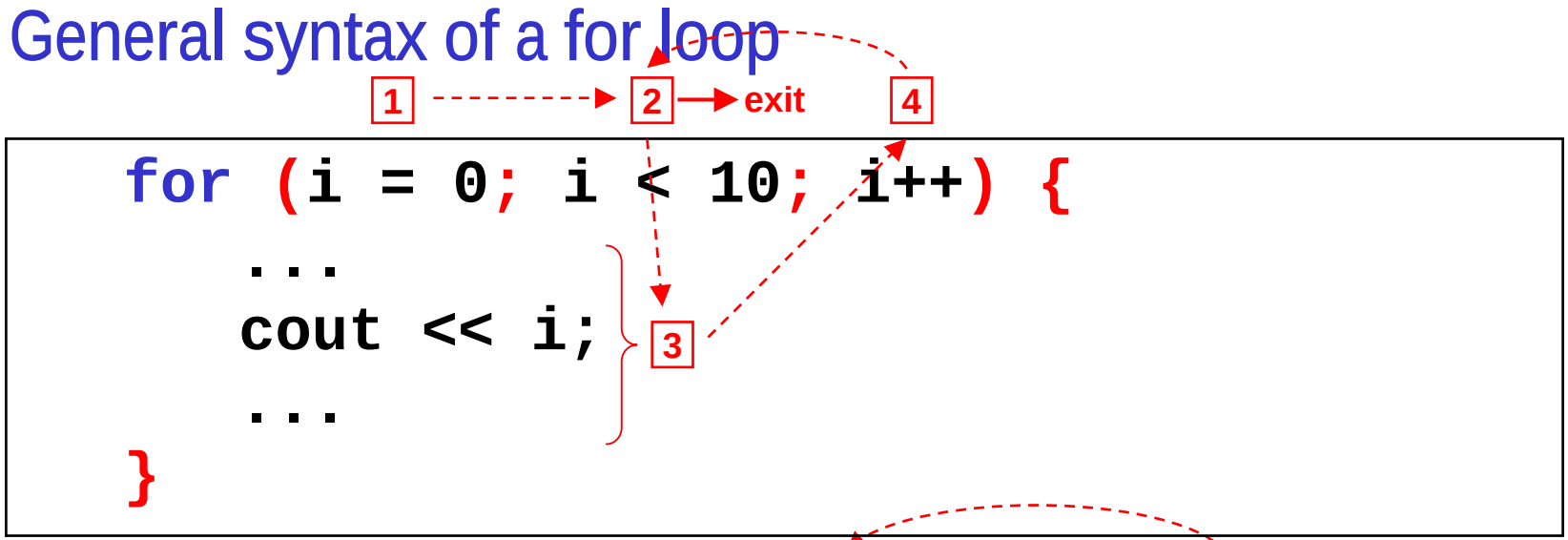


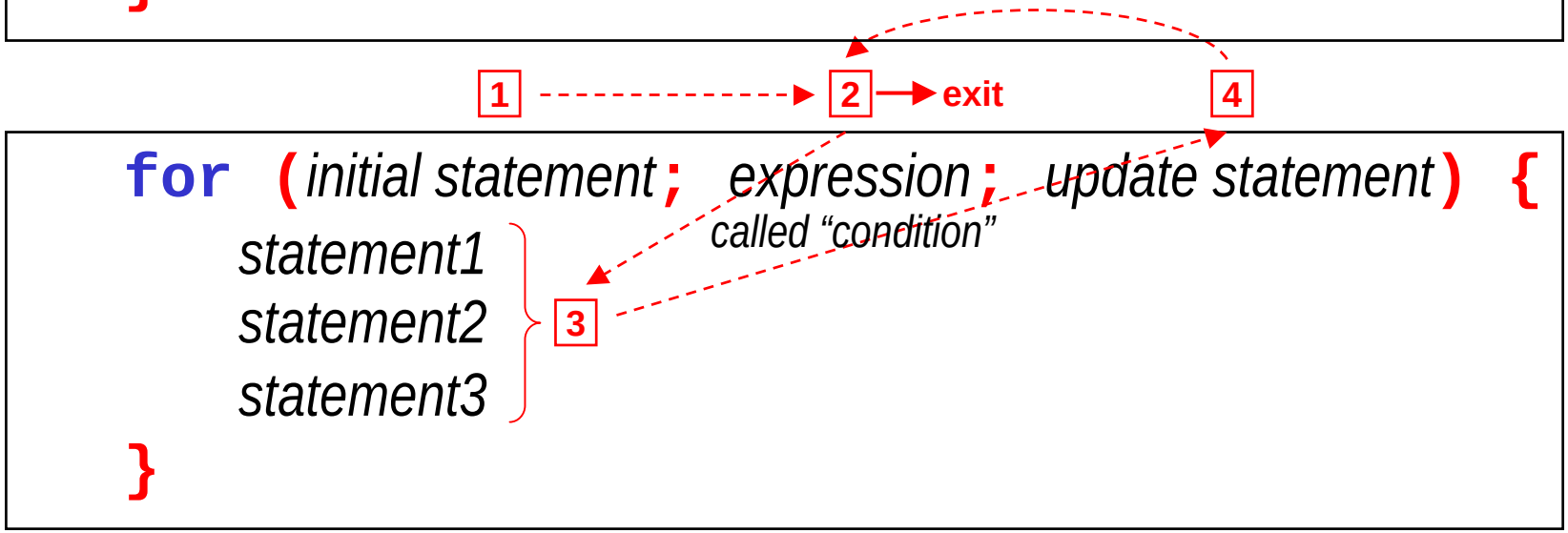
Figure 5-2 `for` loop

6.d For Loops

General form of a for loop

- General syntax of a for loop





6.d For Loops

General form of a for loop

```
void main()
{
  // declare variables
  int num, sum = 0;
  int counter = 0;

  // prompt and increment sum 4 times
  while (counter < 4) {
      cout << "Enter number: ";
      cin >> num; sum += num;

      counter++;
  }

  // display result
  cout << "The sum is: ";
  cout << sum << endl;
}
```

```
void main()
{
  // declare variables
  int num, sum = 0;
  int counter = 0;

  // prompt and increment sum 4 times
  for (counter = 0; counter < 4;
                              counter++) {
      cout << "Enter number: ";
      cin >> num; sum += num;

      counter++;
  }

  // display result
  cout << "The sum is: ";
  cout << sum << endl;
}
```

Implementing a counter-controlled loop with a for loop

6.d For Loops

General form of a for loop

- A for loop is really just a more compact way to write a while loop; it is *not* a new kind of control structure
  - ✓ it packages an initialization statement, a logical expression and an update statement on one line, the for header line
- A for loop executes as follows
  - ✓ the *initial statement* executes
  - ✓ the loop *condition* is evaluated
  - ✓ if the loop *condition* evaluates to true:
    - execute the loop body *statements*
    - execute the *update statement*
  - ✓ repeat previous step until the loop *condition* evaluates to false

6.d For Loops

General form of a for loop

- (Fun?) facts about for loops
  - ✓ the *initial statement* generally initializes some variable
  - ✓ the *initial statement* in the for loop is the first to be executed and is executed only once
  - ✓ if the loop *condition* is initially false, the body *statements* never execute and the loop exits
  - ✓ the *update statement* changes the value of the loop control variable which eventually sets the value of the *condition* to false
  - ✓ the loop executes indefinitely if the loop *condition* stays true

6.d For Loops

General form of a for loop

- (Fun?) facts about for loops (cont'd)
  - ✓ a semicolon at the end of the for line creates an empty loop: this is sometimes used as a (bad) way to slow down execution

```
for (i = 0; i < 10000; i++);
```

  - ✓ the *initial statement*, loop *condition*, and *update statement* may all be omitted, independently or together, for example:

```
for ( ; i < 10; i++) { ... }
for (i = 0; ; i++) { ... }
for (i = 0; i < 10; ) { ... }
for ( ; ; ) { ... }
```

  - ✓ in this case, the flow of execution mostly depends on the body

→ *this is legal syntax but not necessarily good programming!*

6.d For Loops

Example of for loops

- For loops are mainly used as counter-controlled loops
  - ✓ traditional counter-controlled loop

```
for (i = 0; i < max; i++) { ... }
```

  - ✓ counter-controlled loop with different ranges

```
for (i = 1; i <= max; i++) { ... }
for (i = 7; i < max+7; i++) { ... }
```

  - ✓ decreasing counter-controlled loops

```
for (i = max; i > 0; i--) { ... }
for (i = max-1; i >= 0; i--) { ...}
```

6.d For Loops

Example of for loops

- Other types of for loops are possible, but rarely used
  - ✓ flag-controlled for loop

```
for (found = false; !found; found = ...) {
        ...
}
```

is the equivalent of

```
found = false;
while (!found) {
        ...
        found = ...
}
```

6.d For Loops

Nested for loops

- What code can print out a triangle of stars?

- ✓ answer:

```
for (i = 1; i <= 5 ; i++) {
    for (j = 0; j < i; j++)
        cout << "* ";
    cout << endl;
}
```

6.d For Loops

Nested for loops

- What does this code print out?

```
for (i = 5; i >= 1 ; i--) {
    for (j = 0; j < i; j++)
        cout << "* ";
    cout << endl;
}
```

  - ✓ answer:

```
* * * * *
* * * *
* * *
* *
*
```

6.d For Loops

Nested for loops

- What code can print out the identity matrix?

```
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

- ✓
```
for (i = 0; i < 5 ; i++) {
    for (j = 0; j < i; j++) {
        cout << (i == j ? 1 : 0);
        cout << " ";
    }
    cout << endl;
}
```

Computer Science I

CS 135

6. Repetition: While and For Loops

a. Repetition Structures

b. While Loops

c. Do/While Loops

d. For Loops

- ✓ General form of a for loop
- ✓ Examples of for loops
- ✓ Nested for loops

Computer Science I

CS 135

6. Repetition: While and For Loops

a. Repetition Structures

b. While Loops

c. Do/While Loops

d. For Loops

Computer Science I

CS 135

0. Course Presentation
1. Introduction to Programming
2. Functions I: Passing by Value
3. File Input/Output
4. Predefined Functions
5. If and Switch Controls
6. While and For Loops
7. **Functions II: Passing by Reference**
8. **1-D and 2-D Arrays**