C++ Programming

Ch. 5 Loops and Relational Expressions

Spring 2014

Myung-Il Roh

Department of Naval Architecture and Ocean Engineering
Seoul National University

Ch. 5 Loops and Relational Expressions

Contents

The ‘for’ Loop (1/2)

, as long as the loop test condition evaluates to true or nonzero, and the loop terminates execution when the test condition evaluates to false or zero.

☑ A ‘for’ loop provides a

.

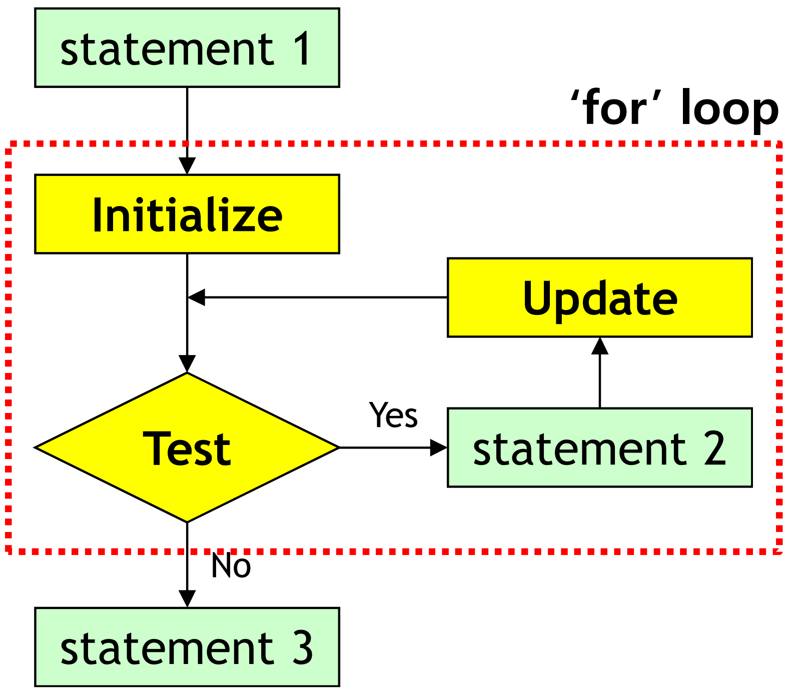
☑ Steps for the ‘for’ Loop

- Setting a value initially
- Performing a test to see whether the loop should continue
- Executing the loop actions
- Updating value(s) used for the test

The 'for' Loop (2/2)

- ☑ Expression
  statement 1;

  statement 3;
- ☑ Using
  for the Test-Expression
  - Ex. *for (int i = 0; i < 5; i++)*
- ☑ Updating Value(s)
  for the Update-Expression
  - Ex. *i++* or *i = i + by*
    (by: user-defined variable)



Operators (1/9)

- + (addition), - (subtraction), * (multiplication), / (division), % (taking the modulus)

- The value of an expression becomes the value of the member on the left.
- Ex.

```
x = 20;
maids = (cooks = 4) + 3;  // cooks = 4, maids = 7.
x = y = z = 0;      // Assignment associates                . Fast way to
                       set several variables to the same value.
```

Operators (2/9)

- Combined addition and assignment operator that accomplishes the same result more concisely
- Ex.

```
L += R;   // Same as 'L = L + R'. Assign L + R to L.
L -= R;   // Same as 'L = L – R'. Assign L - R to L.
L *= R;   // Same as 'L = L * R'. Assign L * R to L.
L /= R;   // Same as 'L = L / R'. Assign L / R to L.
L %= R;   // Same as 'L = L % R'. Assign L % R to L.
```

Operators (3/9)

- Increment operator (++): Prefix increment operator (e.g., ++i), Postfix increment operator (e.g., i++)
- Decrement operator (--): Prefix decrement operator (e.g., --i), Postfix decrement operator (e.g., i--)
- Ex.

```
++i;       // Postfix increment operator. it is same as 'i = i + 1'.
i++;       // Prefix increment operator. it is same as 'i = i + 1'.
--i;       // Prefix decrement operator. it is same as 'i = i - 1'.
i--;       // Postfix decrement operator. it is same as 'i = i - 1'.
```

Operators (4/9)

☑ **Increment and Decrement Operators (continued)**

- The actions of operators are
  .
- Ex.

```
int x = 5;
int  y = ++x;       // increment ‘x’ first, and assign the value to ‘y’
                       later. Thus,                 .
int z = 5;
int y = z++;        // assign ‘z’ (= 5) first, and increment ‘z’ later.
                       Thus,                 .
```

Operators (5/9)

☑ **Increment/Decrement Operators and**

- **Adding an increment/decrement operator to a pointer increases/decreases its value by the number of bytes in the type it points to.**
- **The prefix increment (e.g., '++pt'), prefix decrement (e.g., '--pt'), and dereferencing operators ('*') all have the same precedence and associate from right to left (' ').**
- **The postfix increment (e.g., 'pt++') and decrement (e.g., 'pt--') operators both have the same precedence, which is**

.

Operators (6/9)

☑ Increment/Decrement Operators and Pointers (continued)

- Ex.

```
double arr[5] = {21.1, 32.8, 23.4, 45.2, 37.4};
double *pt = arr;  // pt points to arr[0], i.e. to 21.1.
++pt;              // pt points to arr[1], i.e. to 32.8.
*++pt;     // increment pointer, take the value; i.e.,                .
              (First apply '++' to 'pt' (because the '++' is to the right of the '*') and then
               apply '*' to the new value of 'pt'.)
++*pt;     // increment the pointed to value; i.e.,                   .
                                                   .
              ('++*pt' means obtain the value that 'pt' points to and then increment
               that value.)
(*pt)++;   // increment pointed-to value; i.e.,                       .
                                                   .
              (First the pointer is dereferenced, yielding 24.4. Then the '++' operator
               increments that value to 25.4.)
*pt++;     // dereference original location, then increment pointer
                                                   .
              (The '++' operator operates on 'pt', not on '*pt', so the pointer is
               incremented. The address that gets dereferenced is the original address,
               '&arr[2]', not the new address. Thus, the value of '*pt++' is 'arr[2]' or 25.4.)
```

SYstem Design Laboratory

Operators (7/9)

- Types of relational operators

- Each relational expression reduces to the bool value and to the bool value , so these operators are well suited for use in a loop test expression.
  - Ex.

```
for (x = 20;        ; x--)                  // continue while x is greater than 5.
for (x = 1;         ; ++x)                  // continue while y is not equal to x.
for (cin >> x;          ; cin >> x))        // continue while x is 0.
```

Operators (8/9)

☑ Relational Operators (continued)

- Precedence of operators
  - The relational operators have a lower precedence than the arithmetic operators.
  - Ex. That means this expression:
    ```
    x + 3 > y - 2          // Expression 1
    ```
    corresponds ti this:
    ```
    (x + 3) > (y - 2)      // Expression 2
    ```
    and not to the following:
    ```
    x + (3 > y) - 2        // Expression 3
    ```

- Caution
  - Do not confuse comparison operator '==' with assignment operator '='.
  - Ex.
    ```
    musicians == 4         //
    musicians = 4          //
    ```

Operators (9/9)

☑ Comma (‘,’) Operator

- Be used to separate two or more expressions that are included where only one expression is expected.
- There is no precedence between expressions linked by comma operator.
- Ex.

```
++j, --i    // Two expressions count as one for syntax purposes.
int i, j;   // Comma is a separator here, not an operator.

for (j = 0, i = 0; j <= i; i--, j++)
    i = 20, j = 2 * i;                    // i = 20, j = 40
```

Compound Statements or Blocks (1/2)

- ☑ If the body of the loop is more than one statement, construct a compound statement or block.
- ☑ The block consists of and the statements they enclose and, for the purposes of syntax, .
  - ■ Ex.

```
for (int i = 0; i < 3; i++)
    cout << i << “\t”;
    cout << “\n”;
Result:

for (int j = 0; j < 3; j++)
    cout << i << “\t”;
    cout << “\n”;

Result:
```

Compound Statements or Blocks (2/2)

☑ **Variables declared in a block only can be used in the same block.**

- **Ex.**

```
#include <iostream>
using namespace std;
int main(void)
{
    int x = 20;

        int y = 100;
        cout << x << “ \ n”;
        cout << y << “ \ n”;

    cout << x << “ \ n”;
    cout << y << “ \ n”;                //        . This cannot be compiled.
    return 0;
}
```

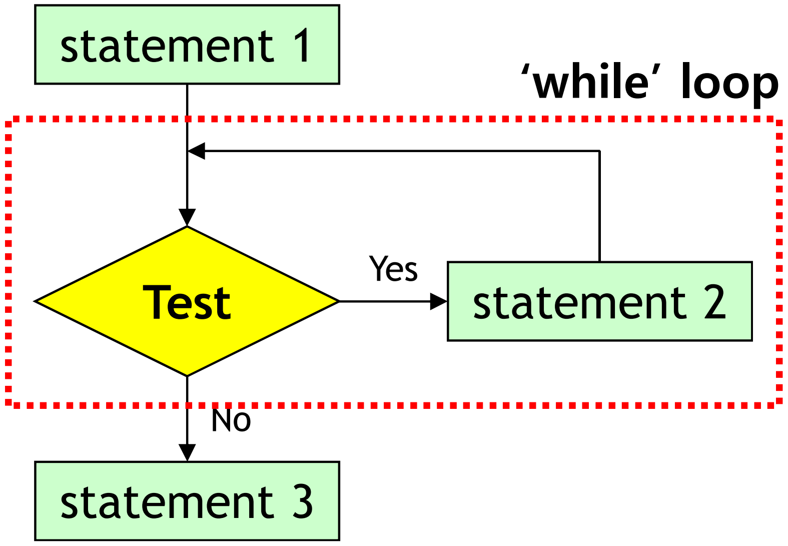
The ‘while’ Loop

- ☑ A ‘while’ loop is a ‘for’ loop stripped of the initialization and update parts; .

- ☑ Expression
  statement 1;

  statement 3;

  - ■ Ex.
```
while (time <1000)
{
    …
}
```



The ‘for’ Loop vs. The ‘while’ Loop (1/2)

☑ **In C++ the for and while loops are essentially equivalent.**

<u>General expression</u>

```
for (                                          )                  ;
{                                                   while (            )
    statement(s)                                    {
}                                                       statement(s)
                                                                        ;
                                                    }
```

<u>Expression having test-expression only</u>

```
for ( ; test-expression; ) {                        while (test-expression) {
    statement(s)                                        statement(s)
}                                                   }
```

<u>Expression for</u>

```
for (    ) {    // always true                      while ( ) {    // always true
    statement(s)                                        statement(s)
}                                                   }
```

The ‘for’ Loop vs. The ‘while’ Loop (2/2)

- ☑ The ‘for’ loop
  - We can because the ‘for’ loop has the initial expression, test expression, and update expression.
- ☑ The ‘while’ loop
  - We can

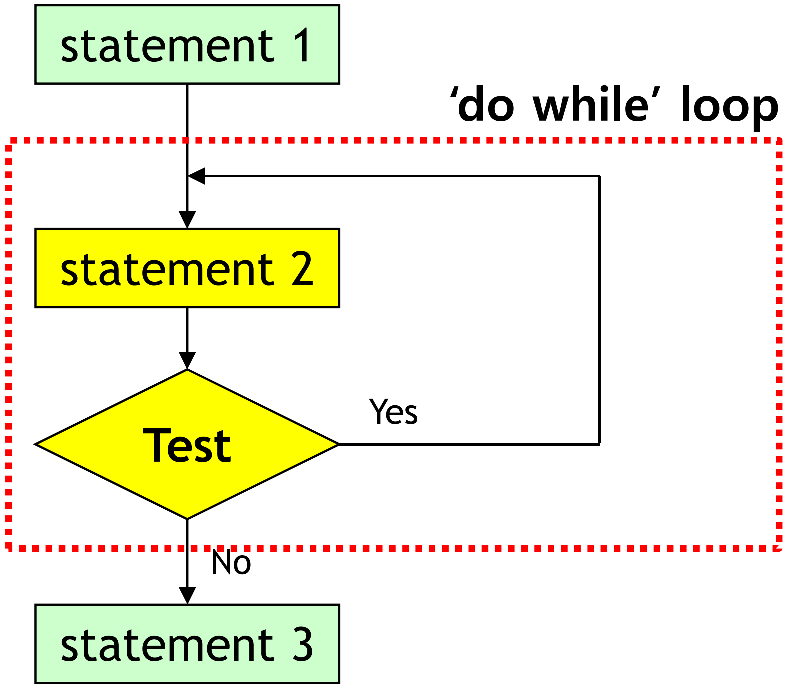
.

The ‘do while’ Loop

to see whether it should continue looping.

- ☑ The loop always because its program flow must pass through the body of the loop before reaching the test.
- ☑ Expression
  statement 1;

  statement 3;



Loops and Text Input

☑ Text Input Using 'cin'

- The 'cin' doesn't read white spaces ('Space' or 'Tab') and the new line character (' \ n').
- The input to cin is buffered. That is, the characters we type don't get sent to the program until we press 'Enter'.

☑ Text Input Using 'cin.get(char)'

- The 'cin.get(char)' reads all characters including white spaces and the new line character.

- **Review**

```
cin >> name;               // Read by a word. Ignore the initial white-space character.
                              Consider white-space character as the end of the string.
                        ;  // Read up to 50 characters or until a newline character.
                                                                                   .
                    ;      // Read up to 50 characters or until a newline character.
                                                                                  .
        ;                  // Read one character regardless of its type.
```

Nested Loops and Two-Dimensional Arrays (1/2)

. Nested loops provide a natural way to process two-dimensional arrays.

☑ **Initialization of One-dimensional Arrays**

- Ex. int atom[5] = {1, 2, 3, 4, 5};

☑ **Initialization of Two-dimensional Arrays**

- Initialize as one-dimensional arrays
  - Ex.
    int atom[2][3] = {1, 2, 3, 4, 5, 6};
    or
    atom[0][0]=1; atom[0][1]=2; atom[0][3]=3;
    atom[1][0]=4; atom[1][1]=5; atom[1][2]=6;
- Initialize as two-dimensional arrays
  - Ex.
    int atom[2][3] = {{1, 2, 3}, {4, 5, 6}}

values of atom[1]

values of atom[0]

Nested Loops and Two-Dimensional Arrays (2/2)

☑ Access of Two-dimensional Arrays with Nested Loops

- Ex.

```
for (int i = 0; i < 2; i++)
{
        for (int j = 0; j < 3; j++)
            cout << atom[i][j] << "\t";
        cout << "\n";
}
```

Result:

```
1       2       3
4       5       6
```

Summary (1/2)

☑ C++ offers three varieties of loops:

.

,
meaning that they examine the test condition before executing the statements in the body of the loop.

, meaning that it
examines the test condition after executing the statements in the body of the loop.

☑ The syntax for each loop calls for the loop body to consist of a single statement. However,
, formed by enclosing several statements within paired curly braces.

Summary (2/2)

- ☑ C++ offers various operators for loops:

  .

- ☑ Relational expressions by relational operators, which compare two values, are often used as loop test conditions. Relational expressions are formed by using one of the six relational operators: <, <=, ==, >=, >, or !=. Relational expressions evaluate to the type bool values true and false.

  . Nested loops provide a natural way to process two-dimensional arrays.

Practice 1

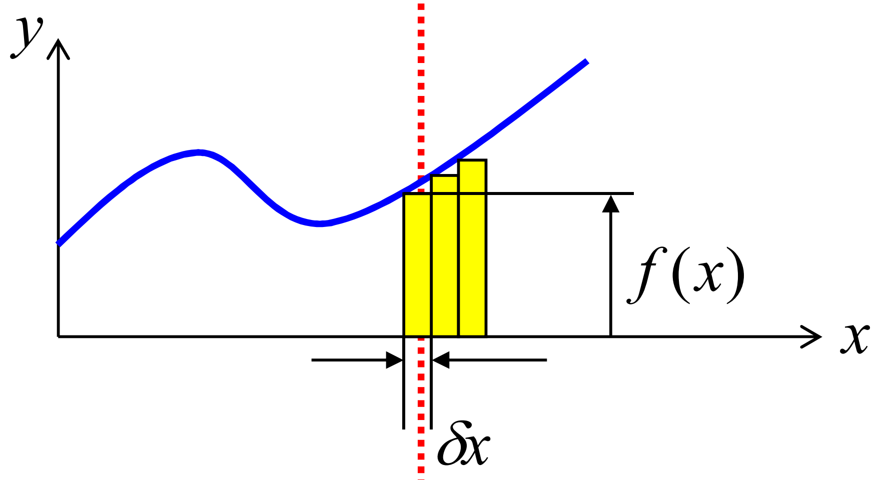
☑ Make a program that calculates the value of the integral as below. To do this, use the mensuration by parts.

$$y = \int_1^2 \log_{10}(x + \sin^2 x)dx$$

Preprocessor directives

```
#include <cmath>
// This header file has definitions of functions as below.
// 'float sin(float);' for sin function
// 'float log10(float);' for log10 (common logarithm) function.

int main(void)
{
    declare variables.
    delta = 1.0 / N;
    ans = 0;
    for (i = 0; i < N; i++) {
        x = 1 + delta * i;
        ans = ans + log(x + sin2(x)) * delta;
    }
}
```



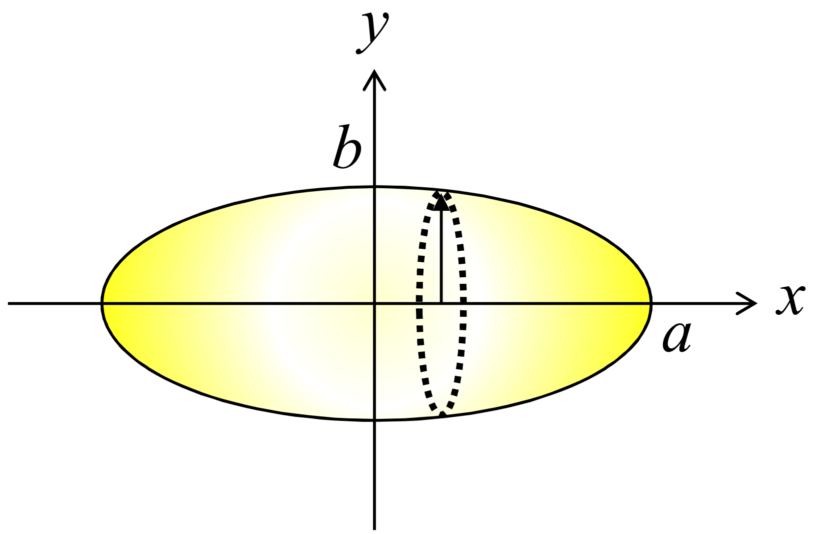
$$Area = \sum_i f(x)\delta x$$

Practice 2

☑ Make a program that calculates the volume and the surface area of the ellipsoidal solid.

- Get the values of a and b when the program is running.
- A formula for a ellipse is as below.

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$$



Practice 3

- ☑ Make a program that reads exam scores for Korean, Mathematics, and English of 10 students and calculates the average score of each student and the average score of each subject for all students.
  - Use two-dimensional arrays for storing the scores of students.

Practice 4

- ☑ Make a sorting program that sorts an array as below, in descending order, and stores the result to the array ‘a’.
  - int a[10] = {7, 2, 5, 9, 4, 1, 2, 10, 5, 6};
  - Refer to the sort algorithms as below.

```
for i = 0 to 8 {
    max = a[i];
    for j = (i + 1) to 9
        if (a[j] > max)
            swap a[j] and max;
    a[i] = max;
}
```

Practice 5

☑ **Make a program that calculates the multiplication of matrices.**

- **In matrices, C = A·B.**

$$\left[c_{ij}\right] = \left[\sum_{k} a_{ik} b_{kj}\right]$$

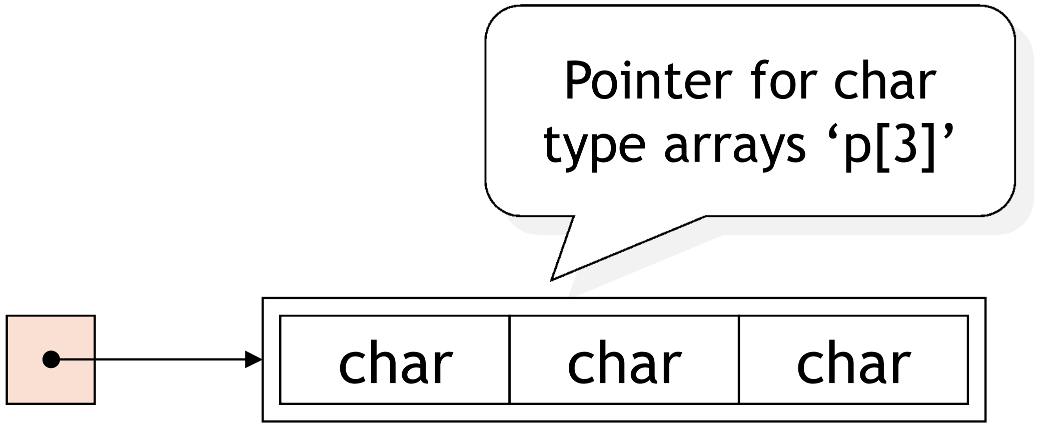
- **Use two-dimensional arrays for storing components of the matrix.**

Reference Slides

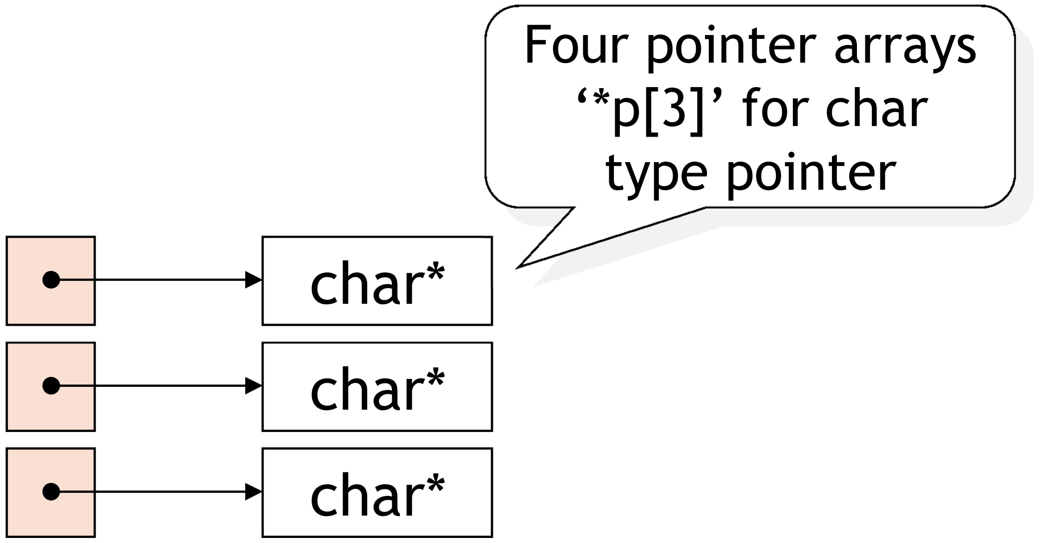
[Appendix] Pointer for Arrays vs. Pointer Arrays

☑ ‘[]’ is prior to the ‘*’.

```
// Pointer for array
char (*p)[3];
```



```
// Pointer arrays
char *p[3];
```


Four pointer arrays ‘*p[3]’ for char type pointer
char*
char*
char*