“C Control Statements : Looping”

Using Bloodshed Dev-C++

Heejin Park

Hanyang University

Introduction(1/2)

- Revisiting the `while` Loop
- The `while` Statement
- Which Is Bigger: Using Relational Operators and Expressions
- Indefinite Loops and Counting Loops
- The `for` Loop
- More Assignment Operators
- The Comma Operator

Introduction(2/2)

- An Exit-Condition Loop: `do while`
- Which Loop?
- Nested Loops
- Introducing Arrays
- A Loop Example Using a Function Return Value

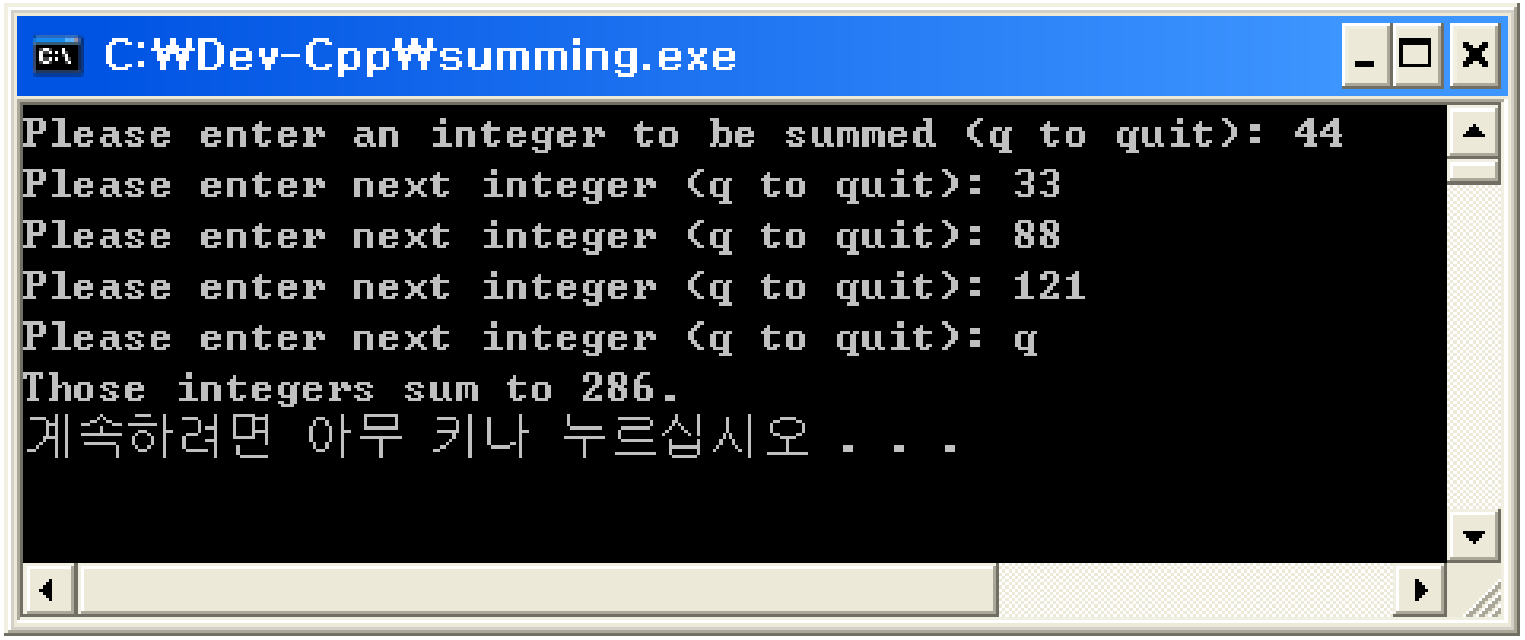
Revisiting the `while` Loop

- **The summing.c Program**

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    long num;
    long sum = 0L;      /* initialize sum to zero   */
    int status;

    printf("Please enter an integer to be summed ");
    printf("(q to quit): ");
    status = scanf("%ld", &num);

    while (status == 1) /* == means "is equal to"   */
    {
        sum = sum + num;
        printf("Please enter next integer (q to quit): ");
        status = scanf("%ld", &num);
    }
    printf("Those integers sum to %ld.\n", sum);

    system("pause");
    return 0;
}
```

Revisiting the `while` Loop

- The summing.c Program



```
Please enter an integer to be summed (q to quit): 44
Please enter next integer (q to quit): 33
Please enter next integer (q to quit): 88
Please enter next integer (q to quit): 121
Please enter next integer (q to quit): q
Those integers sum to 286.
계속하려면 아무 키나 누르십시오 . . .
```

Revisiting the `while` Loop

Program Comments

- Now let's take a closer look at the program structure. You can summarize it as follows:

```
initialize sum to 0
prompt user
read input

while the input is an integer,
     add the input to sum,
     prompt user,
     then read next input
after input completes, print sum
```

Revisiting the `while` Loop

Program Comments

- You can think of the following as a standard format for a loop:

```
get first value to be tested

while the test is successful
     process value
     get next value
```

Revisiting the `while` Loop

- **C-Style Reading Loop**

```
status = scanf("%ld", &num);

while (status == 1)
{
        /* loop actions */
        status = scanf("%ld", &num);
}
```

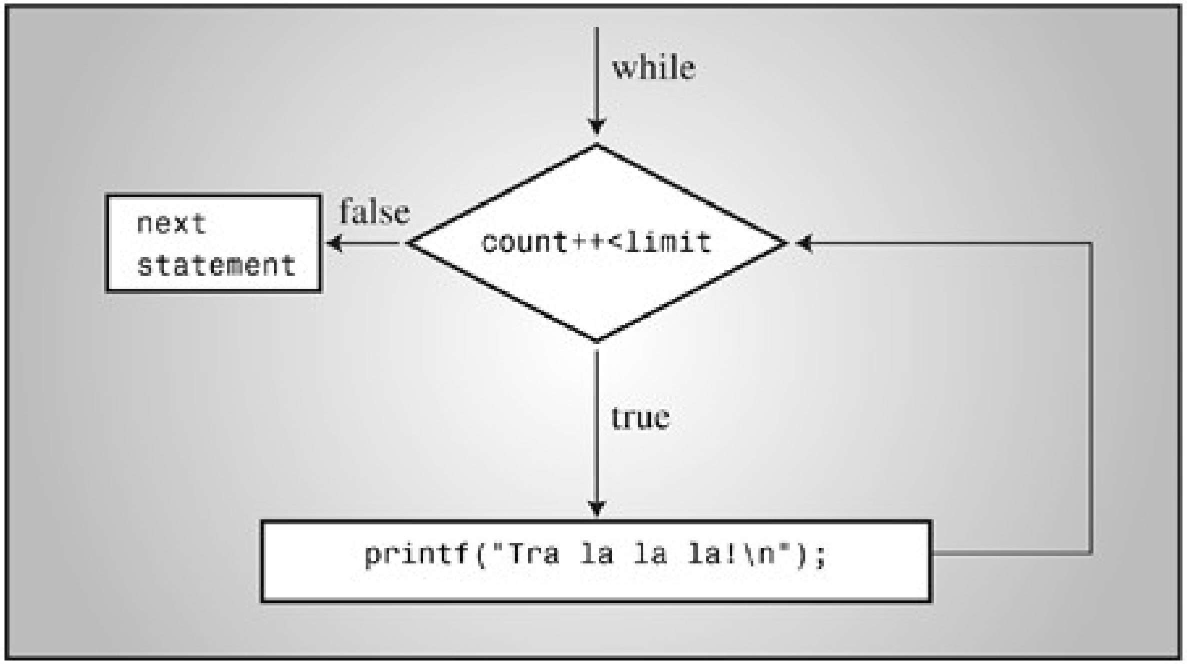
- **can be replaced by the following:**

```
while (scanf("%ld", &num) == 1)
{
        /* loop actions */
}
```

The `while` Statement

- **Structure of the while loop**

```
while(count++ < limit){
              printf("Tra la la la!\n");
}
next statement
```



The while Statement

Terminating a while Loop

- It is important to realize that the decision to terminate the loop or to continue takes place only when the test condition is evaluated.
- Consider these examples

```
index = 1;
while (index < 5)
   printf("Good morning!\n");
```

```
index = 1;
while (--index < 5)
   printf("Good morning!\n");
```

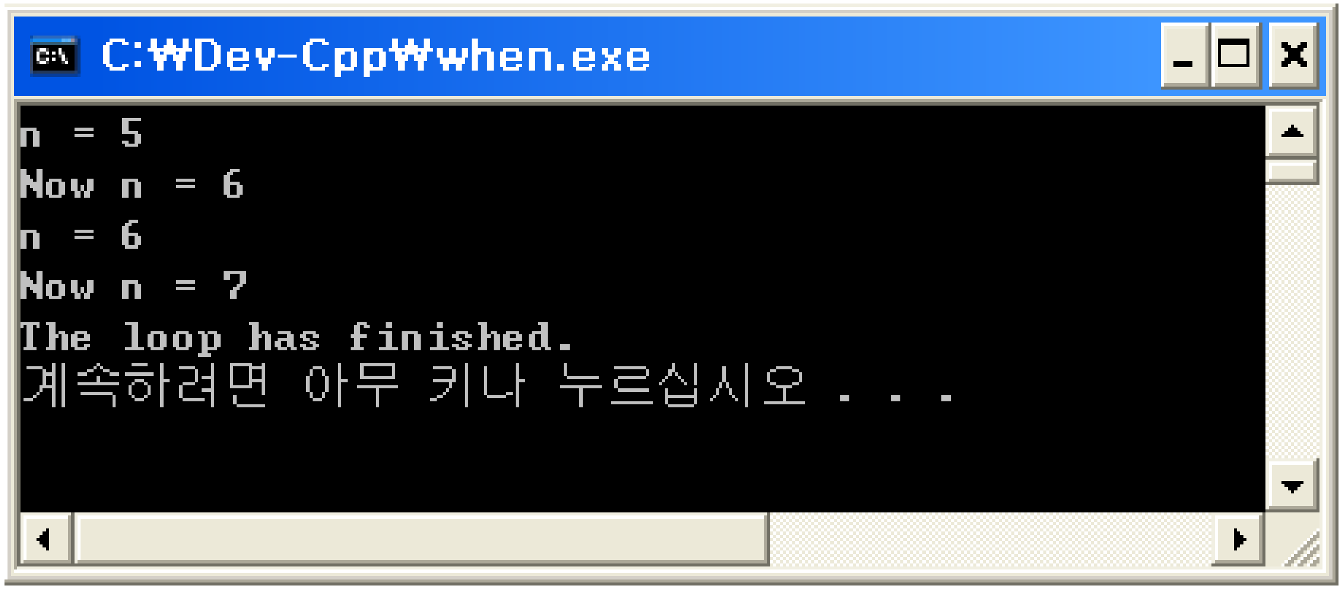
The while Statement

The when.c Program

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int n = 5;

    while (n < 7)                           // line 7
    {
        printf("n = %d\n", n);
        n++;                                // line 10
        printf("Now n = %d\n", n);          // line 11
    }

    printf("The loop has finished.\n");

    system("pause");
    return 0;
}
```

The while Statement

- **The when.c Program**



```
n = 5
Now n = 6
n = 6
Now n = 7
The loop has finished.
계속하려면 아무 키나 누르십시오 . . .
```

The while Statement

while: An Entry-Condition Loop

- In a situation such as the following, the body of the loop is never entered
- because the condition is false to begin with:

```
index = 10;
while (index++ < 5)
    printf("Have a fair day or better.\n");
```

- **Change the first line to**

```
index = 3;
```

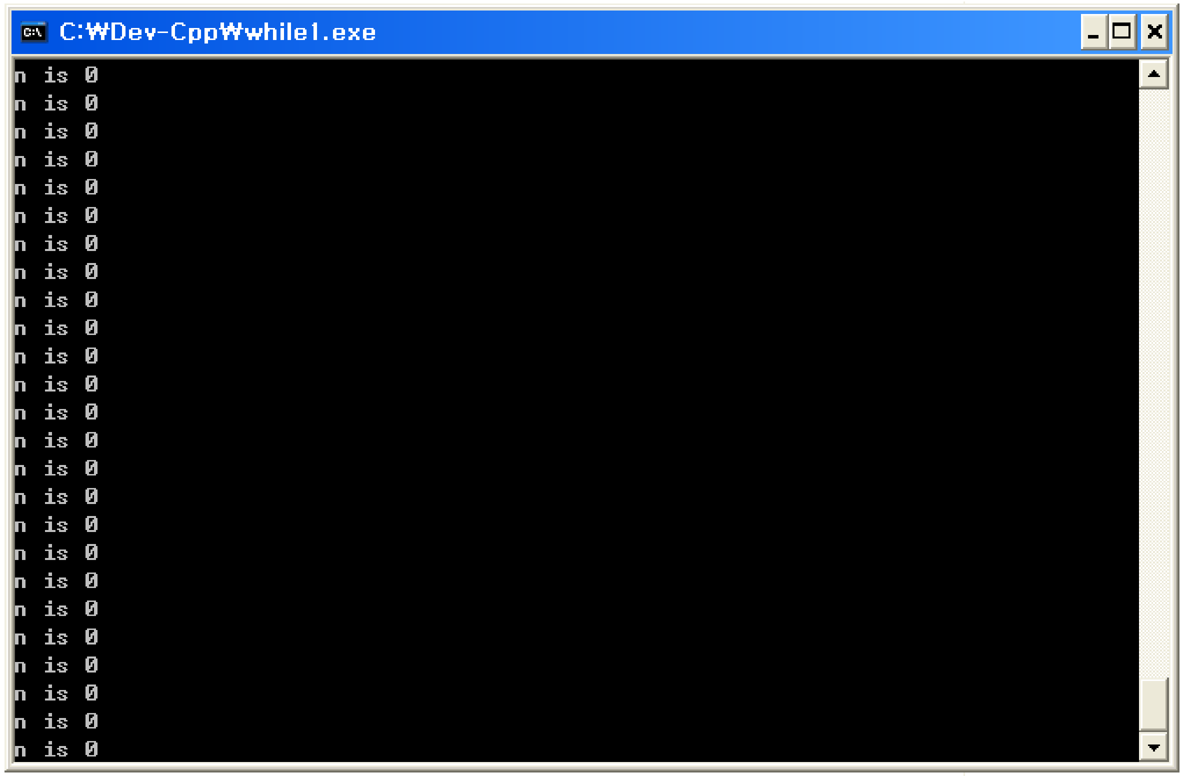
The while Statement

- **The while1.c Program**

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int n = 0;

    while (n < 3)
        printf("n is %d\n", n);
        n++;
    printf("That's all this program does\n");

    system("pause");
    return 0;
}
```

The while Statement

- **The while1.c Program**



C:₩Dev-Cpp₩while1.exe

```
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
n is 0
```

The while Statement

Syntax Points

- Suppose you want to skip over input to the first character that isn't whitespace or a digit.
- You can use a loop like this:

```
while (scanf("%d", &num) == 1)
  ;     /* skip integer input */
```

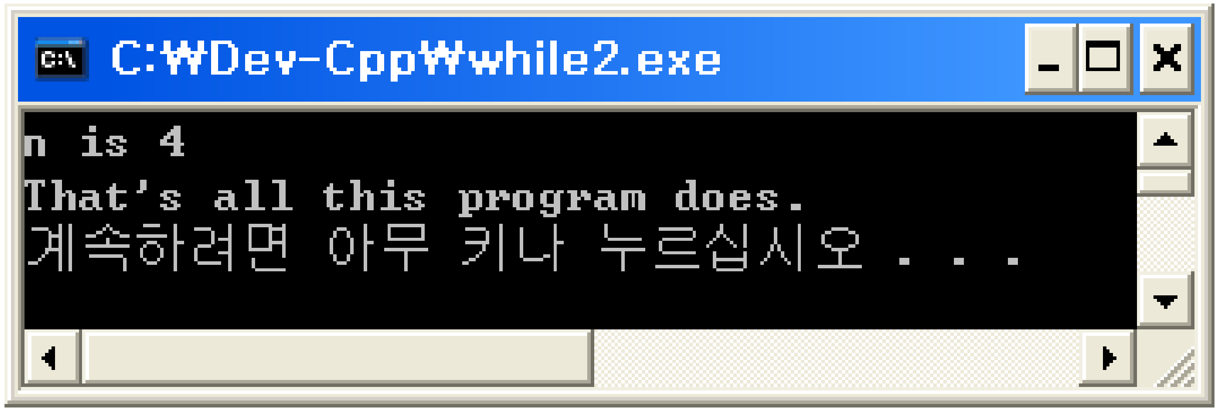
The while Statement

- **The while2.c Program**

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int n = 0;

    while (n++ < 3);
        printf("n is %d\n", n);

    printf("That's all this program does\n");

    system("pause");
    return 0;
}
```

The while Statement

- The while2.c Program



```
n is 4
That's all this program does.
계속하려면 아무 키나 누르십시오 . . .
```

The while Statement

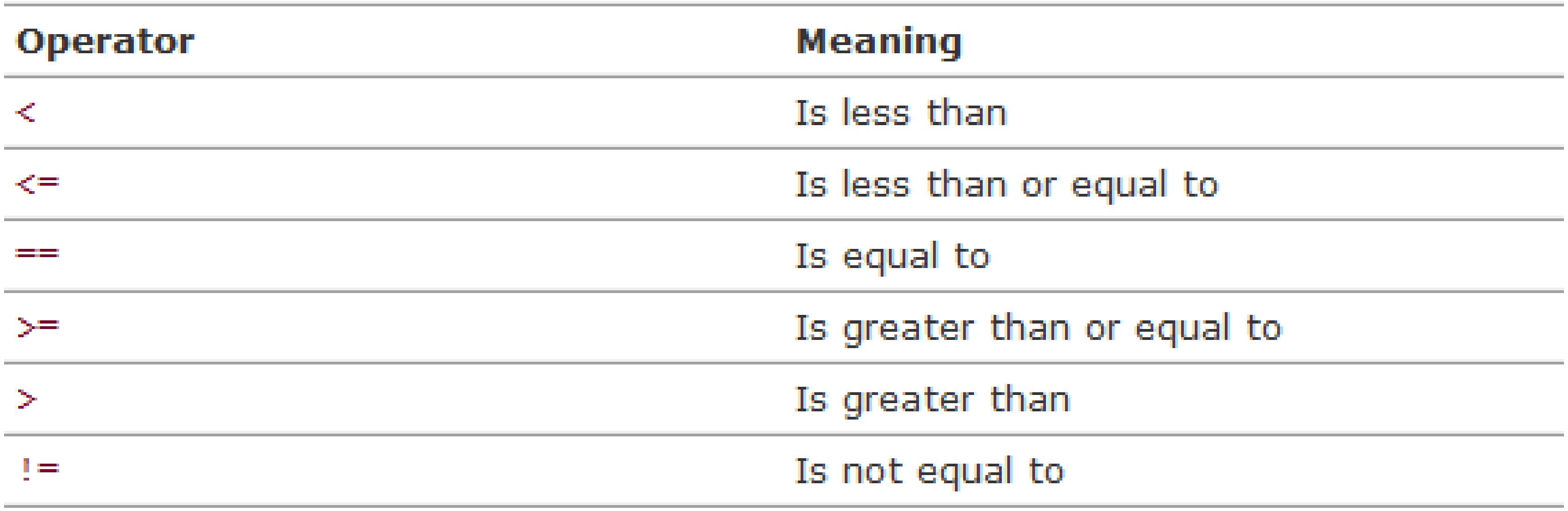
The while2.c Program

- You can use a loop like this:

```
while (scanf("%d", &num) == 1)

   ;     /* skip integer input */
```

Which Is Bigger: Using Relational Operators and Expressions

- **Relational Operators**



Operator	Meaning
`<`	Is less than
`<=`	Is less than or equal to
`==`	Is equal to
`>=`	Is greater than or equal to
`>`	Is greater than
`!=`	Is not equal to

Which Is Bigger: Using Relational Operators and Expressions

Relational Operators

- Here are three unrelated statements containing examples of relational expressions.

```
while (number < 6){
    printf("Your number is too small.\n");
    scanf("%d", &number);
}
```

```
while (ch != '$'){
     count++;
     scanf("%c", &ch);
}
```

```
while (scanf("%f", &num) == 1)
    sum = sum + num;
```

Which Is Bigger: Using Relational Operators and Expressions

The cmpflt.c Program

- The `fabs()` function
- Declared in the `math.h` header file
- Can be handy for floating-point tests.
- Returns the absolute value of a floating-point value.
  - That is, the value without the algebraic sign.

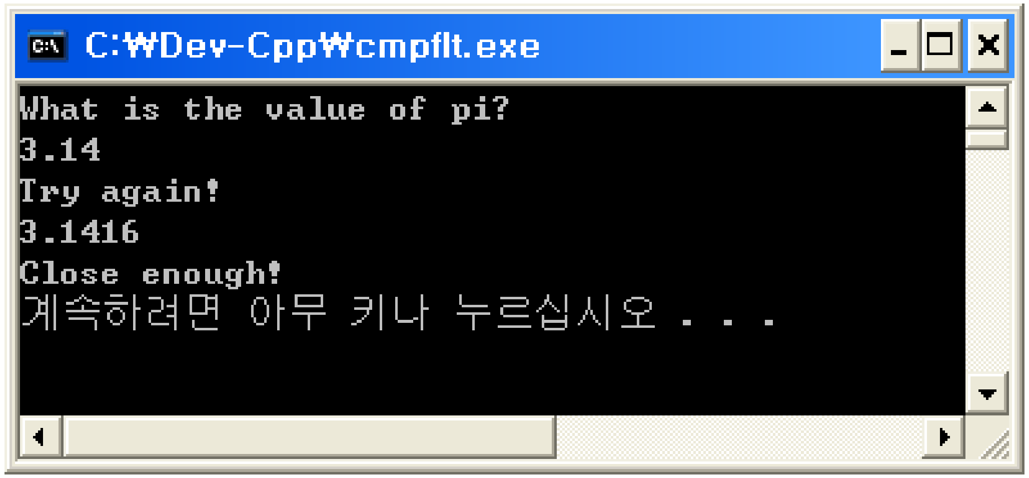
Which Is Bigger: Using Relational Operators and Expressions

- **The cmpflt.c Program**

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main(void)
{
    const double ANSWER = 3.14159;
    double response;

    printf("What is the value of pi?\n");
    scanf("%lf", &response);

    while(fabs(response - ANSWER) > 0.0001)
    {
        printf("Try again!\n");
        scanf("%lf", &response);
    }

    printf("Close enough!\n");

    system("pause");
    return 0;
}
```

Which Is Bigger: Using Relational Operators and Expressions

- The cmpflt.c Program



```
What is the value of pi?
3.14
Try again!
3.1416
Close enough!
계속하려면 아무 키나 누르십시오 . . .
```

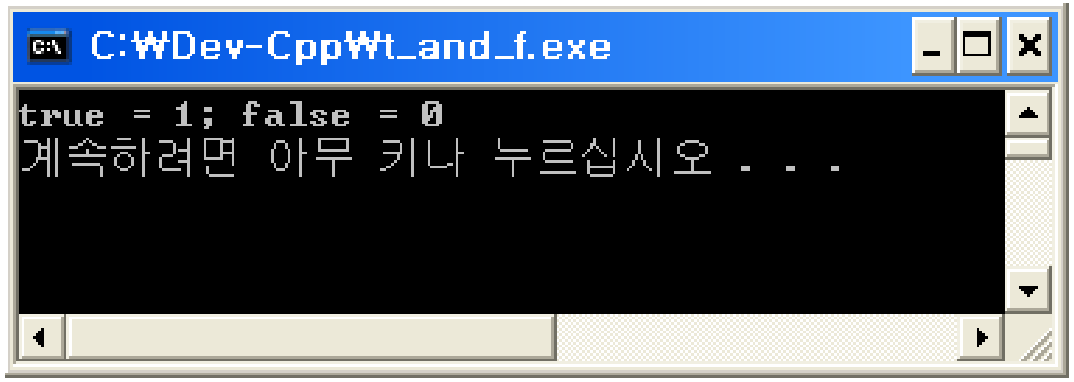
Which Is Bigger: Using Relational Operators and Expressions

- The t_and_f.c Program(What Is Truth?)

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int true_val, false_val;

    true_val = (10 > 2);     /* value of a true relationship  */
    false_val = (10 == 2);   /* value of a false relationship */
    printf("true = %d; false = %d \n", true_val, false_val);

    system("pause");
    return 0;
}
```

Which Is Bigger: Using Relational Operators and Expressions

- The t_and_f.c Program(What Is Truth?)



```
true = 1; false = 0
계속하려면 아무 키나 누르십시오 . . .
```

Which Is Bigger: Using Relational Operators and Expressions

What Is Truth?

- Loops that are meant to run forever.

```
while (1)
{
   ...
}
```

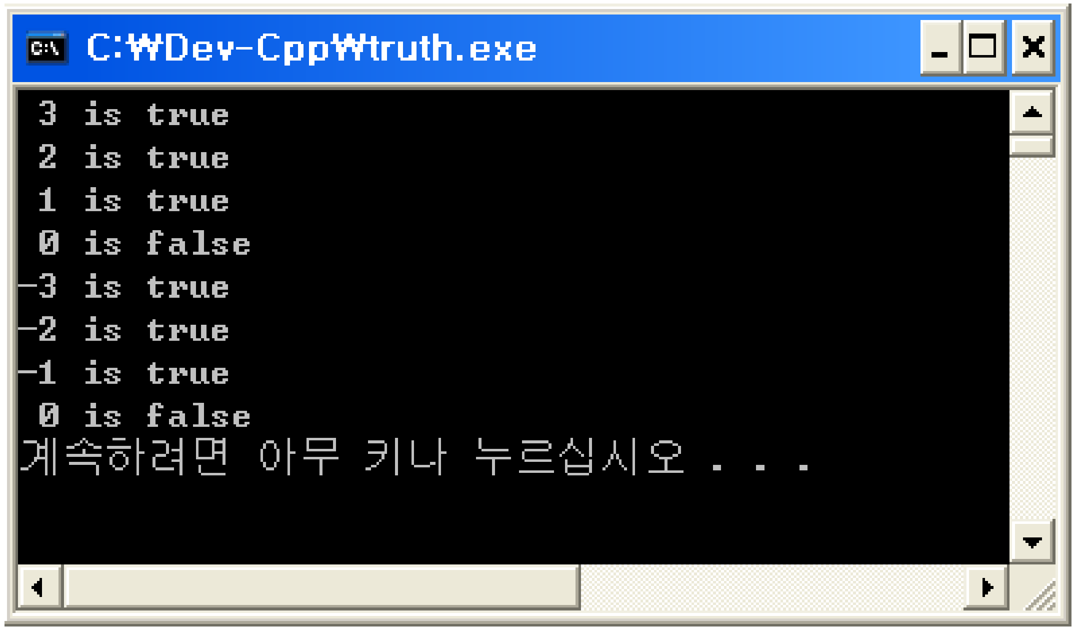
Which Is Bigger: Using Relational Operators and Expressions

- The truth.c Program (What Else Is True?)

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int n = 3;

    while (n)
        printf("%2d is true\n", n--);
    printf("%2d is false\n", n);

    n = -3;
    while (n)
        printf("%2d is true\n", n++);
    printf("%2d is false\n", n);

    system("pause");
    return 0;
}
```

Which Is Bigger: Using Relational Operators and Expressions

- The truth.c Program (What Else Is True?)


C:₩Dev-Cpp₩truth.exe

```
 3 is true
 2 is true
 1 is true
 0 is false
-3 is true
-2 is true
-1 is true
 0 is false
계속하려면 아무 키나 누르십시오 . . .
```

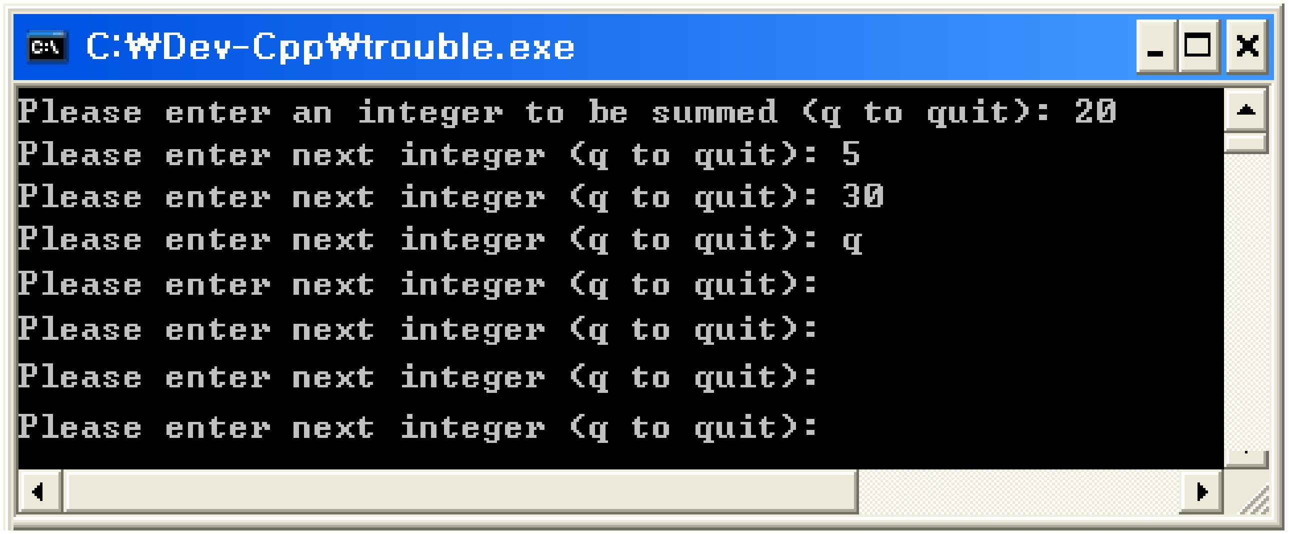
Which Is Bigger: Using Relational Operators and Expressions

- The trouble.c Program (Troubles with Truth)

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    long num;
    long sum = 0L;
    int status;

    printf("Please enter an integer to be summed ");
    printf("(q to quit): ");
    status = scanf("%ld", &num);

    while (status = 1)
    {
        sum = sum + num;
        printf("Please enter next integer (q to quit): ");
        status = scanf("%ld", &num);
    }
    printf("Those integers sum to %ld.\n", sum);

    system("pause");
    return 0;
}
```

Which Is Bigger: Using Relational Operators and Expressions

- The trouble.c Program (Troubles with Truth)

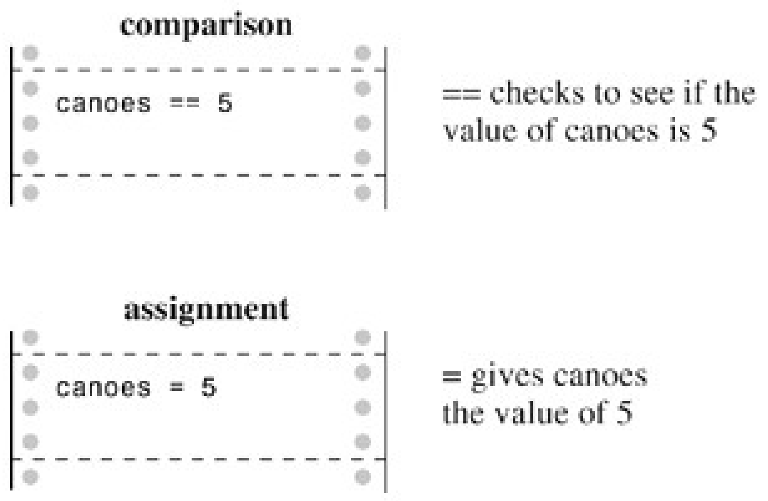


```
Please enter an integer to be summed (q to quit): 20
Please enter next integer (q to quit): 5
Please enter next integer (q to quit): 30
Please enter next integer (q to quit): q
Please enter next integer (q to quit):
Please enter next integer (q to quit):
Please enter next integer (q to quit):
Please enter next integer (q to quit):
```

- After this input,
  printing “Please enter next integer (q to quit): ”
  will be continued infinitely.

Which Is Bigger: Using Relational Operators and Expressions

- The relational operator == and the assignment operator =.


comparison
canoes == 5
== checks to see if the value of canoes is 5
assignment
canoes = 5
= gives canoes the value of 5

```
canoes = 5        ⟵ Assigns the value 5 to canoes
canoes == 5       ⟵ Checks to see whether canoes has the value 5
5 = canoes        ⟵ syntax error
5 == canoes       ⟵ Checks to see whether canoes has the value 5
```

Which Is Bigger: Using Relational Operators and Expressions

The New _Bool Type

- If your system does not yet support the _Bool type,
- you can replace _Bool with `int`, and the example will work the same.
- A _Bool variable can only have a value of 1 (true) or 0 (false).

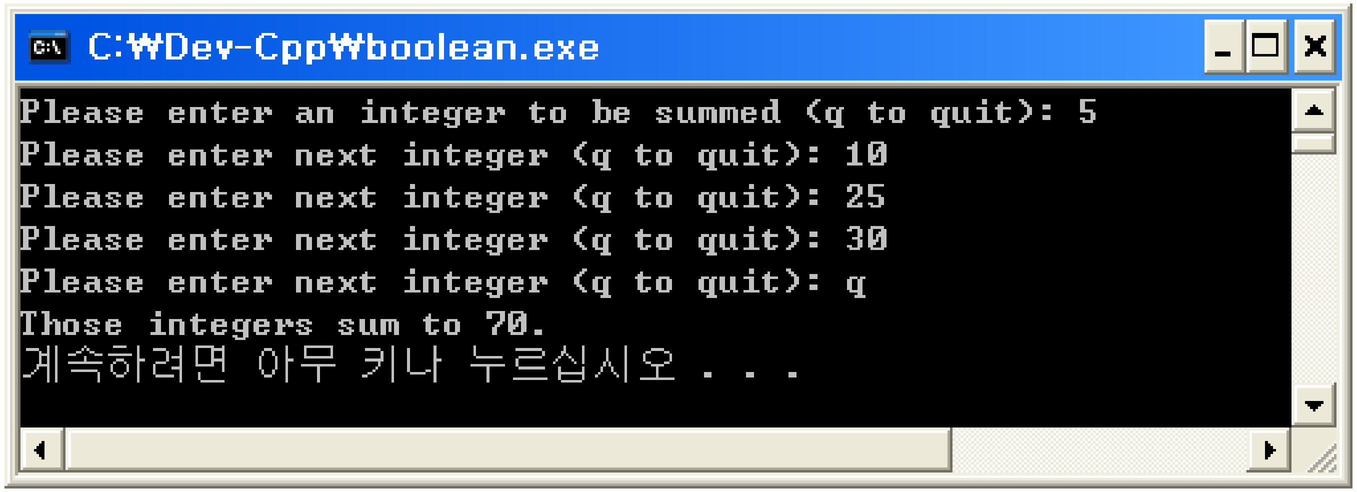
Which Is Bigger: Using Relational Operators and Expressions

- The boolean.c Program

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

int main(void)
{
    long num;
    long sum = 0L;
    _Bool input_is_good;

    printf("Please enter an integer to be summed ");
    printf("(q to quit): ");
    input_is_good = (scanf("%ld", &num) == 1);

    while (input_is_good)
    {
        sum = sum + num;
        printf("Please enter next integer (q to quit): ");
        input_is_good = (scanf("%ld", &num) == 1);
    }
    printf("Those integers sum to %ld.\n", sum);

    system("pause");
    return 0;
}
```

Which Is Bigger: Using Relational Operators and Expressions

- The boolean.c Program



```
Please enter an integer to be summed (q to quit): 5
Please enter next integer (q to quit): 10
Please enter next integer (q to quit): 25
Please enter next integer (q to quit): 30
Please enter next integer (q to quit): q
Those integers sum to 70.
계속하려면 아무 키나 누르십시오 . . .
```

Which Is Bigger: Using Relational Operators and Expressions

The boolean.c Program

- Note how the code assigns the result of a comparison to the variable:

```
input_is_good = (scanf("%ld", &num) == 1);
```

- Also note how the choice of name for the variable makes the while loop test easy to understand:

```
  while (input_is_good)
```

Which Is Bigger: Using Relational Operators and Expressions

Precedence of Relational Operators

- The precedence of the relational operators is <u>less than </u>that of the arithmetic operators.
- Ex)

```
x > y + 2
```

- Means the same as

```
x > (y + 2)
```

- It also means that

```
x = y > 2
```

- means

```
x = (y > 2)
```

Which Is Bigger: Using Relational Operators and Expressions

Precedence of Relational Operators

- The relational operators have a greater precedence than the assignment operator.

```
x_bigger = x > y;
```

```
<  <=  >  >=
```

- means

```
x_bigger = (x > y);
```

```
==  !=
```

Which Is Bigger: Using Relational Operators and Expressions

Precedence of Relational Operators

- The relational operators are themselves organized into two different precedences.
- Higher precedence group:

```
< <= > >=
```

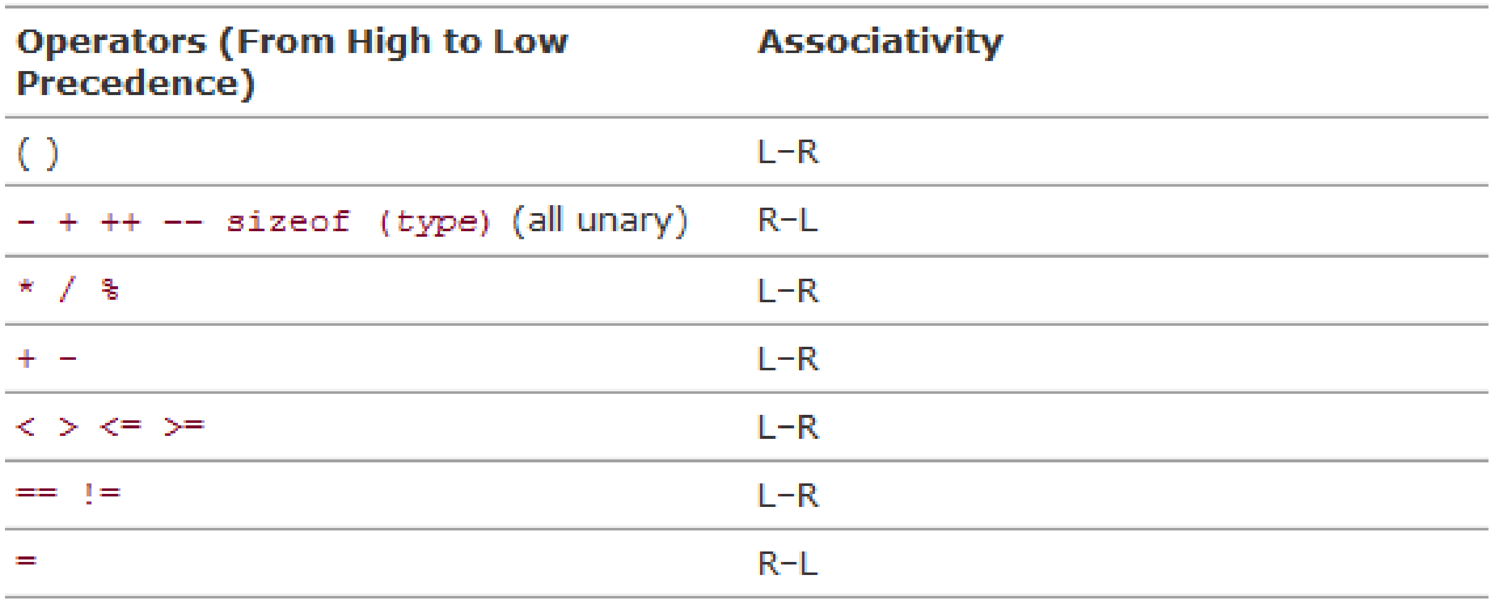
- Lower precedence group:

```
== !=
```

- Like most other operators, the relational operators associate from left to right.
- `ex != wye == zee` is the same as `(ex != wye) == zee`

Which Is Bigger: Using Relational Operators and Expressions

- **Precedence of Relational Operators**



Operators (From High to Low Precedence)	Associativity
`( )`	L-R
`- + ++ -- sizeof (type)` (all unary)	R-L
`* / %`	L-R
`+ -`	L-R
`< > <= >=`	L-R
`== !=`	L-R
`=`	R-L

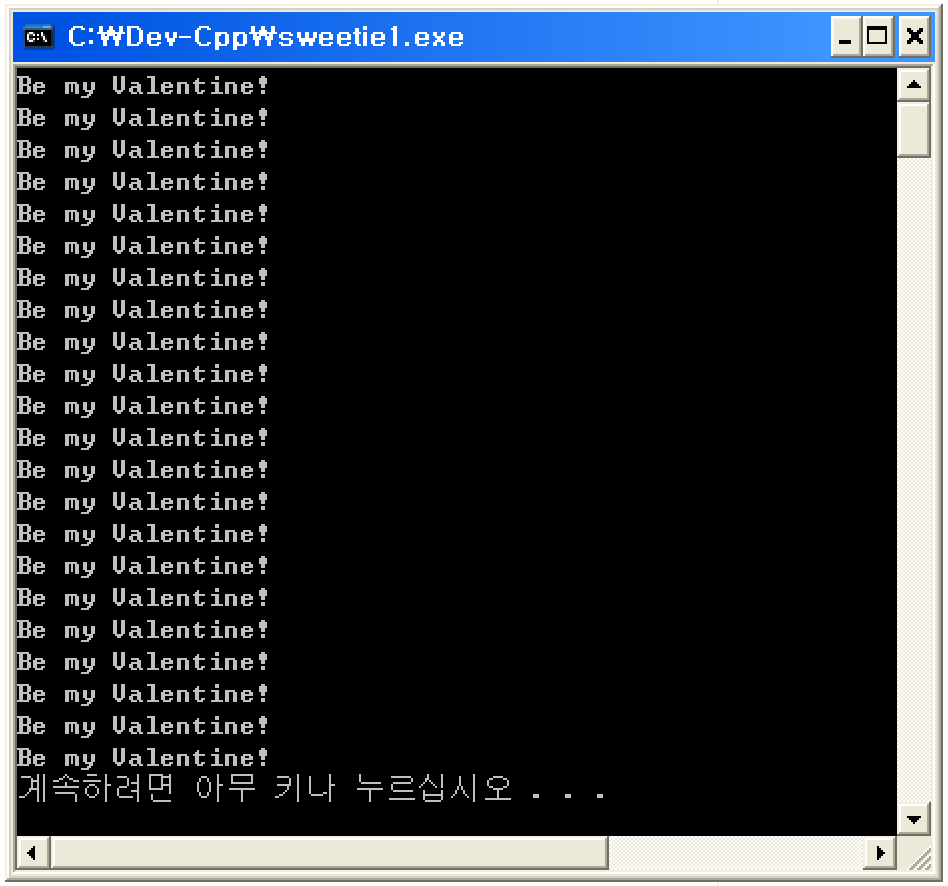
Indefinite Loops and Counting Loops

- **The sweetie1.c Program**

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    const int NUMBER = 22;
    int count = 1;                      // initialization

    while (count <= NUMBER)             // test
    {
        printf("Be my Valentine!\n");   // action
        count++;                        // update count
    }
    system("pause");
    return 0;
}
```

Indefinite Loops and Counting Loops

- The sweetie1.c Program



```
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
계속하려면 아무 키나 누르십시오 . . .
```

Indefinite Loops and Counting Loops

The sweetie1.c Program

- Three actions are involved in setting up a loop that is to be repeated a fixed number of times.

  1) A counter must be initialized.

  2) The counter is compared with some limiting value.

  3) The counter is incremented each time the loop is traversed.

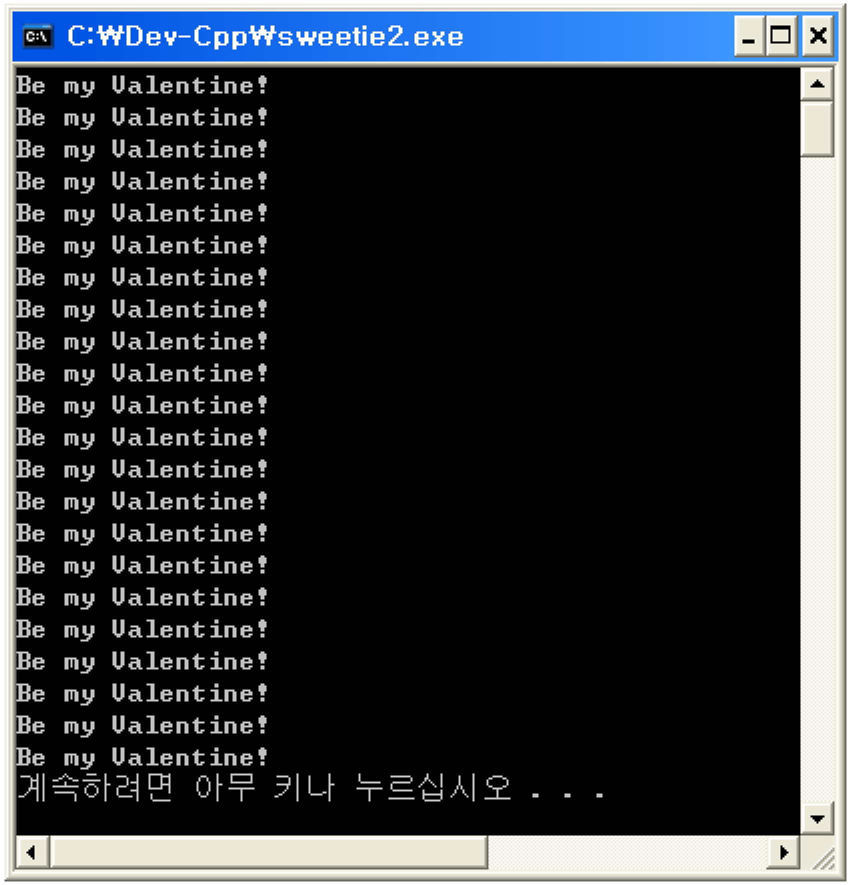
The for Loop

- **The sweetie2.c Program**

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    const int NUMBER = 22;
    int count;

    for (count = 1; count <= NUMBER; count++)
        printf("Be my Valentine!\n");

    system("pause");
    return 0;
}
```

The for Loop

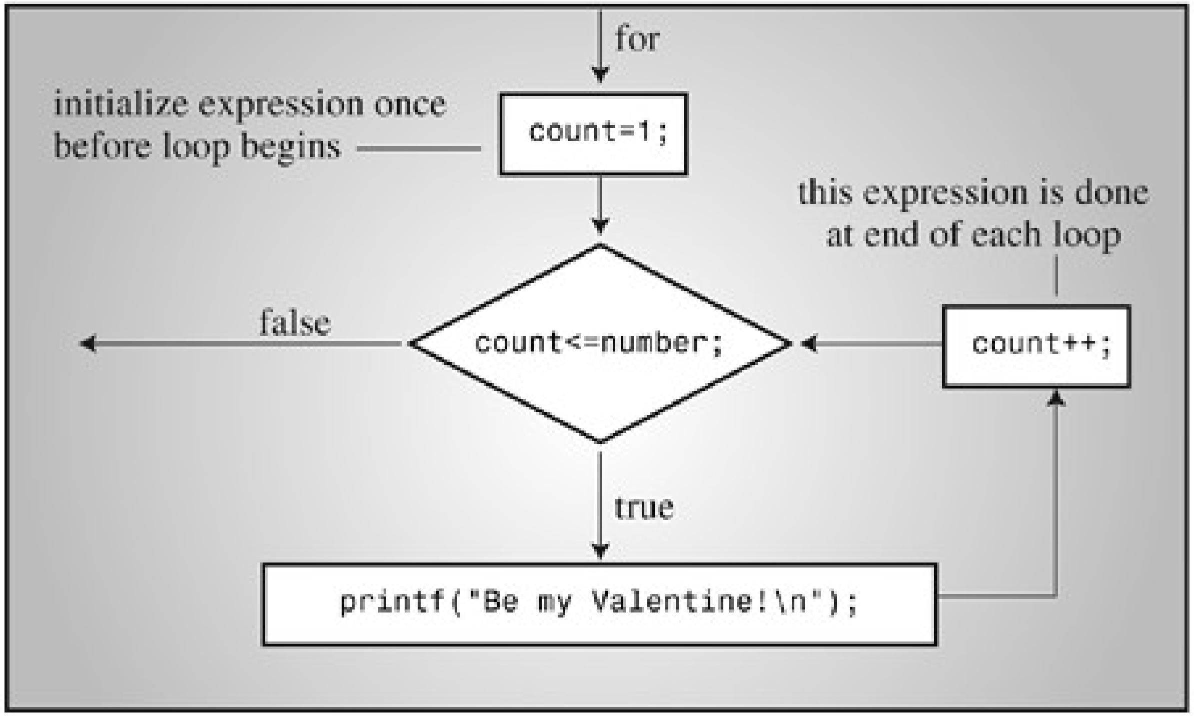
- The sweetie2.c Program



```
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
Be my Valentine!
계속하려면 아무 키나 누르십시오 . . .
```

The for Loop

- **Structure of a for loop**



for
initialize expression once before loop begins
count=1;
this expression is done at end of each loop
false
count<=number;
count++;
true
printf("Be my Valentine!\n");

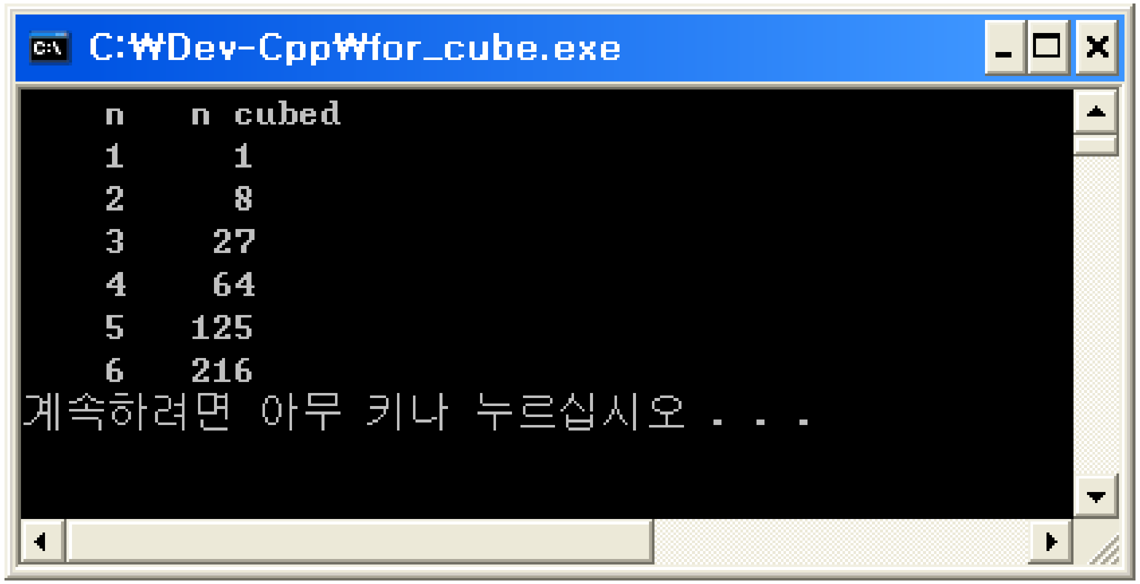
The for Loop

- **The for_cube.c Program**

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int num;
    printf("    n   n cubed\n");

    for (num = 1; num <= 6; num++)
        printf("%5d %5d\n", num, num*num*num);

    system("pause");
    return 0;
}
```

The for Loop

The for_cube.c Program



```
    n    n cubed
    1      1
    2      8
    3     27
    4     64
    5    125
    6    216
계속하려면 아무 키나 누르십시오 . . .
```

The `for` Loop

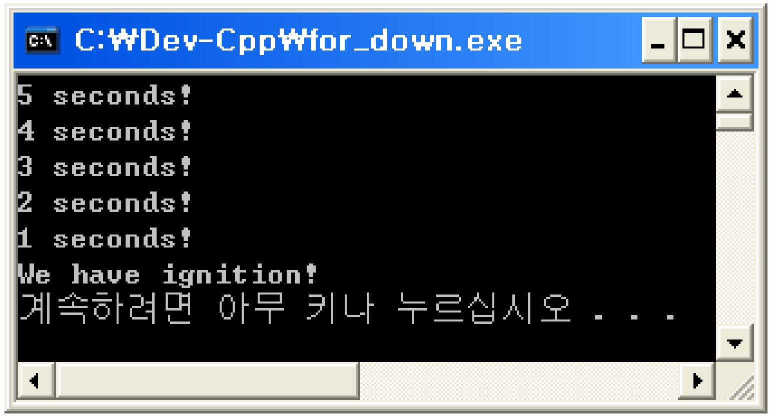
- **Using `for` for Flexibility**
  - Although the for loop looks similar to the FORTRAN DO loop, the Pascal FOR loop, and the BASIC FOR...NEXT loop.
  - This **flexibility** stems from how the three expressions in a for specification can be used.
  - Here are **nine variations**

The for Loop

Using `for` for Flexibility

1) You can use the decrement operator to count down instead of up:

```c
/* for_down.c */
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    int secs;

    for (secs = 5; secs > 0; secs--)
        printf("%d seconds!\n", secs);
    printf("We have ignition!\n");

    system("pause");
    return 0;
}
```



C:₩Dev-Cpp₩for_down.exe

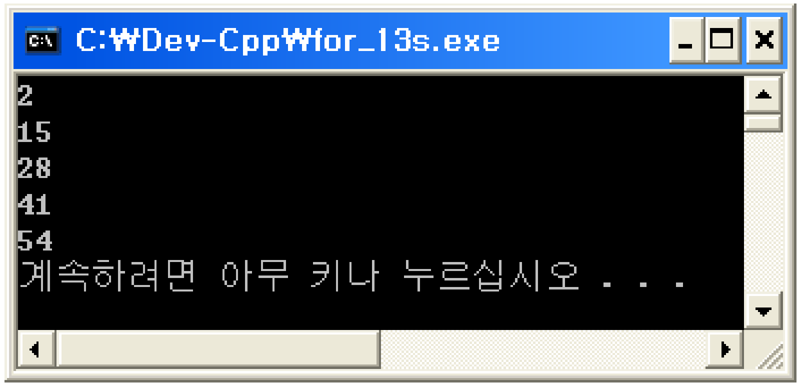
```
5 seconds!
4 seconds!
3 seconds!
2 seconds!
1 seconds!
We have ignition!
계속하려면 아무 키나 누르십시오 . . .
```

The for Loop

Using `for` for Flexibility

2) You can count by twos, tens, and so on, if you want:

```c
/* for_13s.c */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int n;          /* count by 13s */

    for (n = 2;  n < 60; n = n + 13)
        printf("%d \n", n);

    system("pause");
    return 0;
}
```



C:₩Dev-Cpp₩for_13s.exe

```
2
15
28
41
54
계속하려면 아무 키나 누르십시오 . . .
```

The for Loop

Using `for` for Flexibility

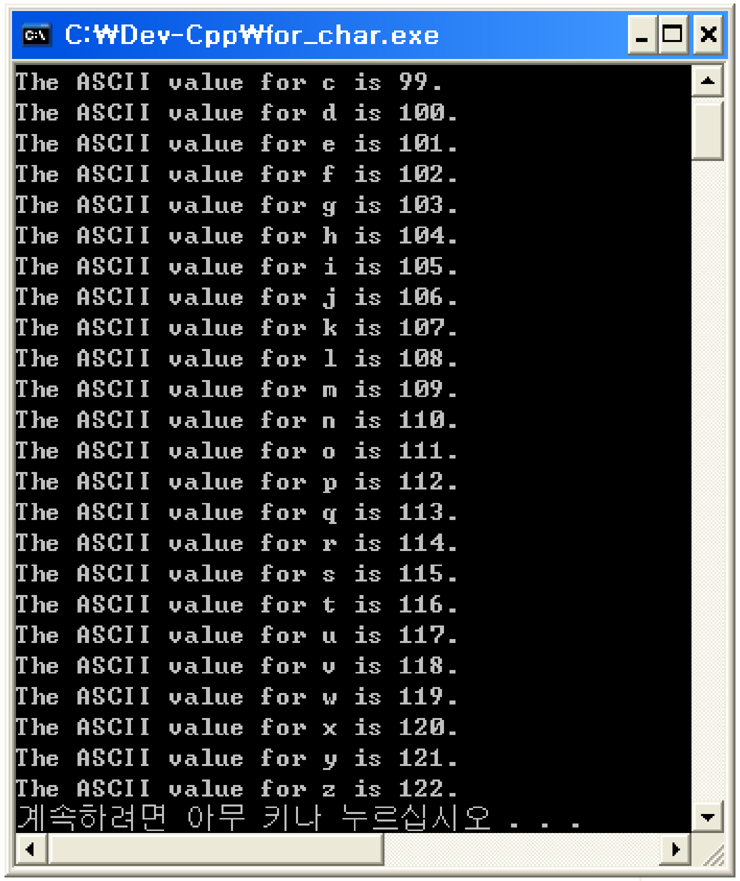
3) You can count by characters instead of by numbers:

```c
/* for_char.c */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char ch;

    for (ch = 'a'; ch <= 'z'; ch++)
        printf("The ASCII value for %c is %d.\n", ch, ch);

    system("pause");
    return 0;
}
```

The for Loop

Using `for` for Flexibility

- **Here's the abridged output:**



C:₩Dev-Cpp₩for_char.exe

```
The ASCII value for c is 99.
The ASCII value for d is 100.
The ASCII value for e is 101.
The ASCII value for f is 102.
The ASCII value for g is 103.
The ASCII value for h is 104.
The ASCII value for i is 105.
The ASCII value for j is 106.
The ASCII value for k is 107.
The ASCII value for l is 108.
The ASCII value for m is 109.
The ASCII value for n is 110.
The ASCII value for o is 111.
The ASCII value for p is 112.
The ASCII value for q is 113.
The ASCII value for r is 114.
The ASCII value for s is 115.
The ASCII value for t is 116.
The ASCII value for u is 117.
The ASCII value for v is 118.
The ASCII value for w is 119.
The ASCII value for x is 120.
The ASCII value for y is 121.
The ASCII value for z is 122.
계속하려면 아무 키나 누르십시오 . . .
```

The for Loop

Using `for` for Flexibility

4) You can test some condition other than the number of iterations.

In the for_cube program, you can replace

```
for (num = 1; num <= 6; num++)
```

with

```
for (num = 1; num*num*num <= 216; num++)
```

The for Loop

Using `for` for Flexibility

5) You can let a quantity increase geometrically instead of arithmetically.

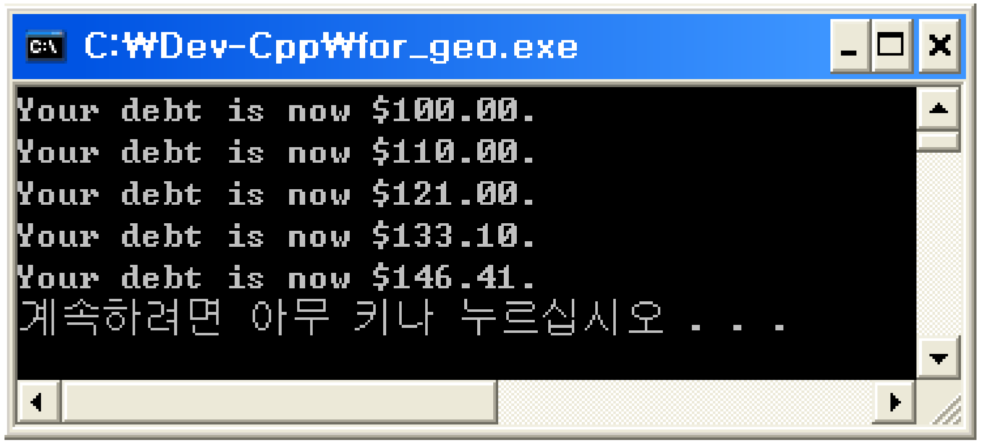
that is, instead of adding a fixed amount each time, you can multiply by a fixed amount:

```
/* for_geo.c */
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    double debt;

    for (debt = 100.0; debt < 150.0; debt = debt * 1.1)
        printf("Your debt is now $%.2f.\n", debt);

    system("pause");
    return 0;
}
```

The for Loop

Using `for` for Flexibility

- This program fragment multiplies debt by 1.1 for each cycle, increasing it by 10% each time.
- The output looks like this:



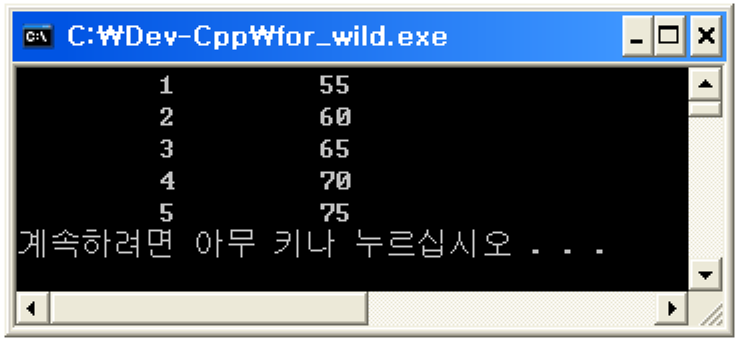
```
Your debt is now $100.00.
Your debt is now $110.00.
Your debt is now $121.00.
Your debt is now $133.10.
Your debt is now $146.41.
계속하려면 아무 키나 누르십시오 . . .
```

The for Loop

Using for for Flexibility

6) You can use any legal expression you want for the third expression. Whatever you put in will be updated for each iteration.

```c
/* for_wild.c */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int x;
    int y = 55;

    for (x = 1; y <= 75; y = (++x * 5) + 50)
        printf("%10d %10d\n", x, y);

    system("pause");
    return 0;
}
```



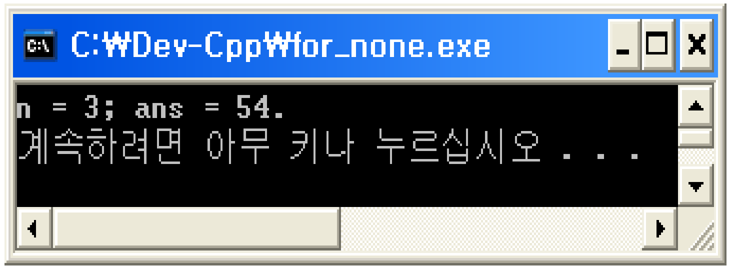
```
         1         55
         2         60
         3         65
         4         70
         5         75
계속하려면 아무 키나 누르십시오 . . .
```

The for Loop

Using `for` for Flexibility

7) You can even leave one or more expressions blank(but **don't omit the semicolons**).

```c
/* for_none.c */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int ans, n;
    ans = 2;

    for (n = 3; ans <= 25; )
        ans = ans * n;
    printf("n = %d; ans = %d.\n", n, ans);

    system("pause");
    return 0;
}
```



```
n = 3; ans = 54.
계속하려면 아무 키나 누르십시오 . . .
```

The for Loop

Using for for Flexibility

7) You can even leave one or more expressions blank(but **don't omit the semicolons**).

Incidentally, an empty middle control expression is considered to be true, so the following loop goes on forever:

```
for (; ; )
     printf("I want some action\n");
```

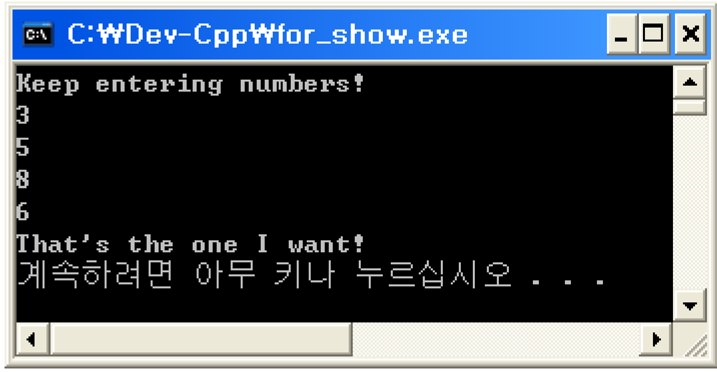
The for Loop

Using `for` for Flexibility

8) The first expression need not initialize a variable.

It could, instead, be a `printf()` statement of some sort.

```c
/* for_show.c */

#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int num = 0;

    for (printf("Keep entering numbers!\n"); num != 6;   )
        scanf("%d", &num);
    printf("That's the one I want!\n");

    system("pause");
    return 0;
}
```



C:₩Dev-Cpp₩for_show.exe

```
Keep entering numbers!
3
5
8
6
That's the one I want!
계속하려면 아무 키나 누르십시오 . . .
```

The for Loop

Using `for` for Flexibility

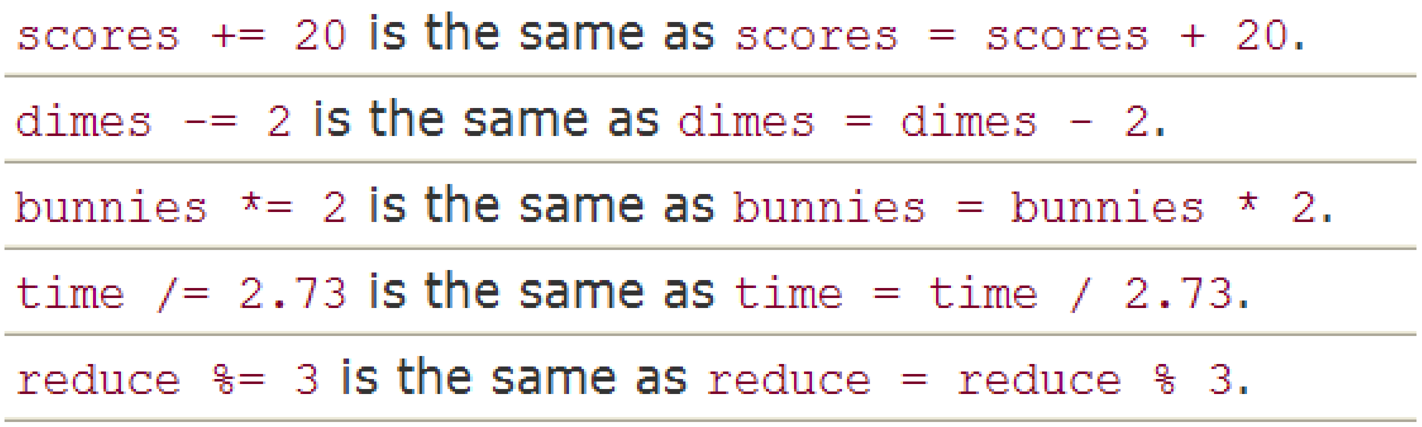
9) The parameters of the loop expressions can be altered by actions within the loop.

```
for (n = 1; n < 10000; n = n + delta)
```

- If after a few iterations your program decides that `delta` is too small or too large
- `delta` can be changed by the user as the loop runs.
- This sort of adjustment is a bit on the dangerous side.
- Ex) setting `delta` to `0` gets you (and the loop) nowhere.

More Assignment Operators

- **More Assignment Operators: +=, -=, *=, /=, %=**



`scores += 20` is the same as `scores = scores + 20`.
`dimes -= 2` is the same as `dimes = dimes - 2`.
`bunnies *= 2` is the same as `bunnies = bunnies * 2`.
`time /= 2.73` is the same as `time = time / 2.73`.
`reduce %= 3` is the same as `reduce = reduce % 3`.

```
x *= 3 * y + 12
```

is the same as

```
x = x * (3 * y + 12)
```

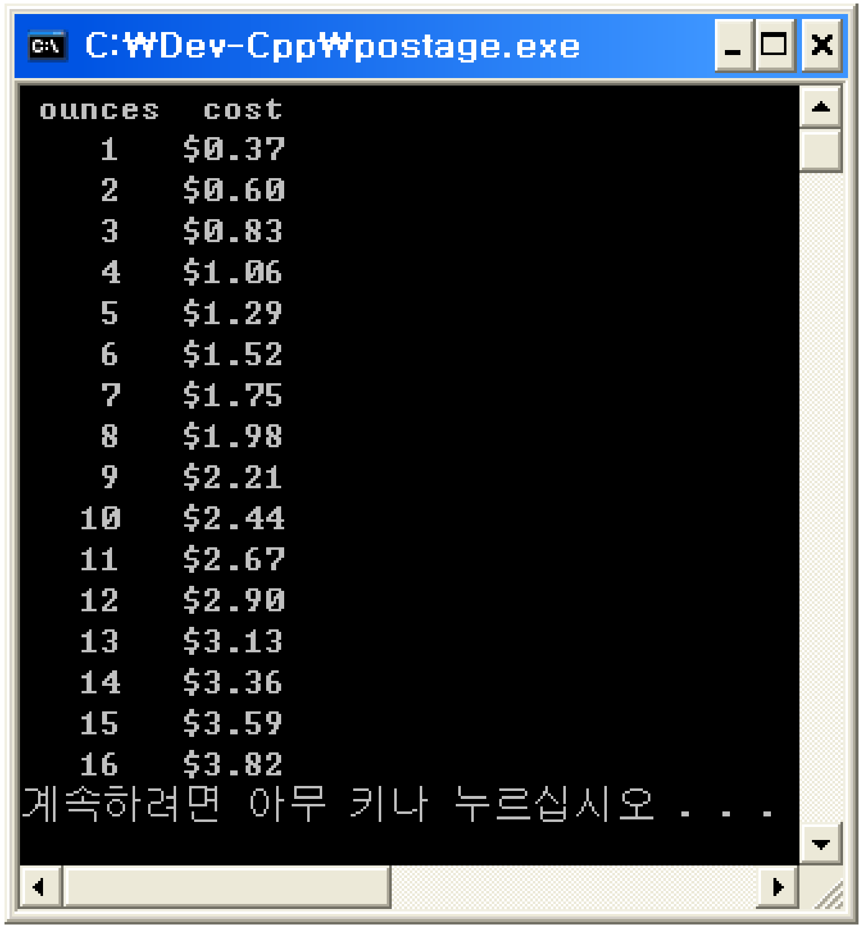
The Comma Operator

- **The postage.c Program**

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    const int FIRST_OZ = 37;
    const int NEXT_OZ = 23;
    int ounces, cost;

    printf(" ounces  cost\n");

    for (ounces=1, cost=FIRST_OZ; ounces <= 16; ounces++,
          cost += NEXT_OZ)
        printf("%5d   $%4.2f\n", ounces, cost/100.0);

    system("pause");
    return 0;
}
```

The Comma Operator

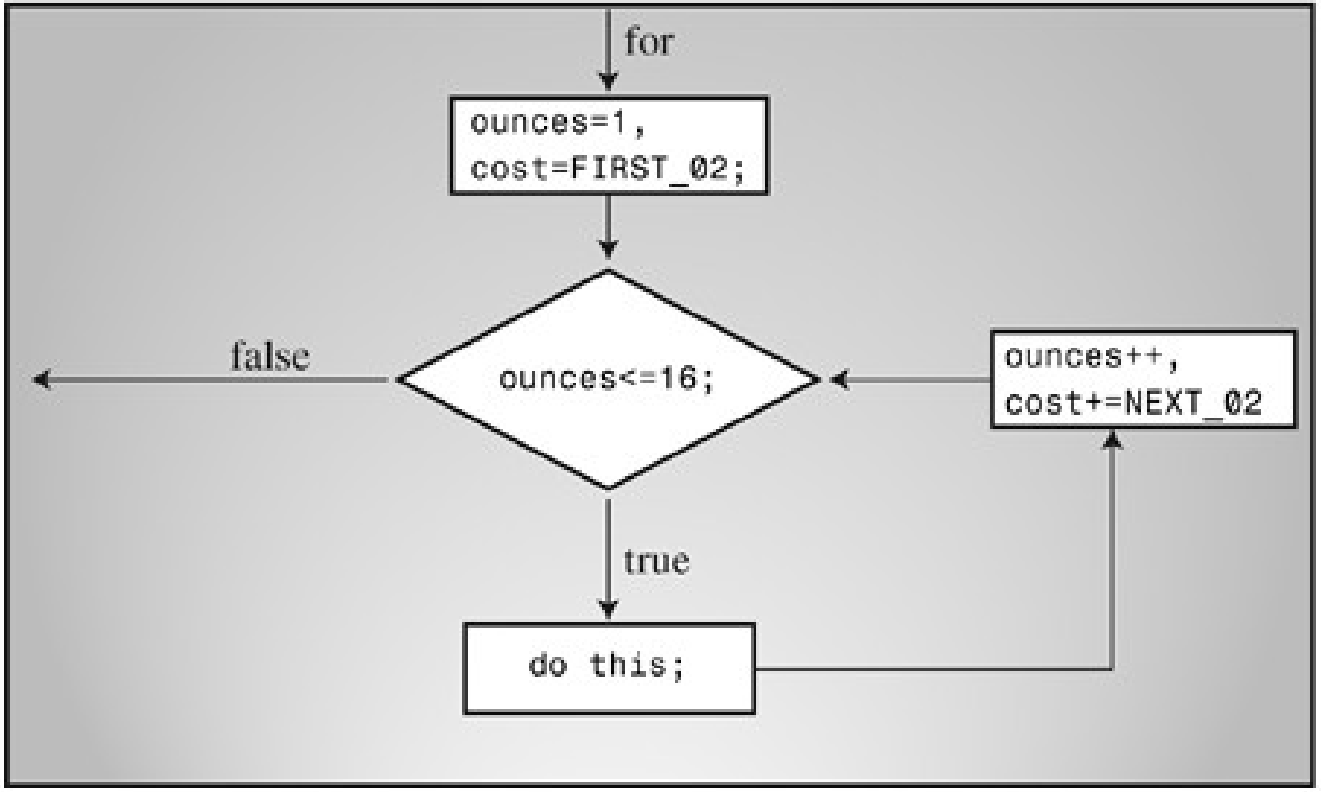
- **The postage.c Program**



```
 ounces   cost
     1    $0.37
     2    $0.60
     3    $0.83
     4    $1.06
     5    $1.29
     6    $1.52
     7    $1.75
     8    $1.98
     9    $2.21
    10    $2.44
    11    $2.67
    12    $2.90
    13    $3.13
    14    $3.36
    15    $3.59
    16    $3.82
계속하려면 아무 키나 누르십시오 . . .
```

The Comma Operator

- The comma operator and the for loop.



The Comma Operator

The comma operator and the for loop.

- The comma operator has **two further properties**.

1) it guarantees that the expressions it separates are evaluated in a left-to-right order.

```
ounces++, cost = ounces * FIRST_OZ
```

2) the value of the whole comma expression is the value of the right-hand member.

```
x = (y = 3, (z = ++y + 2) + 5);
```

The Comma Operator

- **The comma operator and the for loop.**
  - Why anyone would do this is beyond the scope of this book.
  - On the other hand, suppose you get careless and use comma notation in writing a number:

```
houseprice = 249,500;
```

```
houseprice = 249;
500;
```

```
houseprice = (249,500);
```

The Comma Operator

- **The comma operator and the for loop.**
  - The comma also is used as a separator.

```
char ch, date;
```

```
printf("%d %d\n", chimps, chumps);
```

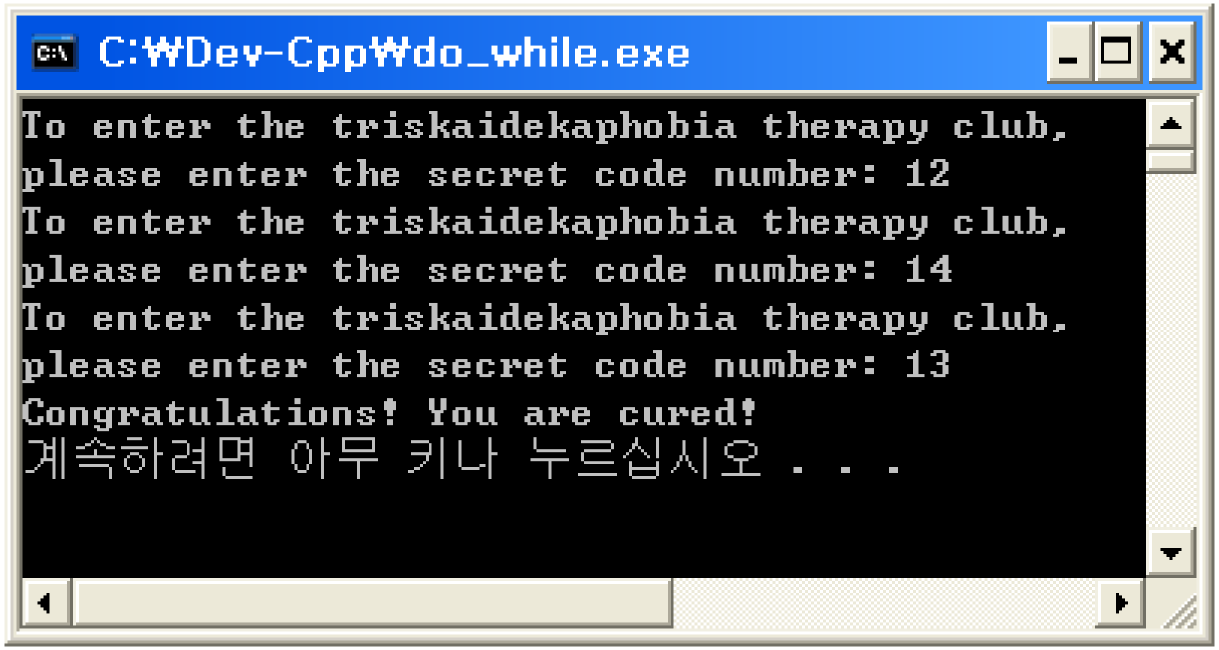
An Exit-Condition Loop: `do while`

- **The do_while.c Program**

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    const int secret_code = 13;
    int code_entered;

    do
    {
        printf("To enter the triskaidekaphobia therapy club,\n");
        printf("please enter the secret code number: ");
        scanf("%d", &code_entered);
    } while (code_entered != secret_code);

    printf("Congratulations! You are cured!\n");

    system("pause");
    return 0;
}
```

An Exit-Condition Loop: `do while`

- The do_while.c Program



C:₩Dev-Cpp₩do_while.exe

```
To enter the triskaidekaphobia therapy club,
please enter the secret code number: 12
To enter the triskaidekaphobia therapy club,
please enter the secret code number: 14
To enter the triskaidekaphobia therapy club,
please enter the secret code number: 13
Congratulations! You are cured!
계속하려면 아무 키나 누르십시오 . . .
```

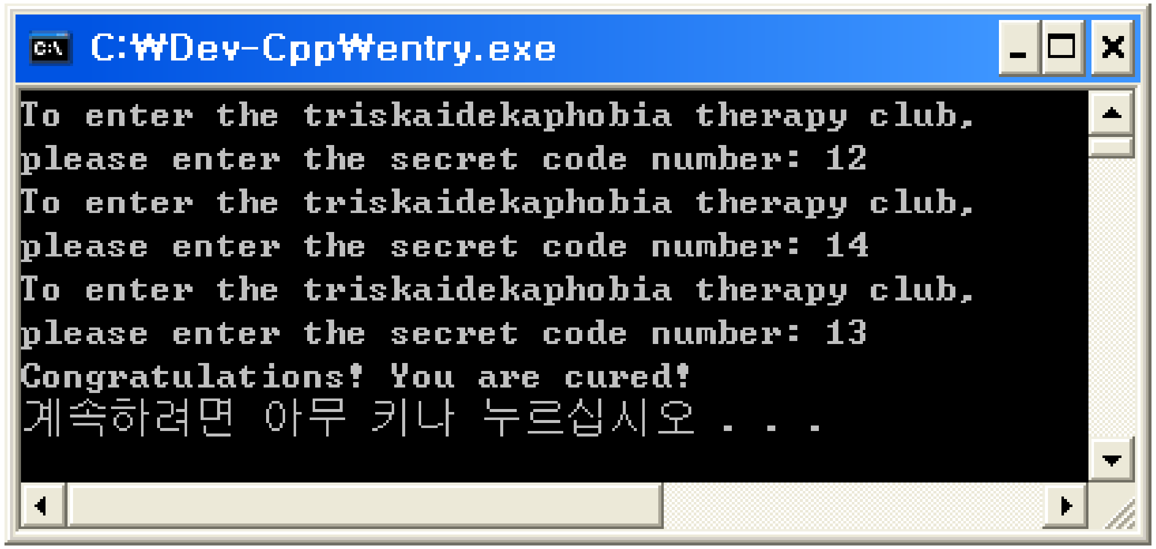
An Exit-Condition Loop: `do while`

- **The entry.c Program**

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    const int secret_code = 13;
    int code_entered;

    printf("To enter the triskaidekaphobia therapy club,\n");
    printf("please enter the secret code number: ");
    scanf("%d", &code_entered);

    while (code_entered != secret_code)
    {
        printf("To enter the triskaidekaphobia therapy club,\n");
        printf("please enter the secret code number: ");
        scanf("%d", &code_entered);
    }
    printf("Congratulations! You are cured!\n");

    system("pause");
    return 0;
}
```

An Exit-Condition Loop: `do while`

- The entry.c Program

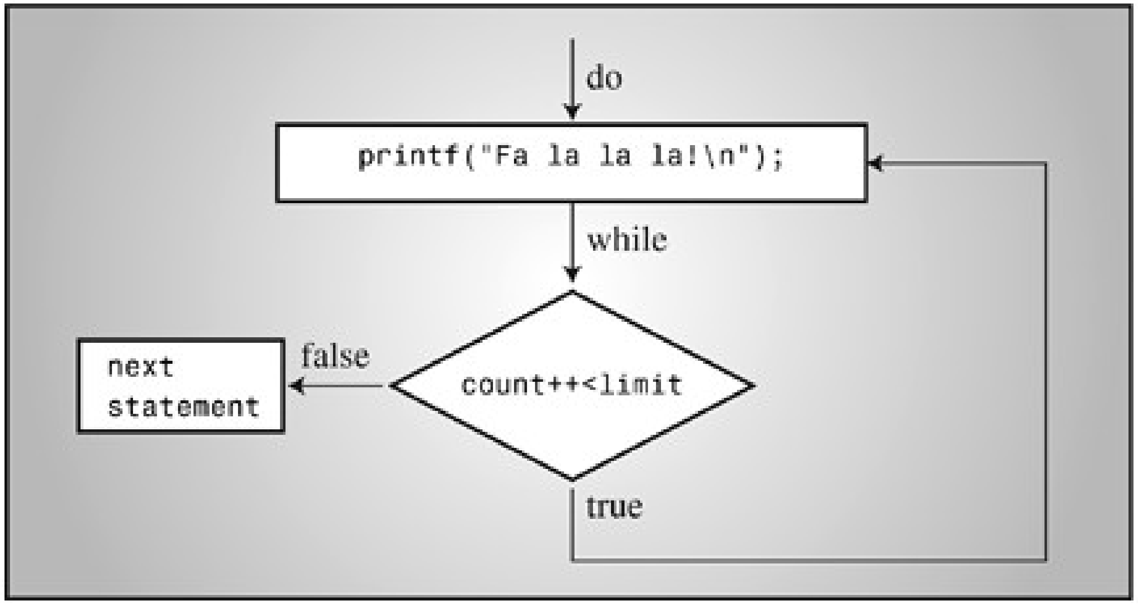


```
To enter the triskaidekaphobia therapy club,
please enter the secret code number: 12
To enter the triskaidekaphobia therapy club,
please enter the secret code number: 14
To enter the triskaidekaphobia therapy club,
please enter the secret code number: 13
Congratulations! You are cured!
계속하려면 아무 키나 누르십시오 . . .
```

An Exit-Condition Loop: `do while`

- **Structure of a do while loop.**
  - Here is the general form of the do while loop:

```
do
   statement
while ( expression );
```



An Exit-Condition Loop: `do while`

- **Structure of a do while loop.**
  - A **`do while`** loop
  - always executed **<u>at least once</u>** because the test is made after the body of the loop has been executed.
  - A **`for`** loop or a **`while`** loop
  - can be executed **<u>zero times</u>** because the test is made before execution.

An Exit-Condition Loop: `do while`

- Structure of a do while loop.
  - Ex) a password program

```
do
{
     prompt for password
     read user input
} while (input not equal to password);
```

An Exit-Condition Loop: `do while`

- **Structure of a do while loop.**
  - **Ex) a password program**
  - Avoid a do while structure of the type shown in the following pseudocode:

```
do
{
    ask user if he or she wants to continue
    some clever stuff
} while (answer is yes);
```

  - Here, after the user answers "no," some clever stuff gets done anyway because the test comes too late.

Which Loop?

Which Loop?

- When you decide you need a loop, which one should you use?
- To make a for loop like a while, you can omit the first and third expressions.
- `for ( ;test; )` is the same as `while (test)`

Which Loop?

Which Loop?

- To make a `while` like a `for`, preface it with an initialization and include update statements.

```
initialize;

while (test)
{
   body;
   update;
}
```

- Is the same as

```
for (initialize; test; update)
    body;
```

Which Loop?

Which Loop?

- A `while` loop is natural for the following condition:

```
while (scanf("%ld", &num) == 1)
```

- The `for` loop is a more natural choice `for` loops involving counting with an index:

```
for (count = 1; count <= 100; count++)
```

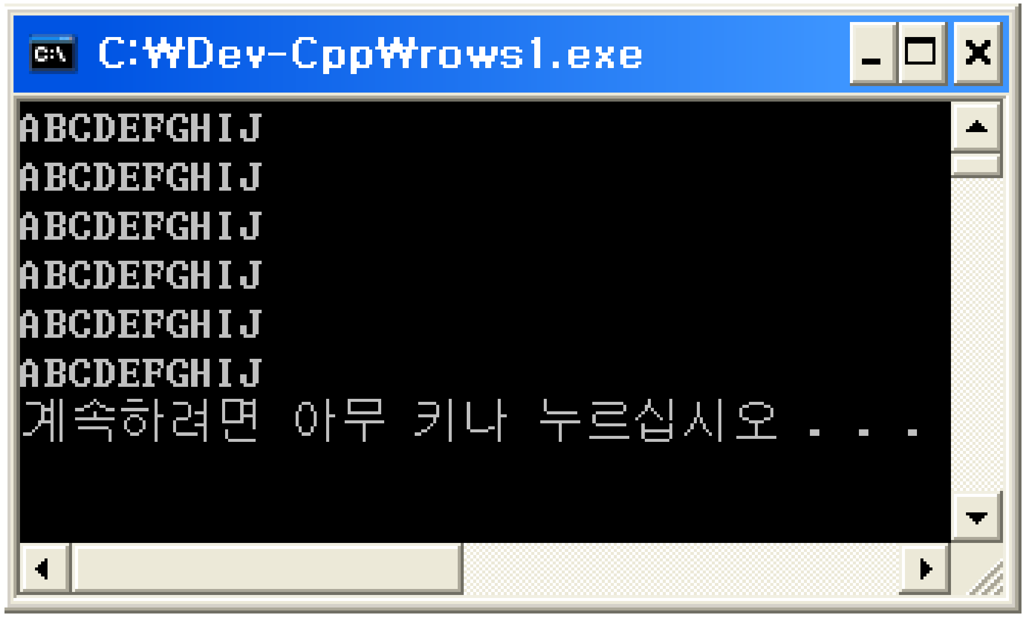
Nested Loops

The rows1.c Program

```
#include <stdio.h>
#include <stdlib.h>
#define ROWS 6
#define CHARS 10

int main(void)
{
    int row;
    char ch;

    for (row = 0; row < ROWS; row++)                /* line 10 */
    {
        for (ch = 'A'; ch < ('A' + CHARS); ch++)   /* line 12 */
            printf("%c", ch);
        printf("\n");
    }
    system("pause");
    return 0;
}
```

Nested Loops

The rows1.c Program



```
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
계속하려면 아무 키나 누르십시오 . . .
```

Nested Loops

The rows2.c Program

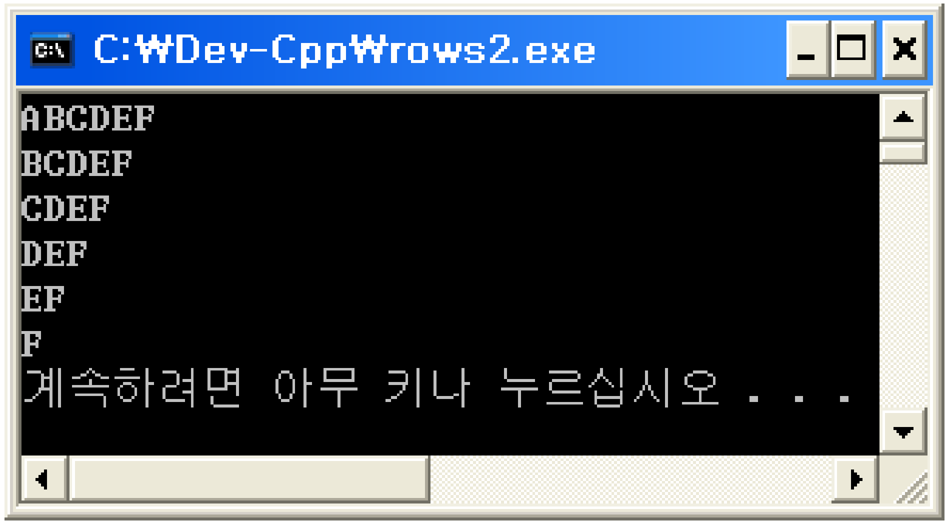
- **A Nested Variation**

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    const int ROWS = 6;
    const int CHARS = 6;
    int row;
    char ch;

    for (row = 0; row < ROWS; row++)
    {
        for (ch = ('A' + row);  ch < ('A' + CHARS); ch++)
            printf("%c", ch);
        printf("\n");
    }
    system("pause");
    return 0;
}
```

Nested Loops

The rows2.c Program

- **A Nested Variation**



C:₩Dev-Cpp₩rows2.exe

```
ABCDEF
BCDEF
CDEF
DEF
EF
F
계속하려면 아무 키나 누르십시오 . . .
```

Introducing Arrays

Arrays

- Array is a collection of same type elements under the same variable identifier referenced by index number.

```
float debts[20];
```

debts is an array with 20 elements, each of which can hold a type float value.

The first element of the array is called debts[0]

The second element is called debts[1], and so on, up to debts[19].

Introducing Arrays

Arrays

- Note that the numbering of array elements starts with 0, not 1.
- Each element can be assigned a float value.

```
debts[5] = 32.54;

debts[6] = 1.2e+21;
```

Introducing Arrays

Arrays

- You can read a value into a particular element.

```
scanf("%f", &debts[4]);// read a value into the 5th element
```

- Each of the following, for example, is **bad** code:

```
debts[20] = 88.32; // no such array element
```

```
debts[33] = 828.12;// no such array element
```

Introducing Arrays

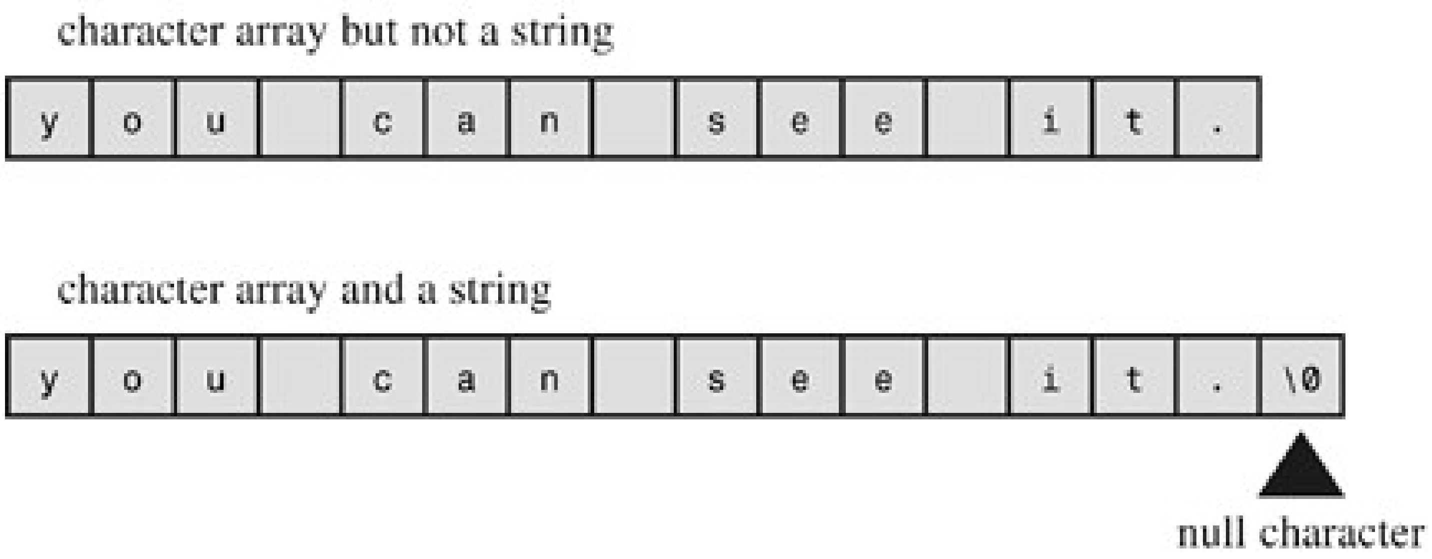
Arrays

- An array can be of any data type.

```
int nannies[22];   /* an array to hold 22 integers */

char actors[26];   /* an array to hold 26 characters */

long big[500];     /* an array to hold 500 long integers */
```

Introducing Arrays

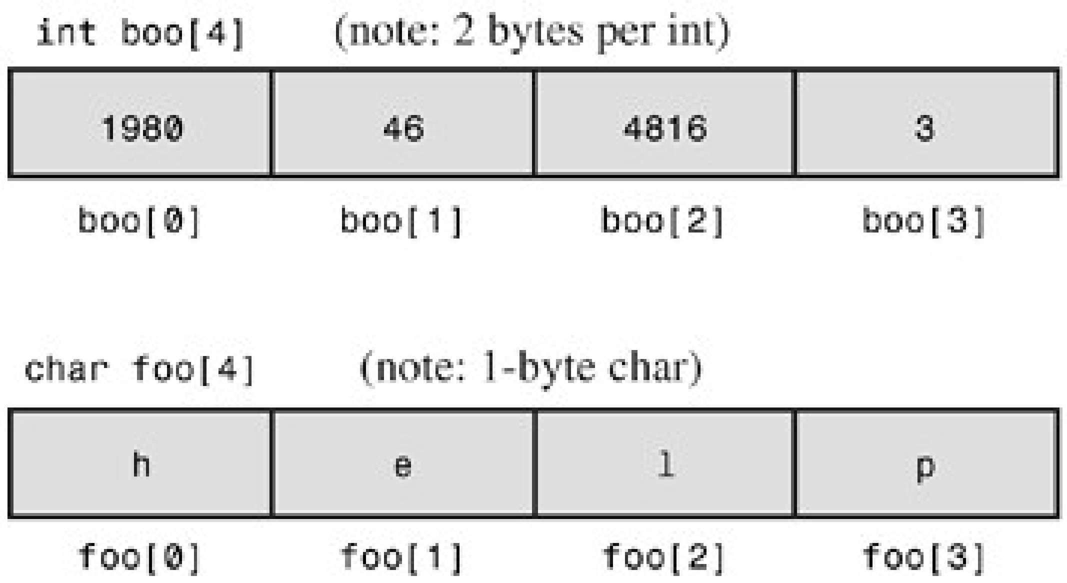
- **Character arrays and strings**



character array but not a string
y o u c a n s e e i t .
character array and a string
y o u c a n s e e i t . \0
null character

Introducing Arrays

- The `char` and `int` arrays in memory



Introducing Arrays

The scores_in.c Program(1/2)

- **Using a for Loop with an Array**

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10
#define PAR 72

int main(void)
{
    int index, score[SIZE];
    int sum = 0;
    float average;

    printf("Enter %d golf scores:\n", SIZE);
```

Introducing Arrays

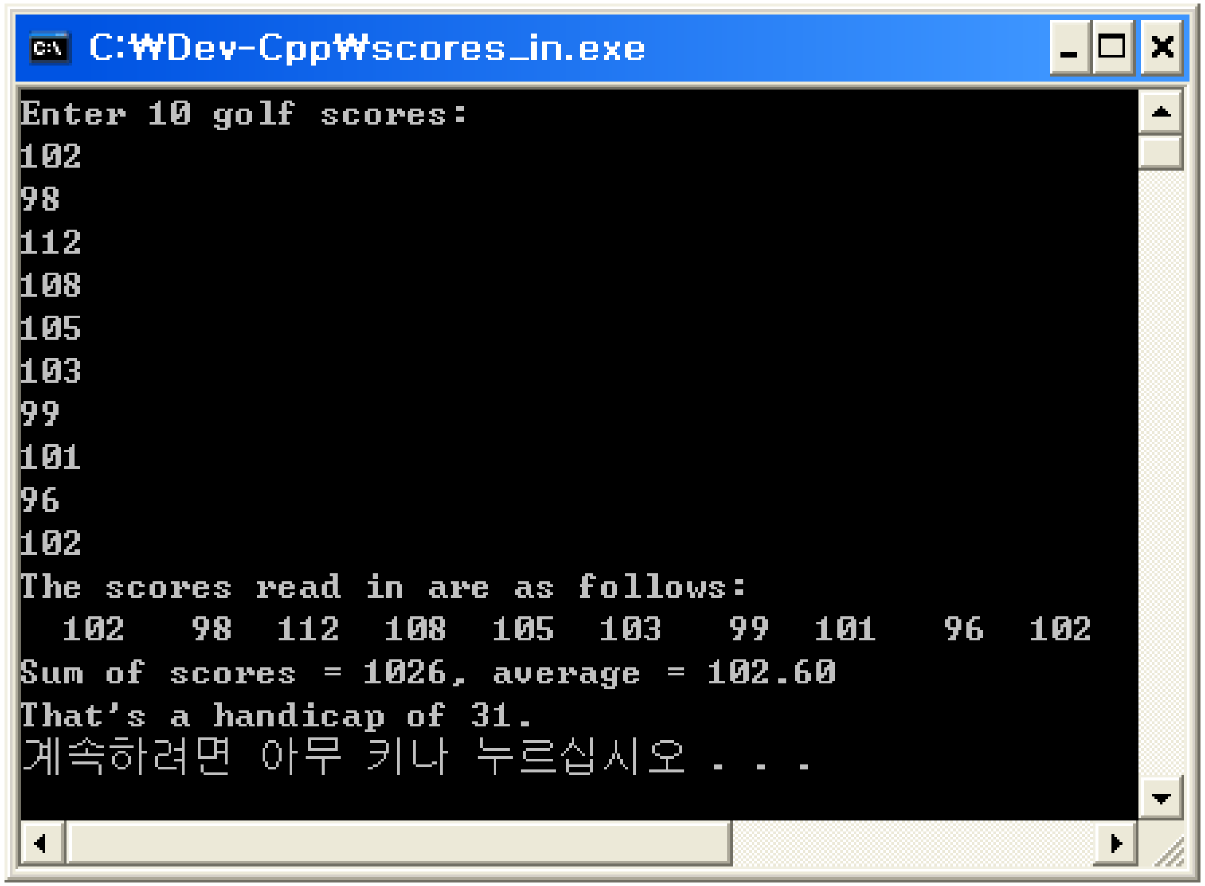
The scores_in.c Program(2/2)

- **Using a for Loop with an Array**

```
    for (index = 0; index < SIZE; index++)
        scanf("%d", &score[index]);   // read in the ten scores
    printf("The scores read in are as follows:\n");

    for (index = 0; index < SIZE; index++)
        printf("%5d", score[index]); // verify input
    printf("\n");

    for (index = 0; index < SIZE; index++)
        sum += score[index];          // add them up
    average = (float) sum / SIZE;     // time-honored method
    printf("Sum of scores = %d, average = %.2f\n", sum, average);
    printf("That's a handicap of %.0f.\n", average - PAR);

    system("pause");
    return 0;
}
```

Introducing Arrays

- The scores_in.c Program



```
Enter 10 golf scores:
102
98
112
108
105
103
99
101
96
102
The scores read in are as follows:
   102    98   112   108   105   103    99   101    96   102
Sum of scores = 1026, average = 102.60
That's a handicap of 31.
계속하려면 아무 키나 누르십시오 . . .
```

Introducing Arrays

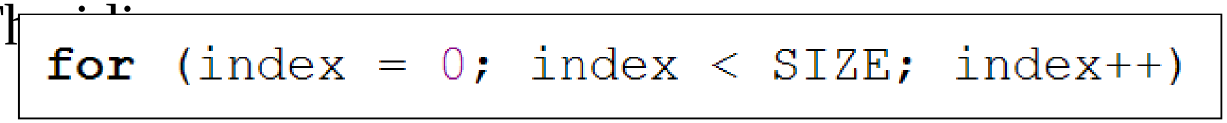
The scores_in.c Program

- This example illustrates several style points.

1) It's a good idea to use a `#define` directive to create a manifest constant

(`SIZE`) to specify the size of the array.

2) Th



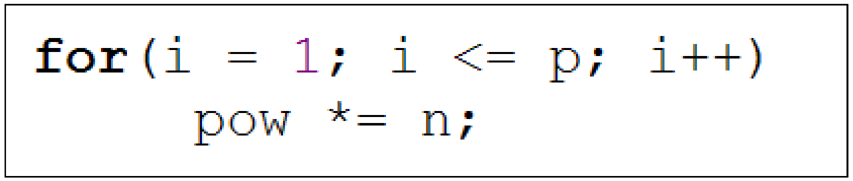
```
for (index = 0; index < SIZE; index++)
```

is a handy one for processing an array of size `SIZE`.

A Loop Example Using a Function Return Value

Using a Function Return Value

- Let's look at an algorithm



```
for(i = 1; i <= p; i++)
        pow *= n;
```

- **To write a function with a return value**, do the following:
- When you define a function, state the type of value it returns.
- Use the keyword return to indicate the value to be returned.

A Loop Example Using a Function Return Value

- **Using a Function Return Value**
  - For example

```
double power(double n, int p)   // returns a double
{
    double pow = 1;
    int i;

    for (i = 1; i <= p; i++)
        pow *= n;

    return pow;                 // return the value of pow
}
```

A Loop Example Using a Function Return Value

Using a Function Return Value

- Here you return the value of a variable, but you can return the **value of expressions**, too.
- For instance, the following is a valid statement:

```
return 2 * x + b;
```

A Loop Example Using a Function Return Value

- The power.c Program(1/2)

```
#include <stdio.h>
#include <stdlib.h>

double power(double n, int p); // ANSI prototype

int main(void)
{
    double x, xpow;
    int exp;

    printf("Enter a number and the positive integer power");
    printf(" to which\nthe number will be raised. Enter q");
    printf(" to quit.\n");
```

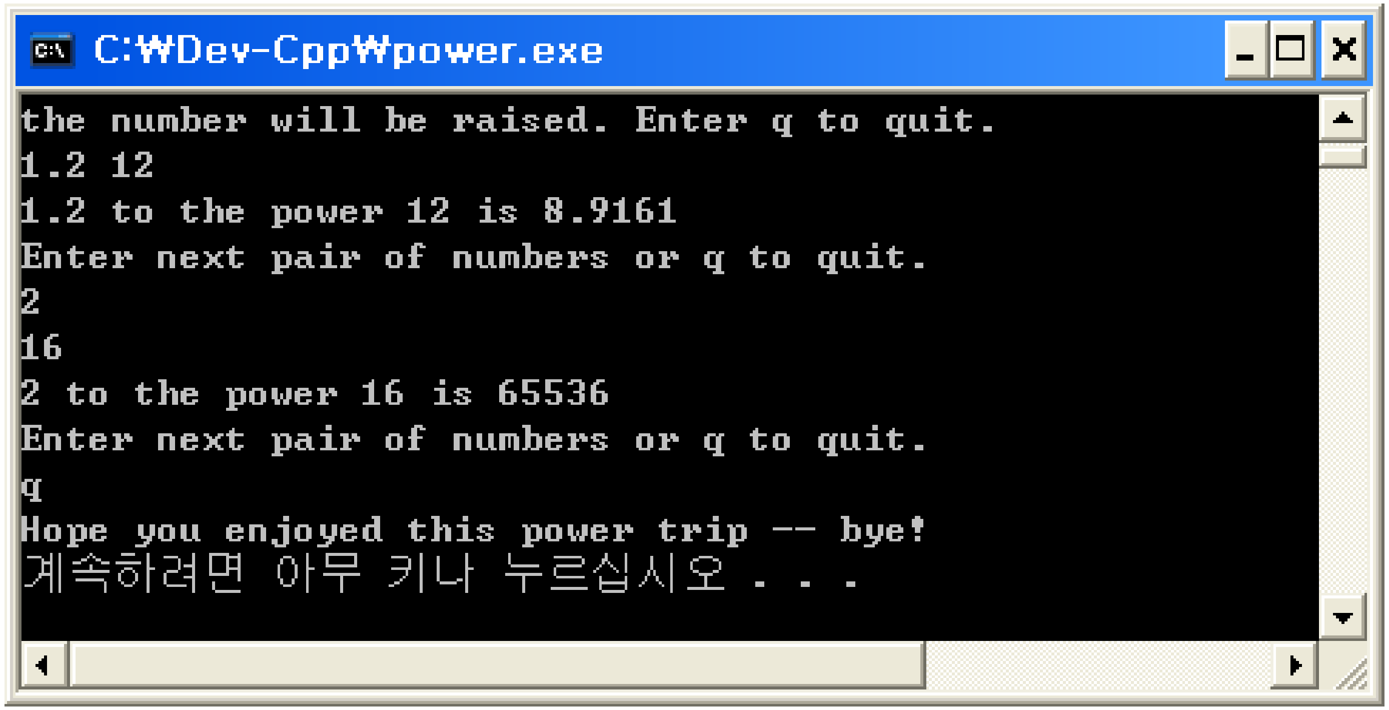
A Loop Example Using a Function Return Value

The power.c Program(2/2)

```
    while (scanf("%lf%d", &x, &exp) == 2)
    {
        xpow = power(x,exp);    // function call
        printf("%.3g to the power %d is %.5g\n", x, exp, xpow);
        printf("Enter next pair of numbers or q to quit.\n");
    }
    printf("Hope you enjoyed this power trip -- bye!\n");

    system("pause");
    return 0;
}

double power(double n, int p)   // function definition
{
    double pow = 1;
    int i;

    for (i = 1; i <= p; i++)
        pow *= n;
    return pow;                 // return the value of pow
}
```

A Loop Example Using a Function Return Value

- The power.c Program



```
the number will be raised. Enter q to quit.
1.2 12
1.2 to the power 12 is 8.9161
Enter next pair of numbers or q to quit.
2
16
2 to the power 16 is 65536
Enter next pair of numbers or q to quit.
q
Hope you enjoyed this power trip -- bye!
계속하려면 아무 키나 누르십시오 . . .
```

A Loop Example Using a Function Return Value

- **Using Functions with Return Values**
  - These are the basic elements in defining and using a function with a return value.
  - Declaring the function
  - Calling the function
  - Defining the function
  - Using the return keyword

A Loop Example Using a Function Return Value

- **Using Functions with Return Values**

Ex) if you are supposed to declare functions before you use their return values

Qustion1)

how come you used the return value of `scanf()`without declaring `scanf()`?

Question 2)

Why do you have to declare `power()`separately when your definition of it says it is type double?