“C Control Statements : Branching and jumps”

Using Bloodshed Dev-C++

Heejin Park

Hanyang University

Introduction

- The `if` Statement
- Adding `else` to the `if` Statement
- Let's Get Logical
- A Word-Count Program
- The Conditional Operator: ?:
- Loop Aids: `continue` and `break`
- Multiple Choice: `switch` and `break`
- The `goto` Statement

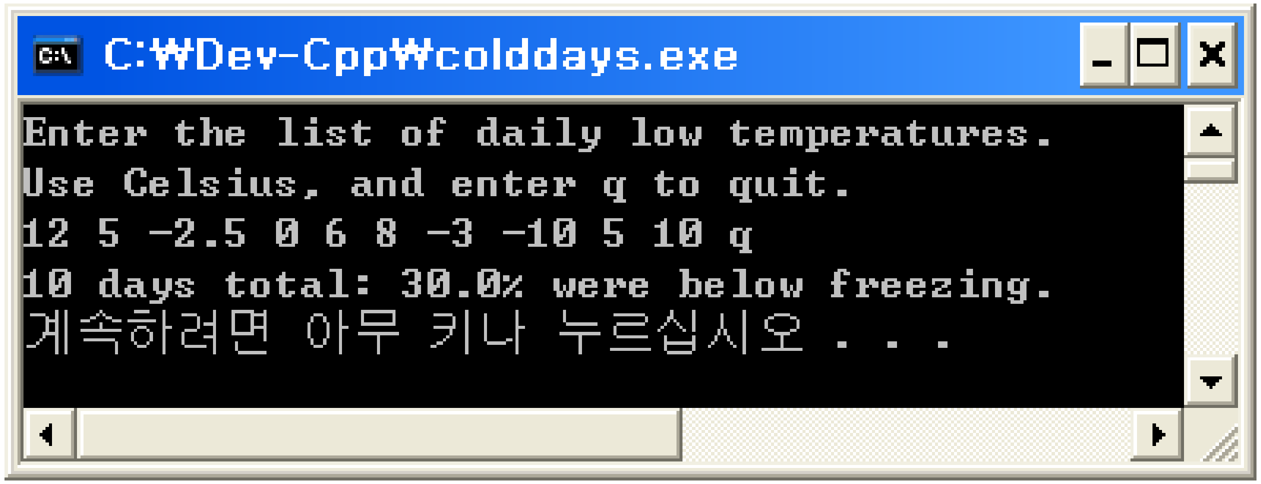
The `if` Statement

The colddays.c Program

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

int main(void)
{
    const int FREEZING = 0;
    float temperature;
    int cold_days = 0;
    int all_days = 0;

    printf("Enter the list of daily low temperatures.\n");
    printf("Use Celsius, and enter q to quit.\n");

    while (scanf("%f", &temperature) == 1)
    {
        all_days++;
        if (temperature < FREEZING)
            cold_days++;
    }
    if (all_days != 0)
        printf("%d days total: %.1f%% were below freezing.\n",
               all_days, 100.0 * (float) cold_days / all_days);
    if (all_days == 0)
        printf("No data entered!\n");

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

The `if` Statement

- The colddays.c Program



```
Enter the list of daily low temperatures.
Use Celsius, and enter q to quit.
12 5 -2.5 0 6 8 -3 -10 5 10 q
10 days total: 30.0% were below freezing.
계속하려면 아무 키나 누르십시오 . . .
```

The `if` Statement

The colddays.c Program

- Here is the new statement in the while block:
- This if statement instructs the computer to increase `cold_days` by 1 if the value just read (`temperature`) is less than zero.

```
if (temperature < FREEZING)
    cold_days++;
```

The `if` Statement

The colddays.c Program

- The if statement is called a ***branching statement*** or ***selection statement***.
- The general form is this:

```
if (expression)
     statement
```

The `if` Statement

The colddays.c Program

- The statement portion can be a simple statement, as in the example, or it can be a compound statement or block, marked off by braces:

```
if (score > big)
    printf("Jackpot!\n");   // simple statement
```

```
if (joe > ron)
{                                // compound statement
    joecash++;
    printf("You lose, Ron.\n");
}
```

Adding `else` to the `if` Statement

`if else`

- C also enables you to choose between two statements by using the `if else` form.

```
if (all_days != 0)
    printf("%d days total: %.1f%% were below freezing.\n",
           all_days, 100.0 * (float) cold_days / all_days);
if (all_days == 0)
    printf("No data entered!\n");
```

```
if (all_days!= 0)
    printf("%d days total: %.1f%% were below freezing.\n",
           all_days, 100.0 * (float) cold_days / all_days);
else
    printf("No data entered!\n");
```

Adding `else` to the `if` Statement

- **`if else`**
  - Note the general form of the `if else` statement:

```
if (expression)
     statement1
else
     statement2
```

Adding else to the if Statement

if else

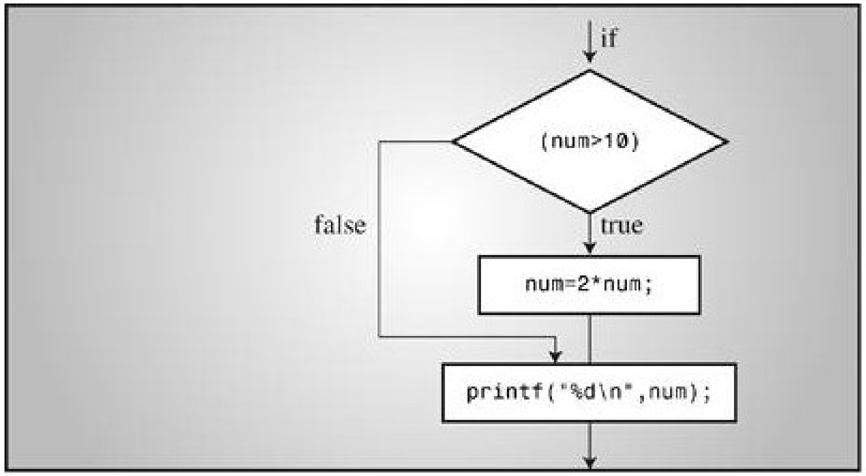
- If you want more than one statement between the if and the else, you must use braces to create a single block.

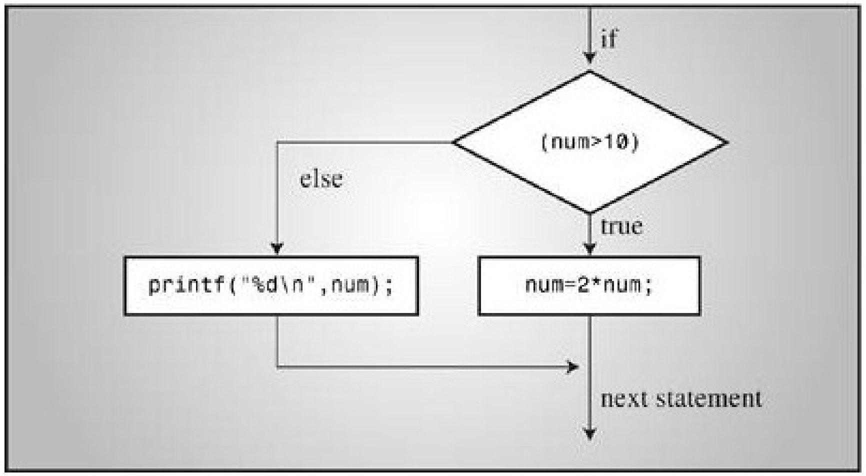
```
if (x > 0)
    printf("Incrementing x:\n");
    x++;

else                 // will generate an error
    printf("x <= 0 \n");
```

```
if (x > 0)
{
    printf("Incrementing x:\n");
    x++;
}
else
    printf("x <= 0 \n");
```

Adding `else` to the `if` Statement

- `if` versus `if else`





Adding `else` to the `if` Statement

Another Example: Introducing `getchar()` and `putchar()`

- The `getchar()` function
- takes no arguments, and it returns the next character from input.

```
ch = getchar();
```

- This statement has the same effect as the following statement:

```
scanf("%c", &ch);
```

Adding `else` to the `if` Statement

Another Example: Introducing `getchar()` and `putchar()`

- The `putchar()` function
- Prints its argument

```
putchar(ch);
```

- This statement has the same effect as the following statement:

```
printf("%c", ch);
```

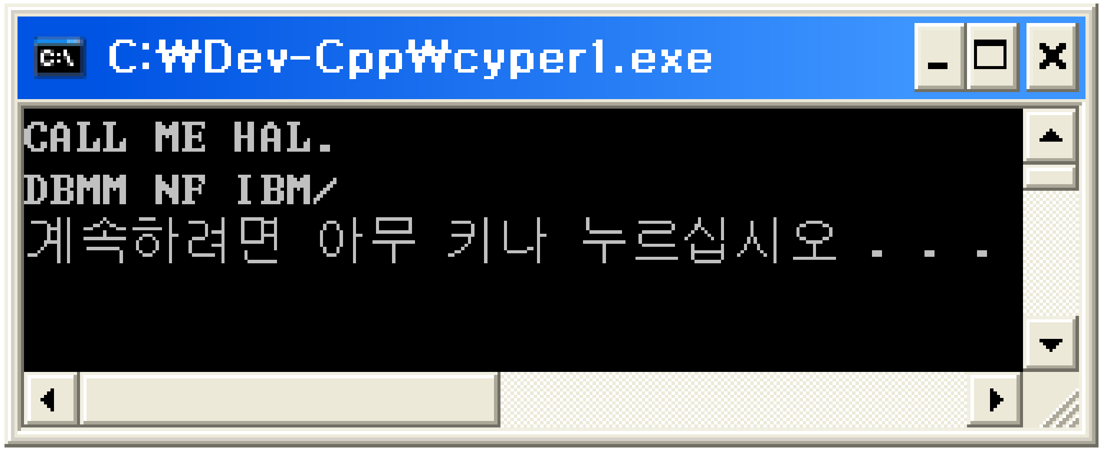
Adding `else` to the `if` Statement

- **The cypher1.c Program**

```c
#include <stdio.h>
#include <stdlib.h>
#define SPACE ' '                 /* that's quote-space-quote */

int main(void)
{
    char ch;

    ch = getchar();               /* read a character         */
    while (ch != '\n')            /* while not end of line    */
    {
        if (ch == SPACE)          /* leave the space          */
            putchar(ch);          /* character unchanged      */
        else
            putchar(ch + 1);      /* change other characters  */
        ch = getchar();           /* get next character       */
    }
    putchar(ch);                  /* print the newline        */

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

Adding `else` to the `if` Statement

- The cypher1.c Program



```
CALL ME HAL.
DBMM NF IBM/
계속하려면 아무 키나 누르십시오 . . .
```

Adding else to the if Statement

The cypher1.c Program

- You can replace a loop of the form

```
ch = getchar();                 /* read a character          */

while (ch != '\n')              /* while not end of line     */
{
    ...                         /* process character         */
    ch = getchar();             /* get next character        */
}
```

⬇

```
while ((ch = getchar()) != '\n')
{
    ...                            /* process character       */
}
```

Adding `else` to the `if` Statement

The cypher1.c Program

- The critical line

```
while ((ch = getchar()) != '\n')
```

- It demonstrates a characteristic C programming style—combining two actions in one expression.

```
while (
        (ch = getchar())              // assign a value to ch
                        != '\n')// compare ch to \n
```

Adding `else` to the `if` Statement

The cypher1.c Program

- Suppose that you mistakenly used this:

```
while (ch = getchar() != '\n')
```

- The statement illustrates once again that characters really are stored as integers.

```
putchar(ch + 1);    /* change other characters  */
```

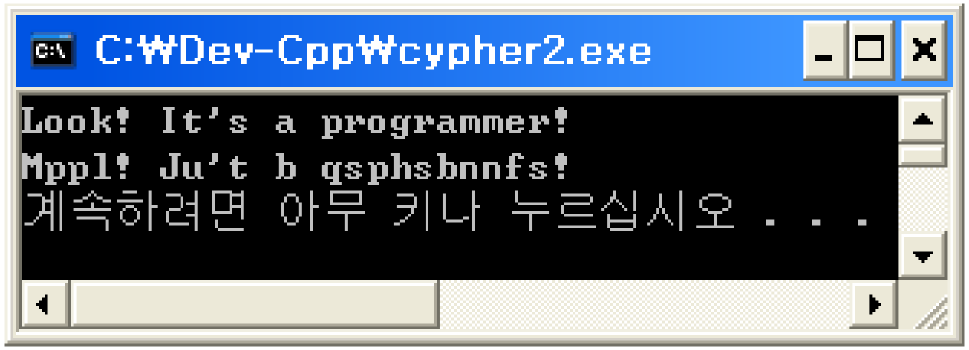
Adding else to the if Statement

- **The cypher2.c Program**
  - **The ctype.h Family of Character Functions**

```
#include <stdio.h>
#include <ctype.h>                 // for isalpha()

int main(void)
{
    char ch;

    while ((ch = getchar()) != '\n')
    {
        if (isalpha(ch))           // if a letter,
            putchar(ch + 1);       // change it
        else                       // otherwise,
            putchar(ch);           // print as is
    }
    putchar(ch);                   // print the newline
    return 0;
}
```

Adding `else` to the `if` Statement

- The cypher2.c Program



```
Look! It's a programmer!
Mppl! Ju't b qsphsbnnfs!
계속하려면 아무 키나 누르십시오 . . .
```

Adding `else` to the `if` Statement

The cypher2.c Program

- Note that the mapping functions don't modify the original argument.
- Instead, they return the modified value. That is,

```
tolower(ch);          // no effect on ch
```

Adding `else` to the `if` Statement

- The ctype.h Character-Testing Functions(1/2)

Name	True If the Argument Is
`isalnum()`	Alphanumeric (alphabetic or numeric)
`isalpha()`	Alphabetic
`isblank()`	A standard blank character (space, horizontal tab, or newline) or any additional locale-specific character so specified
`iscntrl()`	A control character, such as Ctrl+B
`isdigit()`	A digit
`isgraph()`	Any printing character other than a space
`islower()`	A lowercase character

Adding `else` to the `if` Statement

The ctype.h Character-Testing Functions(2/2)

Name	True If the Argument Is
`isprint()`	A printing character
`ispunct()`	A punctuation character (any printing character other than a space or an alphanumeric character)
`isspace()`	A whitespace character (a space, newline, formfeed, carriage return, vertical tab, horizontal tab, or, possibly, other locale-defined character)
`isupper()`	An uppercase character
`isxdigit()`	A hexadecimal-digit character

Adding `else` to the `if` Statement

The ctype.h Character-Mapping Functions

Name	Action
`tolower()`	If the argument is an uppercase character, this function returns the lowercase version; otherwise, it just returns the original argument.
`toupper()`	If the argument is a lowercase character, this function re-turns the uppercase version; otherwise, it just returns the original argument.

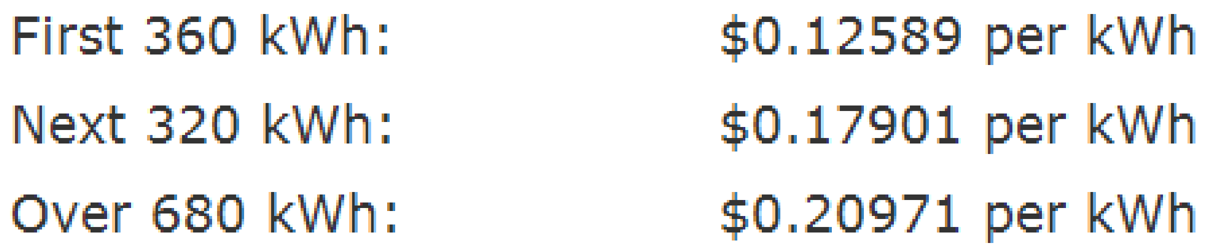
- Doesn't change `ch`. To change `ch`, do this:

```
ch = tolower(ch);  // convert ch to lowercase
```

Adding `else` to the `if` Statement

Multiple Choice `else if`

- Life often offers us more than two choices.
- Here are the rates one company charges for electricity, based on kilowatt-hours (kWh):



First 360 kWh:	$0.12589 per kWh
Next 320 kWh:	$0.17901 per kWh
Over 680 kWh:	$0.20971 per kWh

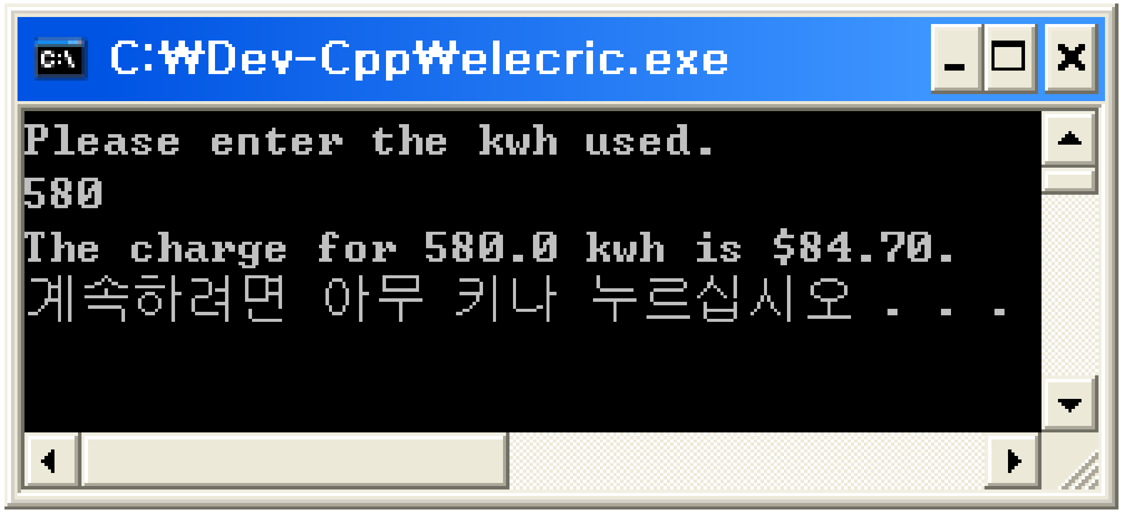
Adding `else` to the `if` Statement

- **The electric.c Program**

```
#include <stdio.h>
#define RATE1   0.12589       /* rate for first 360 kwh       */
#define RATE2   0.17901       /* rate for next 320 kwh        */
#define RATE3   0.20971       /* rate for over 680 kwh        */
#define BREAK1  360.0         /* first breakpoint for rates   */
#define BREAK2  680.0         /* second breakpoint for rates  */
#define BASE1   (RATE1 * BREAK1) /* cost for 360 kwh          */
#define BASE2  (BASE1 + (RATE2 * (BREAK2 - BREAK1)))
                              /* cost for 680 kwh             */

int main(void){
    double kwh;               /* kilowatt-hours used          */
    double bill;              /* charges                      */

    printf("Please enter the kwh used.\n");
    scanf("%lf", &kwh);       /* %lf for type double          */

    if (kwh <= BREAK1)
        bill = RATE1 * kwh;
    else if (kwh <= BREAK2)   /* kwh between 360 and 680      */
        bill = BASE1 + (RATE2 * (kwh - BREAK1));
    else                      /* kwh above 680                */
        bill = BASE2 + (RATE3 * (kwh - BREAK2));

    printf("The charge for %.1f kwh is $%1.2f.\n", kwh, bill);
    return 0;
}
```

Adding `else` to the `if` Statement

- The electric.c Program

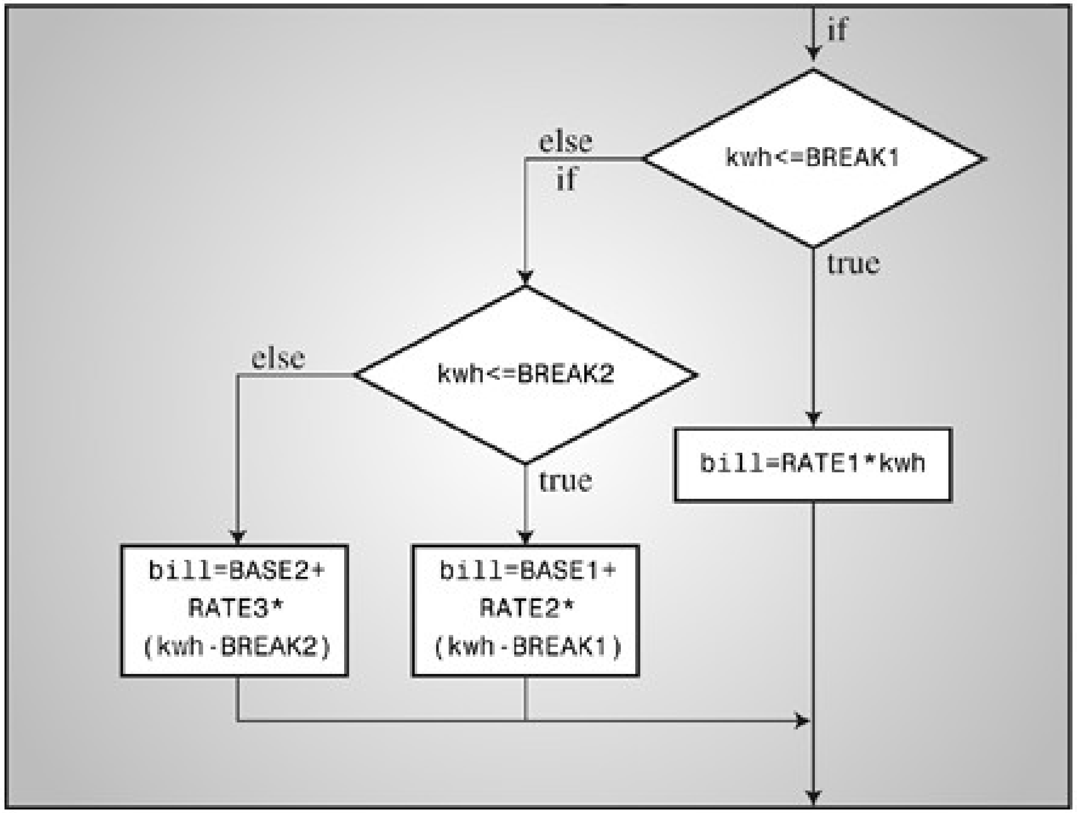


```
Please enter the kwh used.
580
The charge for 580.0 kwh is $84.70.
계속하려면 아무 키나 누르십시오 . . .
```

Adding `else` to the `if` Statement

The electric.c Program

- Program flow for electric.c


if
kwh<=BREAK1
else if
true
kwh<=BREAK2
else
true
bill=RATE1*kwh
bill=BASE2+ RATE3* (kwh-BREAK2)
bill=BASE1+ RATE2* (kwh-BREAK1)

Adding `else` to the `if` Statement

The electric.c Program

- Actually, the else if is a variation on what you already knew.
- Ex) the core of the program is just another way of writing.

```
if (kwh <=BREAK1)
     bill = RATE1 * kwh;
else
     if (kwh <=BREAK2)
          bill = BASE1 + RATE2 * (kwh - BREAK1);
     else
          bill = BASE2 + RATE3 * (kwh - BREAK2);
```

Adding `else` to the `if` Statement

The electric.c Program

- You can string together as many else if statements as you need (within compiler limits, of course), as illustrated by this fragment:

```
if (score < 1000)
        bonus = 0;
else if (score < 1500)
        bonus = 1;
else if (score < 2000)
        bonus = 2;
else if (score < 2500)
        bonus = 4;
else
        bonus = 6;
```

Adding `else` to the `if` Statement

Pairing `else` with `if`

- When you have a lot of ifs and elses, how does the computer decide which if goes with which else?

```
if (number > 6)
     if (number < 12)
         printf("You're close!\n");
else
     printf("Sorry, you lose a turn!\n");
```

Adding `else` to the `if` Statement

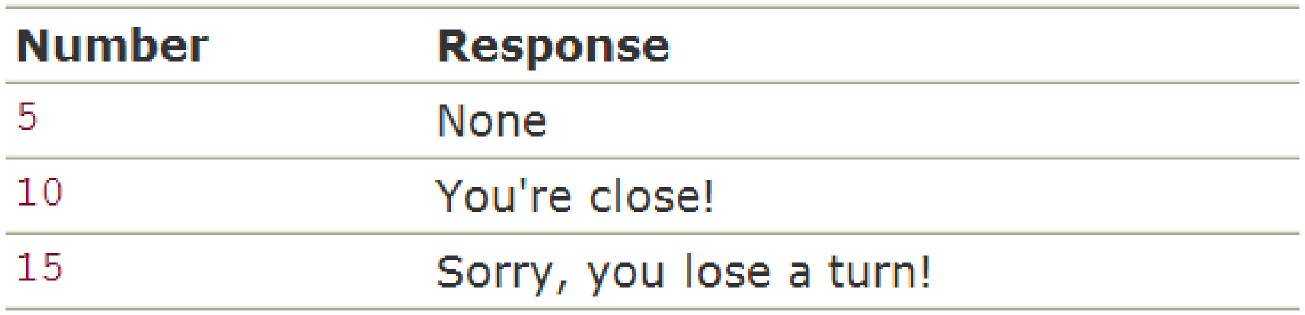
Pairing `else` with `if`

- When is **<u>`Sorry,you lose a turn!`</u>** printed?
- When `number` is **less than or equal to `6`**?
- When `number` is **greater than `12`**?
- In other words, does the **`else`** go with the first **`if`** or the second?

Adding `else` to the `if` Statement

Pairing `else` with `if`

- That is, you would get these responses:

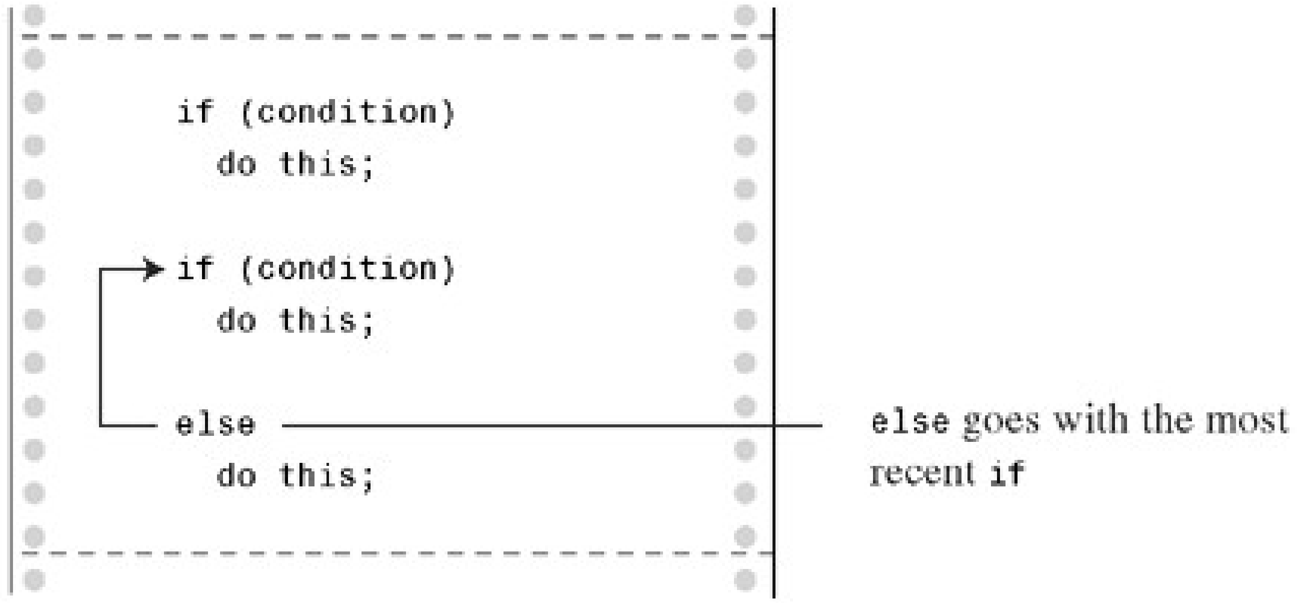


Number	Response
5	None
10	You're close!
15	Sorry, you lose a turn!

Adding `else` to the `if` Statement

Pairing `else` with `if`

- The rule for `if else` pairings.

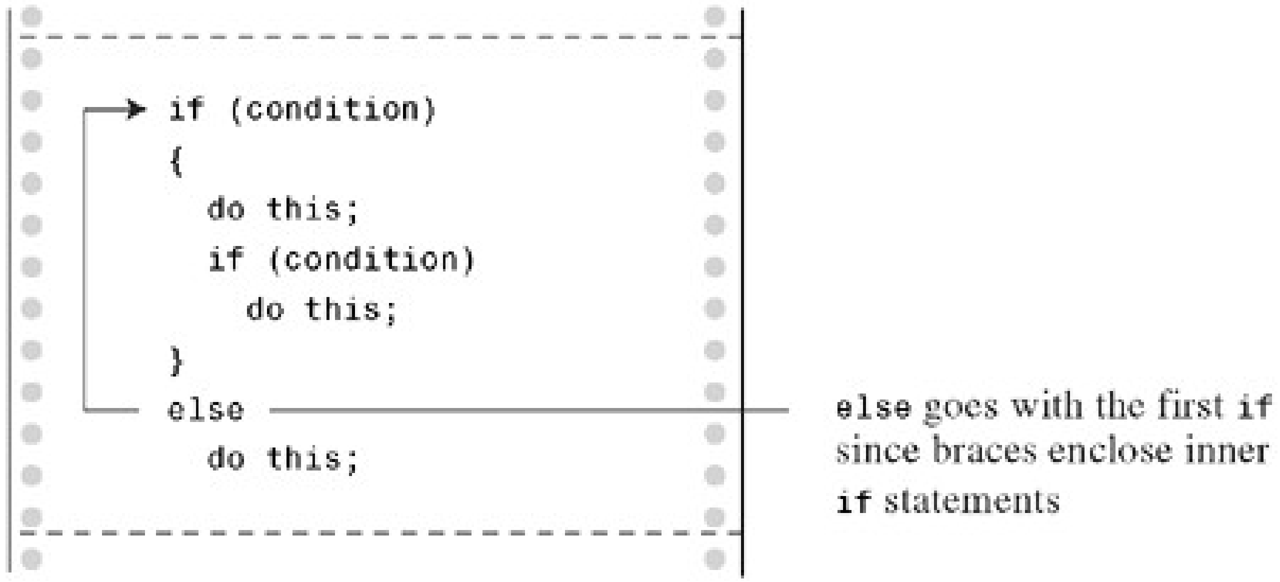


```
if (condition)
  do this;

if (condition)
  do this;

else
  do this;
```

Adding `else` to the `if` Statement

Pairing `else` with `if`

- The rule for `if else` pairings.



```
if (condition)
{
  do this;
  if (condition)
    do this;
}
else
  do this;
```

else goes with the first if since braces enclose inner if statements

Adding else to the if Statement

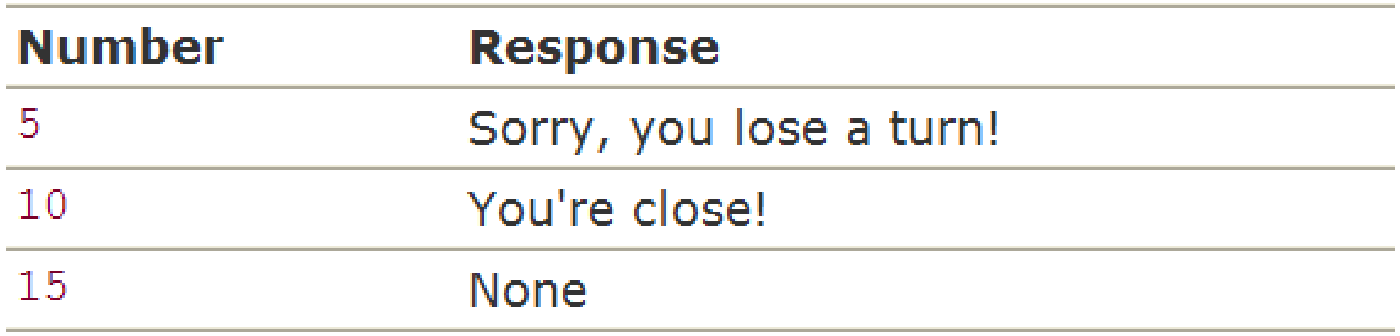
Pairing else with if

- If you really want the else to go with the first if,
- you could write the fragment this way:

```
if (number > 6)
{
        if (number < 12)
              printf("You're close!\n");
}
else
     printf("Sorry, you lose a turn!\n");
```

Adding `else` to the `if` Statement

- **Pairing `else` with `if`**
  - Now you would get these responses:



Number	Response
5	Sorry, you lose a turn!
10	You're close!
15	None

Adding `else` to the `if` Statement

More Nested `if`s

- **`nested if`**
- used when choosing a particular selection leads to an additional choice.

```
prompt user
while the scanf() return value is 1
    analyze the number and report results
    prompt user
```

- Recall that by using `scanf()` in the loop test condition.

Adding `else` to the `if` Statement

More Nested `if`s

- **`nested if`**
- Next, you need a plan for finding divisors.
- Perhaps the most obvious approach is something like this:

```
for (div = 2; div < num; div++)
  if (num % div == 0)
       printf("%d is divisible by %d\n", num, div);
```

Adding else to the if Statement

More Nested ifs

- **nested if**
- You can express the test condition as follows:

```
for (div = 2; (div * div) <= num; div++)
    if (num % div == 0)
        printf("%d is divisible by %d and %d.\n",
                num, div, num / div);
```

  – If num is 144, the loop runs through div = 12.
  – If num is 145, the loop runs through div = 13.

Adding `else` to the `if` Statement

More Nested `if`s

- **`nested if`**
- We need to address just **two more problems**, and then you'll be ready to program.

```
for (div = 2; (div * div) <= num; div++)
{
    if (num % div == 0)
    {
        if (div * div != num)
            printf("%d is divisible by %d and %d.\n",
                    num, div, num / div);
        else
            printf("%d is divisible by %d.\n", num, div);
    }
}
```

Adding else to the if Statement

- The divisors.c Program(1/2)

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

int main(void)
{
    unsigned long num;                  // number to be checked
    unsigned long div;                  // potential divisors
    bool isPrime;                       // prime flag

    printf("Please enter an integer for analysis; ");
    printf("Enter q to quit.\n");
```

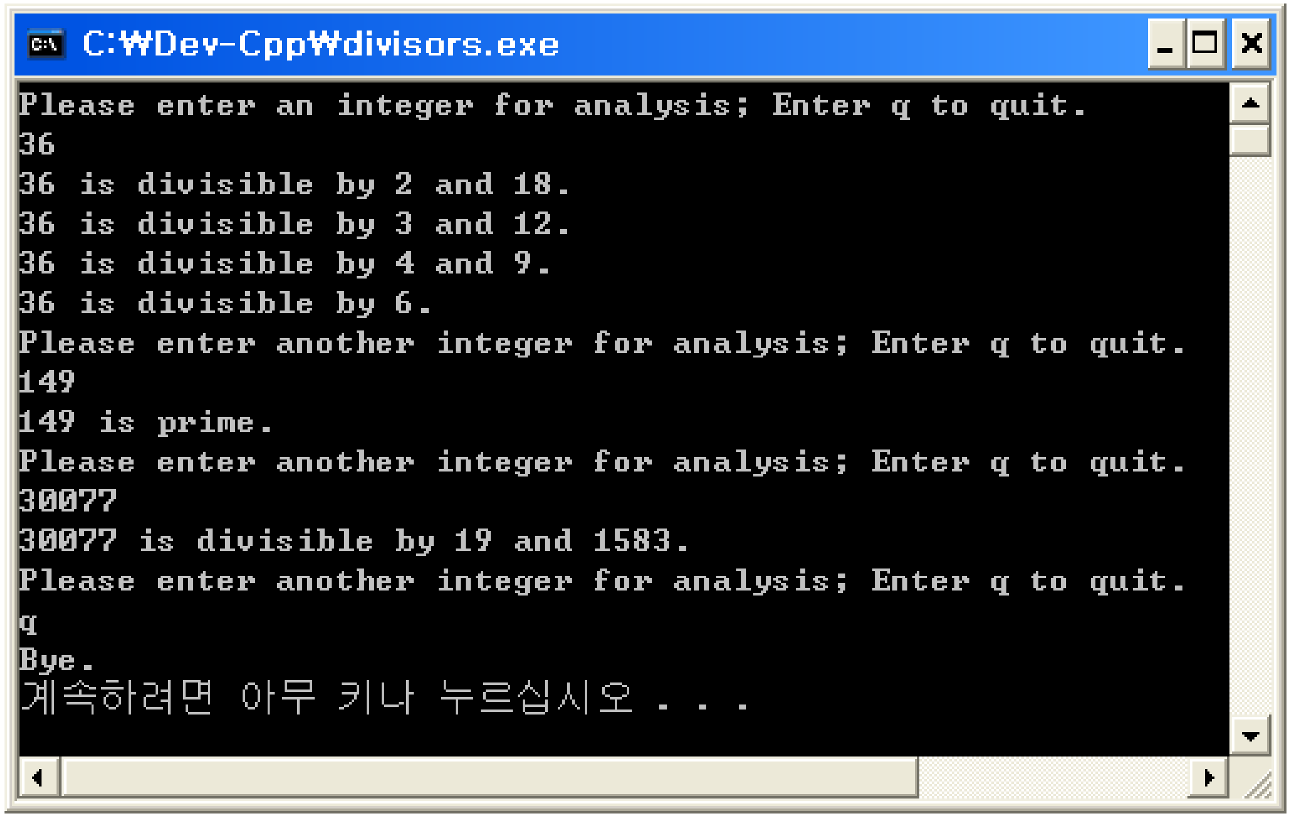
Adding else to the if Statement

- The divisors.c Program(2/2)

```
    while (scanf("%lu", &num) == 1)
    {
        for (div = 2, isPrime= true; (div * div) <= num; div++)
        {
            if (num % div == 0)
            {
                if ((div * div) != num)
                printf("%lu is divisible by %lu and %lu.\n",
                        num, div, num / div);
                else
                    printf("%lu is divisible by %lu.\n",
                           num, div);
                isPrime= false; // number is not prime
            }
        }
        if (isPrime)
            printf("%lu is prime.\n", num);
        printf("Please enter another integer for analysis; ");
        printf("Enter q to quit.\n");
    }
    printf("Bye.\n");
    return 0;
}
```

Adding else to the if Statement

- The divisors.c Program



C:₩Dev-Cpp₩divisors.exe

```
Please enter an integer for analysis; Enter q to quit.
36
36 is divisible by 2 and 18.
36 is divisible by 3 and 12.
36 is divisible by 4 and 9.
36 is divisible by 6.
Please enter another integer for analysis; Enter q to quit.
149
149 is prime.
Please enter another integer for analysis; Enter q to quit.
30077
30077 is divisible by 19 and 1583.
Please enter another integer for analysis; Enter q to quit.
q
Bye.
계속하려면 아무 키나 누르십시오 . . .
```

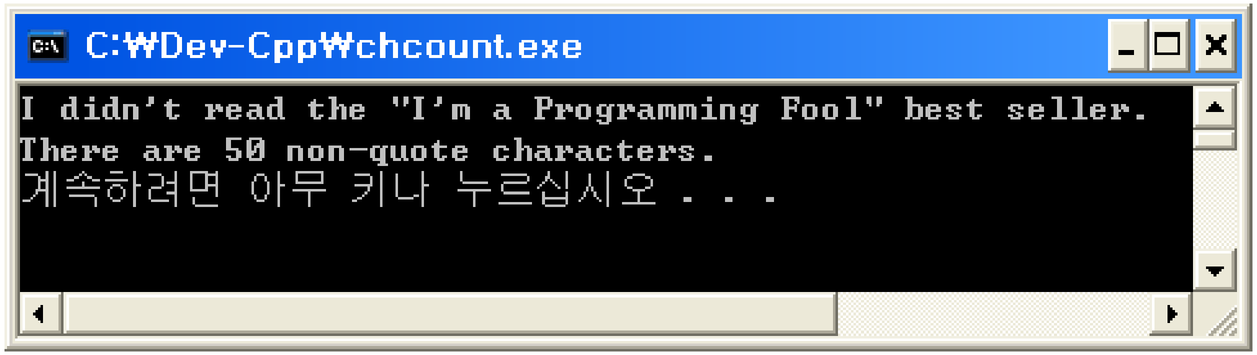
Let's Get Logical

The chcount.c Program

```
#include <stdio.h>
#include <stdlib.h>
#define PERIOD '.'

int main(void)
{
    int ch;
    int charcount = 0;
    while ((ch = getchar()) != PERIOD)
    {
        if (ch != '"' && ch != '\'')
            charcount++;
    }

    printf("There are %d non-quote characters.\n", charcount);

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

Let's Get Logical

- The chcount.c Program



C:₩Dev-Cpp₩chcount.exe

```
I didn't read the "I'm a Programming Fool" best seller.
There are 50 non-quote characters.
계속하려면 아무 키나 누르십시오 . . .
```

Let's Get Logical

Logical operators

- C has three logical operators.

Operator	Meaning
&&	and
||	or
!	not

Let's Get Logical

Logical operators

- Suppose `exp1` and `exp2` are two simple relational expressions, such as `cat > rat` and `debt == 1000`.
- Then you can state the following:
  - `exp1 && exp2` is true only if both `exp1` and `exp2` are true.
  - `exp1 || exp2` is true if either `exp1` or `exp2` is true or if both are true.
  - `!exp1` is true if `exp1` is false, and it's false if `exp1` is true.

Let's Get Logical

- Logical operators
  - `5 > 2 && 4 > 7`
  - false because only one subexpression is true.
  - `5 > 2 || 4 > 7`
  - true because at least one of the subexpressions is true.
  - `!(4 > 7)`
  - true because `4` is not greater than `7`.
  - equivalent to the following: `4 <= 7`

Let's Get Logical

Logical operators

- If you are unfamiliar or uncomfortable with logical operators, remember that,

```
(practice && time) == perfection
```

Let's Get Logical

Alternate Spellings: The iso646.h Header File

- **`iso646.h` header file**
- The C99 standard has added alternative spellings for the logical operators.
- If you use this header file, you can use **`and`** instead of **`&&`**, or instead of **`||`**, and **`not`** instead of **`!`**.

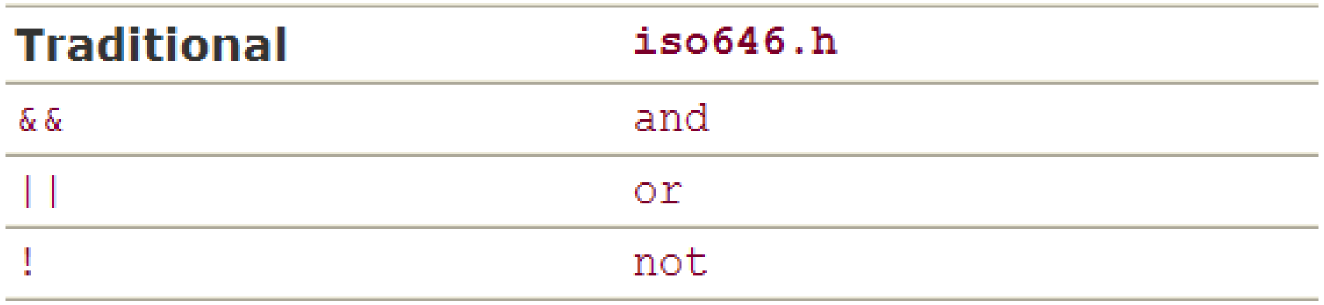
```
if (ch != '"' && ch != '\'')
    charcount++;
```

↓

```
if (ch != '"' and ch != '\'')
    charcount++;
```

Let's Get Logical

- **Alternate Spellings: The iso646.h Header File**
  - **Alternative Representations of Logical Operators**



Traditional	iso646.h
`&&`	`and`
`||`	`or`
`!`	`not`

Let's Get Logical

Precedence

- The ! operator has a very high precedence
- higher than multiplication
- same as the increment operators
- just below that of parentheses

- The `&&` operator has higher precedence than `||`
- both rank below the relational operators and above assignment in precedence.

Let's Get Logical

Precedence

- Ex)

```
a > b && b > c || b > d
```

- would be interpreted as

```
((a > b) && (b > c)) || (b > d)
```

That is, **`b`** is between **`a`** and **`c`**, or **`b`** is greater than **`d`**.

Let's Get Logical

Order of Evaluation

- C ordinarily does not guarantee which parts of a complex expression are evaluated first.

```
apples = (5 + 3) * (9 + 6);
```

- The expression `5 + 3` might be evaluated before `9 + 6`, or it might be evaluated afterward.

Let's Get Logical

Order of Evaluation

- The `&&` and `||` operators are sequence points
- so all side effects take place before a program moves from one operand to the next.

```
while ((c = getchar()) != ' ' && c != '\n')
```

- This construction sets up a loop that reads characters up to the first space or newline character.
- The first subexpression gives a value to `c`, which then is used in the second subexpression.

Let's Get Logical

Order of Evaluation

- Here is another example:

```
if (number != 0 && 12/number == 2)
    printf("The number is 5 or 6.\n");
```

- Consider this example:

```
while ( x++ < 10 && x + y < 20)
```

- The fact that the `&&` operator is a sequence point guarantees that `x` is incremented before the expression on the right is evaluated.

Let's Get Logical

Ranges

- You can use the `&&` operator to test for ranges.

```
if (range >= 90 && range <= 100)
    printf("Good show!\n");
```

- It's important to avoid imitating common mathematical notation.

```
if (90 <= range <= 100)     // NO! Don't do it!
    printf("Good show!\n");
```

Let's Get Logical

Ranges

- The order of evaluation for the <= operator is left-to-right.
- the test expression is interpreted as follows:

```
(90 <= range) <= 100
```

Let's Get Logical

Ranges

- A lot of code uses range tests to see whether a character is, say, a lowercase letter.
- For instance, suppose `ch` is a `char` variable:

```
if (ch >= 'a' && ch <= 'z')
    printf("That's a lowercase character.\n");
```

Let's Get Logical

Ranges

- The more portable way

```
if (islower(ch))
    printf("That's a lowercase character.\n");
```

- The **`islower()`** function from the **`ctype.h`** family
  - works regardless of the particular character code used.
  - However, some ancient implementations lack the `ctype.h` family.

A Word-Count Program

- **A word-counting program**

1) The program should read input character-by-character, and it should have some way of knowing when to stop.

2) It should be able to recognize and count the following units:

    Characters

    Lines

    Words

A Word-Count Program

A word-counting program

- Here's a pseudocode representation:

```
read a character
while there is more input
     increment character count
     if a line has been read, increment line count
     if a word has been read, increment word count
     read next character
```

- You already have a model for the input loop:

```
while ((ch = getchar()) != STOP)
{
   ...
}
```

`STOP` represents some value for `ch` that signals the end of the input.

A Word-Count Program

- **A word-counting program**
  - Here is the most straightforward test expression for detecting non-whitespace:

```
/* true if c is not whitespace */
c != ' ' && c != '\n' && c != '\t'
```

```
/* true if c is whitespace */
c == ' ' || c == '\n' || c == '\t'
```

A Word-Count Program

A word-counting program

- It is simpler to use the `ctype.h` function `isspace()`, which returns true if its argument is a whitespace character.
- `isspace(c)` is true if c is whitespace, and `!isspace(c)` is true if `c` isn't whitespace.

```
if c is not whitespace and inword is false
     set inword to true and count the word
if c is whitespace and inword is true
     set inword to false
```

A Word-Count Program

A word-counting program

- If you do use a Boolean variable,
- the usual idiom is to use the value of the variable itself as a test condition.

- `if (inword)` Instead of `if (inword == true)`

- `if (!inword)` Instead of `if (inword == false)`

A Word-Count Program

- **The wordcnt.c Program(1/2)**

```
#include <stdio.h>
#include <ctype.h>              // for isspace()
#include <stdbool.h>            // for bool, true, false
#define STOP '|'

int main(void)
{
    char c;                      // read in character
    char prev;                   // previous character read
    long n_chars = 0L;           // number of characters
    int n_lines = 0;             // number of lines
    int n_words = 0;             // number of words
    int p_lines = 0;             // number of partial lines
    bool inword = false;         // == true if c is in a word

    printf("Enter text to be analyzed (| to terminate):\n");
    prev = '\n';                 // used to identify complete lines
```

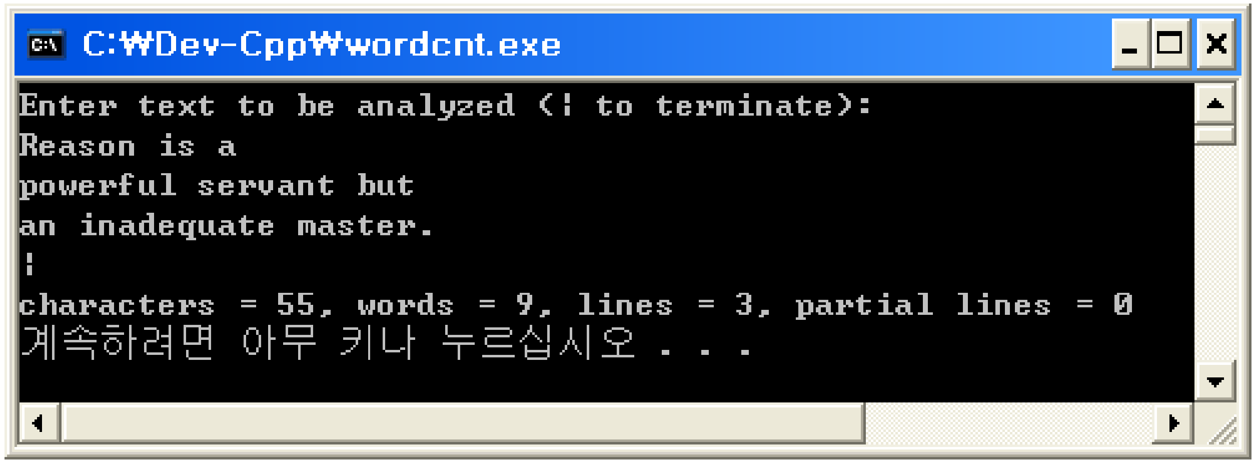
A Word-Count Program

The wordcnt.c Program(2/2)

```
    while ((c = getchar()) != STOP)
    {
        n_chars++;              // count characters
        if (c == '\n')
            n_lines++;          // count lines

        if (!isspace(c) && !inword)
        {
            inword = true;      // starting a new word
            n_words++;          // count word
        }

        if (isspace(c) && inword)
            inword = false;     // reached end of word
        prev = c;               // save character value
    }

    if (prev != '\n')
        p_lines = 1;
    printf("characters = %ld, words = %d, lines = %d, ",
          n_chars, n_words, n_lines);
    printf("partial lines = %d\n", p_lines);
    system("pause");
    return 0;
}
```

A Word-Count Program

The wordcnt.c Program



C:₩Dev-Cpp₩wordcnt.exe

```
Enter text to be analyzed (| to terminate):
Reason is a
powerful servant but
an inadequate master.
|
characters = 55, words = 9, lines = 3, partial lines = 0
계속하려면 아무 키나 누르십시오 . . .
```

A Word-Count Program

The wordcnt.c Program

- The program uses logical operators to translate the pseudocode to C.

```
if c is not whitespace and inword is false
```

- Gets translated into the following:

```
if (!isspace(c) && !inword)
```

- The entire test condition certainly is more readable than testing for each whitespace character individually:

```
if (c != ' ' && c != '\n' && c != '\t' && !inword)
```

The Conditional Operator: ?:

?: conditional operator

- C offers a shorthand way to express one form of the if else statement.
- It is called a conditional expression
- uses the ?: conditional operator
- Ex)

```
x = (y < 0) ? -y : y;
```

  - "If `y` is less than zero, `x = -y`; otherwise, `x = y`."

The Conditional Operator: ?:

?: conditional operator

- In `if else` terms, the meaning can be expressed as follows:

```
if (y < 0)
    x = -y;
else
    x = y;
```

- The following is the general form of the conditional expression:

```
expression1 ? expression2 : expression3
```

The Conditional Operator: ?:

?: conditional operator

- You can use the conditional expression when you have a variable to which you want to assign one of two possible values.

```
max = (a > b) ? a : b;
```

- This sets max to a if it is greater than b, and to b otherwise.

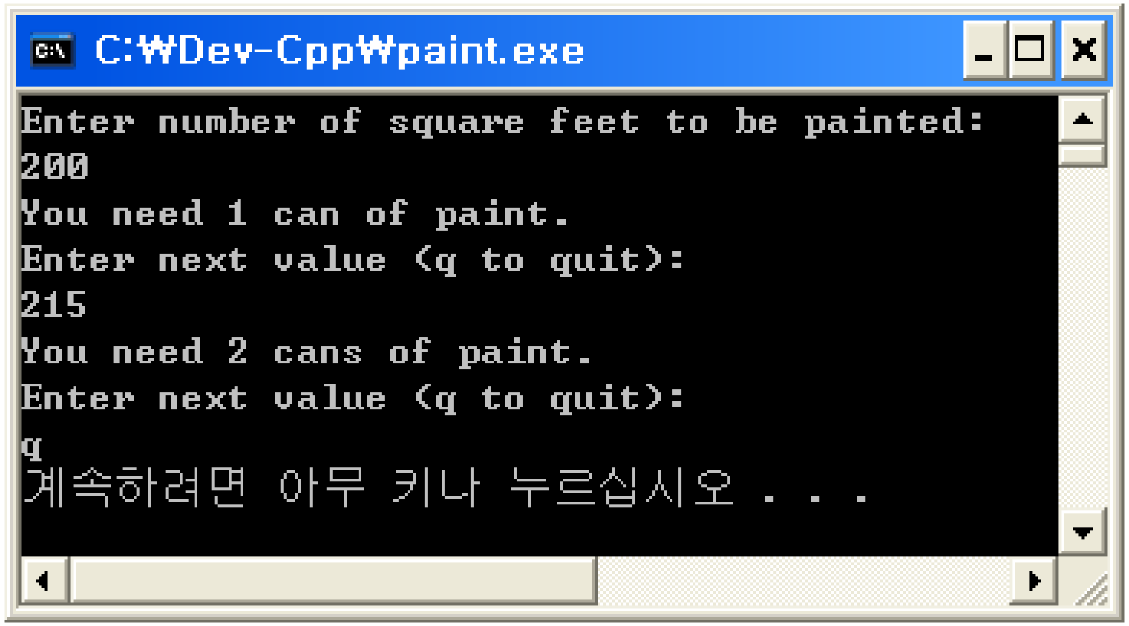
The Conditional Operator: ?:

The paint.c Program

```
#include <stdio.h>
#define COVERAGE 200        /* square feet per paint can */

int main(void)
{
    int sq_feet;
    int cans;

    printf("Enter number of square feet to be painted:\n");
    while (scanf("%d", &sq_feet) == 1)
    {
        cans = sq_feet / COVERAGE;
        cans += ((sq_feet % COVERAGE == 0)) ? 0 : 1;
        printf("You need %d %s of paint.\n", cans,
                cans == 1 ? "can" : "cans");

        printf("Enter next value (q to quit):\n");
    }
    system("pause");
    return 0;
}
```

The Conditional Operator: ?:

- The paint.c Program



```
Enter number of square feet to be painted:
200
You need 1 can of paint.
Enter next value (q to quit):
215
You need 2 cans of paint.
Enter next value (q to quit):
q
계속하려면 아무 키나 누르십시오 . . .
```

The Conditional Operator: ?:

The paint.c Program

- `cans` is rounded down to the integer part.
- If `sq_feet % COVERAGE` is 0,
- `COVERAGE` divides evenly into `sq_feet` and `cans` is left unchanged.
- Otherwise, there is a remainder, so `1` is added.

```
cans += ((sq_feet % COVERAGE == 0)) ? 0 : 1;
```

The Conditional Operator: ?:

The paint.c Program

- The final argument to the `printf()` function is also a conditional expression:

```
cans == 1 ? "can" : "cans");
```

- If the value of `cans` is 1, the string `"can"`is used.
- Otherwise, `"cans"` is used.

Loop Aids: `continue` and `break`

- **The `continue` Statement**
  - This statement can be used in the three loop forms.
  - When encountered,
  - it causes the rest of an iteration to be skipped and the next iteration to be started.
  - If the continue statement is inside nested structures,
  - it affects only the innermost structure containing it.

Loop Aids: continue and break

- The skippart.c Program(1/3)

```
#include <stdio.h>

int main(void)
{
    const float MIN = 0.0f;
    const float MAX = 100.0f;
    float score;
    float total = 0.0f;
    int n = 0;
    float min = MAX;
    float max = MIN;

    printf("Enter the first score (q to quit): ");
```

Loop Aids: continue and break

The skippart.c Program(2/3)

```
while (scanf("%f", &score) == 1)
{
    if (score < MIN || score > MAX)
    {
        printf("%0.1f is an invalid value. Try again: ",
                score);
        continue;
    }
    printf("Accepting %0.1f:\n", score);

    min = (score < min)? score: min;
    max = (score > max)? score: max;
    total += score;
    n++;

    printf("Enter next score (q to quit): ");
}
```

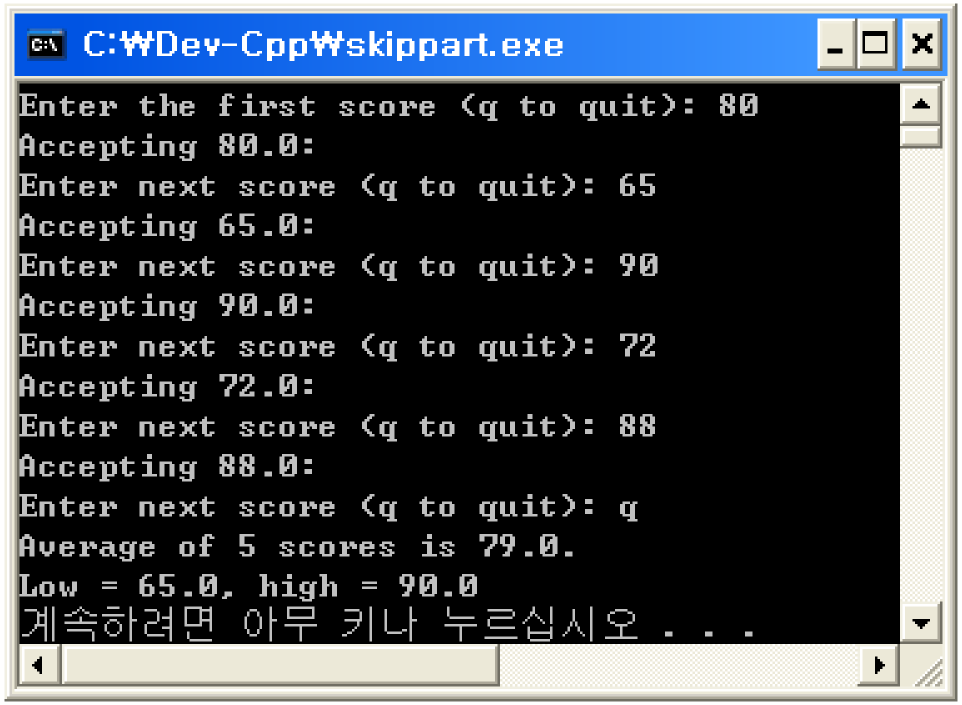
Loop Aids: continue and break

- The skippart.c Program(3/3)

```
    if (n > 0)
    {
        printf("Average of %d scores is %0.1f.\n", n, total / n);
        printf("Low = %0.1f, high = %0.1f\n", min, max);
    }
    else
        printf("No valid scores were entered.\n");
    return 0;
}
```

Loop Aids: continue and break

- The skippart.c Program



C:₩Dev-Cpp₩skippart.exe

```
Enter the first score (q to quit): 80
Accepting 80.0:
Enter next score (q to quit): 65
Accepting 65.0:
Enter next score (q to quit): 90
Accepting 90.0:
Enter next score (q to quit): 72
Accepting 72.0:
Enter next score (q to quit): 88
Accepting 88.0:
Enter next score (q to quit): q
Average of 5 scores is 79.0.
Low = 65.0, high = 90.0
계속하려면 아무 키나 누르십시오 . . .
```

Loop Aids: `continue` and `break`

The skippart.c Program

- Two ways you could have avoided using `continue`.
- One way is omitting the `continue` and making the remaining part of the loop an `else` block:

```
  if (score < 0 || score > 100)
      /* printf() statement */
else
{
      /* statements */
}
```

```
if (score >= 0 && score <= 100)
{
    /* statements */
}
```

Loop Aids: `continue` and `break`

The skippart.c Program

- Another use for `continue` is as a placeholder.
- Ex) the following loop reads and discards input up to, and including, the end of a line:

```
while (getchar() != '\n')
   ;
```

- The code is much more readable if you use continue:

```
while (getchar() != '\n')
   continue;
```

Loop Aids: continue and break

The skippart.c Program

- Don't use `continue` if it complicates rather than simplifies the code.

```
while ((ch = getchar() ) != '\n')
{
     if (ch == '\t')
          continue;
     putchar(ch);
}
```

Loop Aids: continue and break

The skippart.c Program

- The loop could have been expressed more economically as this:

```
while ((ch = getchar()) != '\n')
      if (ch != '\t')
           putchar(ch);
```

Loop Aids: `continue` and `break`

The skippart.c Program

- For the `while` and `do while` loops,
- the next action taken after the `continue` statement is to evaluate the loop test expression.

```
while (count < 10)
{
    ch = getchar();

    if (ch == '\n')
        continue;

    putchar(ch);
    count++;
}
```

Loop Aids: continue and break

The skippart.c Program

- For a `for` loop,
- the next actions are to evaluate the update expression and then the loop test expression.

```
for (count = 0; count < 10; count++)
{
    ch = getchar();
    if (ch == '\n')
        continue;
    putchar(ch);
}
```

Loop Aids: `continue` and `break`

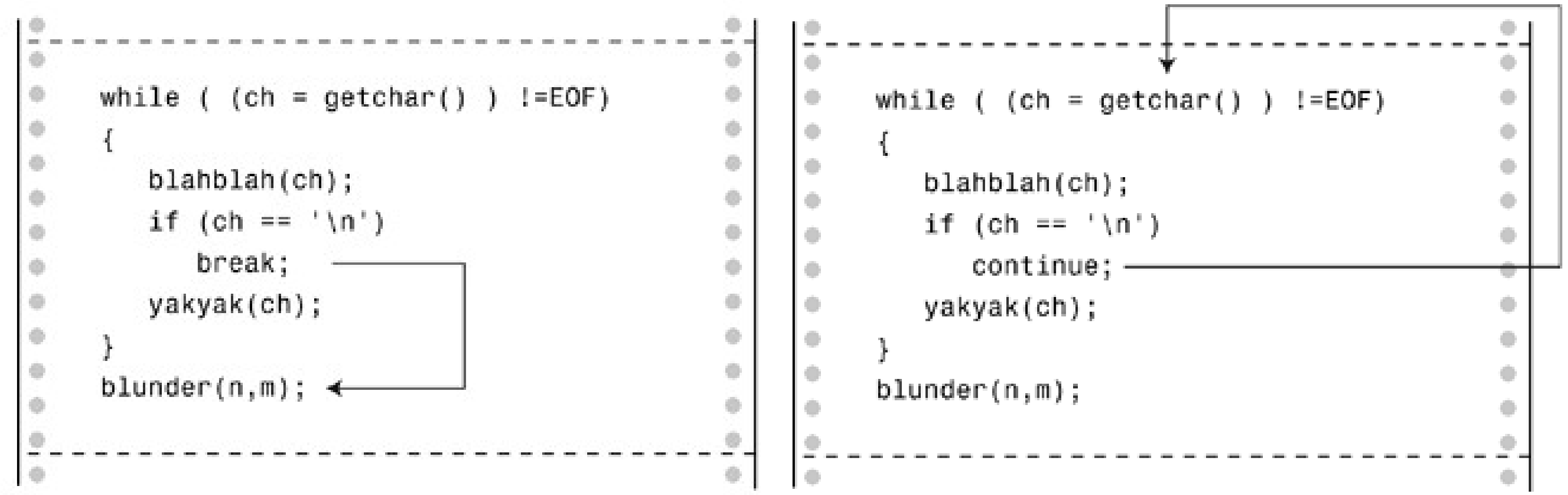
The skippart.c Program

- **In a `for` loop**
- Behaves slightly **differently** from the `while` example.
- when the continue statement is executed, first count is incremented and then it's compared to 10.
  - Only non-newline characters are displayed.
  - However, this time, newline characters are included in the count, so it reads 10 characters, including newlines.

Loop Aids: continue and break

The break Statement

- A `break` statement in a loop causes the program to break free of the loop that encloses it and to proceed to the next stage of the program.
- **Comparing break and continue**



```
while ( (ch = getchar() ) !=EOF)
{
    blahblah(ch);
    if (ch == '\n')
        break;
    yakyak(ch);
}
blunder(n,m);
```

```
while ( (ch = getchar() ) !=EOF)
{
    blahblah(ch);
    if (ch == '\n')
        continue;
    yakyak(ch);
}
blunder(n,m);
```

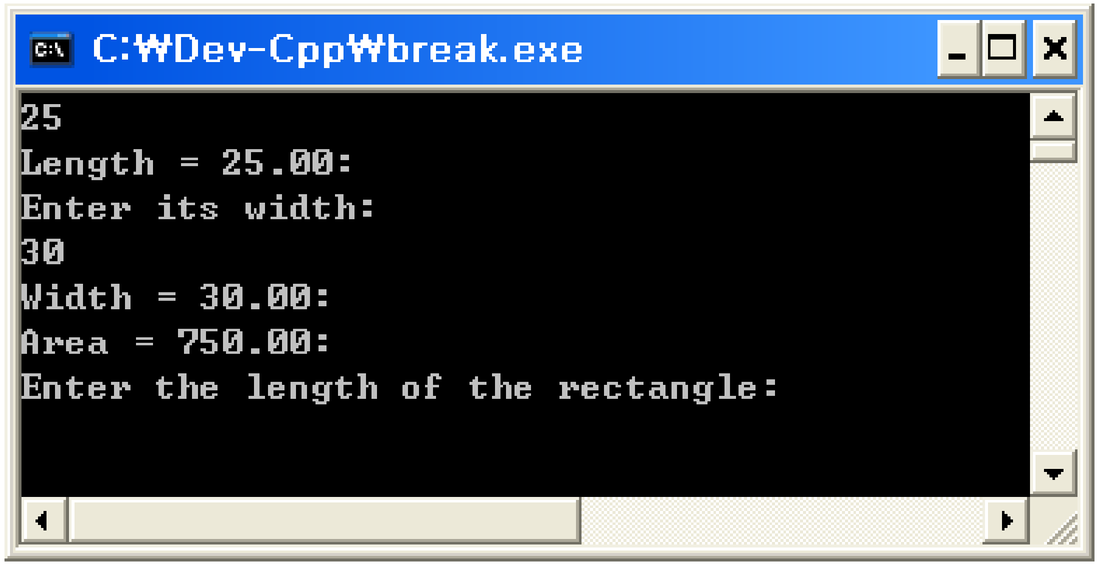
Loop Aids: continue and break

The break.c Program

```
#include <stdio.h>

int main(void)
{
    float length, width;

    printf("Enter the length of the rectangle:\n");

    while (scanf("%f", &length) == 1)
    {
        printf("Length = %0.2f:\n", length);
        printf("Enter its width:\n");
        if (scanf("%f", &width) != 1)
            break;
        printf("Width = %0.2f:\n", width);
        printf("Area = %0.2f:\n", length * width);
        printf("Enter the length of the rectangle:\n");
    }
    printf("Done.\n");

    return 0;
}
```

Loop Aids: continue and break

- The break.c Program



```
25
Length = 25.00:
Enter its width:
30
Width = 30.00:
Area = 750.00:
Enter the length of the rectangle:
```

Loop Aids: `continue` and `break`

The break.c Program

- You could have controlled the loop this way:

```
while (scanf("%f %f", &length, &width) == 2)
```

- However, using `break` makes it simple to echo each input value individually.

Loop Aids: `continue` and `break`

The break.c Program

- As with `continue`, don't use `break` when it complicates code.

```
while ((ch = getchar()) != '\n')
{
    if (ch == '\t')
          break;

    putchar(ch);
}
```

Loop Aids: continue and break

The break.c Program

- The logic is clearer if both tests are in the same place:

```
while ((ch = getchar() ) != '\n' && ch != '\t')
      putchar(ch);
```

Loop Aids: `continue` and `break`

The `break` statement

- Terminates the execution of the nearest enclosing loop or conditional statement in which it appears.

```
int p, q;
scanf("%d", &p);

while ( p > 0)
{
    printf("%d\n", p);
    scanf("%d", &q);
    while( q > 0)
    {
        printf("%d\n",p*q);
        if (q > 100)
            break;            // break from inner loop

        scanf("%d", &q);
    }
    if (q > 100)
        break;                // break from outer loop

    scanf("%d", &p);
}
```

Multiple Choice: switch and break

- **The animals.c Program(1/3)**

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

int main(void)
{
    char ch;

    printf("Give me a letter of the alphabet, and I will give ");
    printf("an animal name\nbeginning with that letter.\n");
    printf("Please type in a letter; type # to end my act.\n");

    while ((ch = getchar()) != '#')
    {
        if('\n' == ch)
            continue;
        if (islower(ch))      /* lowercase only          */
```

Multiple Choice: switch and break

- The animals.c Program(2/3)

```
switch (ch)
{
    case 'a' :
          printf("argali, a wild sheep of Asia\n");
          break;
    case 'b' :
          printf("babirusa, a wild pig of Malay\n");
          break;
    case 'c' :
          printf("coati, racoonlike mammal\n");
          break;
    case 'd' :
          printf("desman, aquatic, molelike critter\n");
          break;
    case 'e' :
          printf("echidna, the spiny anteater\n");
          break;
    case 'f' :
          printf("fisher, brownish marten\n");
          break;
    default :
          printf("That's a stumper!\n");
}                /* end of switch          */
```

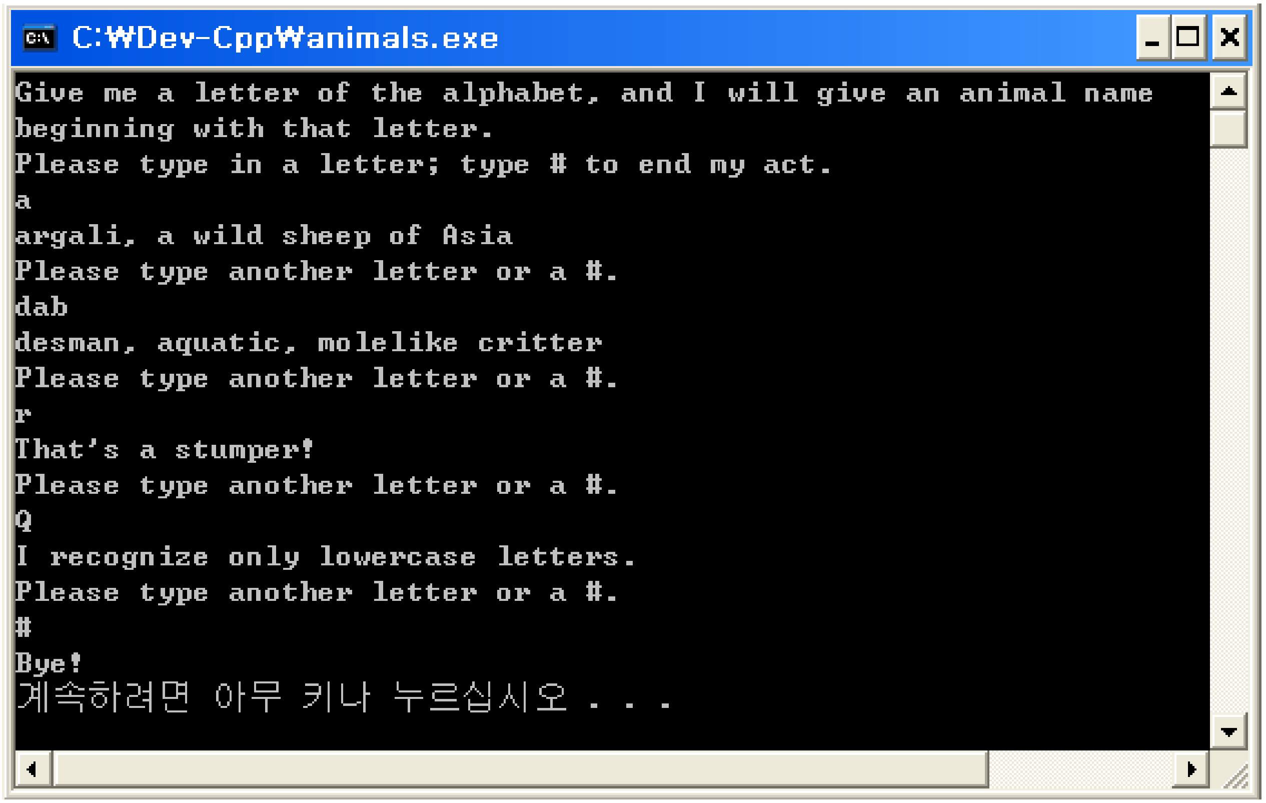
Multiple Choice: switch and break

The animals.c Program(3/3)

```
        else
            printf("I recognize only lowercase letters.\n");

        while (getchar() != '\n')
              continue;           /* skip rest of input line */

        printf("Please type another letter or a #.\n");
    }                             /* while loop end          */
    printf("Bye!\n");

    return 0;
}
```

Multiple Choice: switch and break

- **The animals.c Program**



```
Give me a letter of the alphabet, and I will give an animal name
beginning with that letter.
Please type in a letter; type # to end my act.
a
argali, a wild sheep of Asia
Please type another letter or a #.
dab
desman, aquatic, molelike critter
Please type another letter or a #.
r
That's a stumper!
Please type another letter or a #.
Q
I recognize only lowercase letters.
Please type another letter or a #.
#
Bye!
계속하려면 아무 키나 누르십시오 . . .
```

Multiple Choice: `switch` and `break`

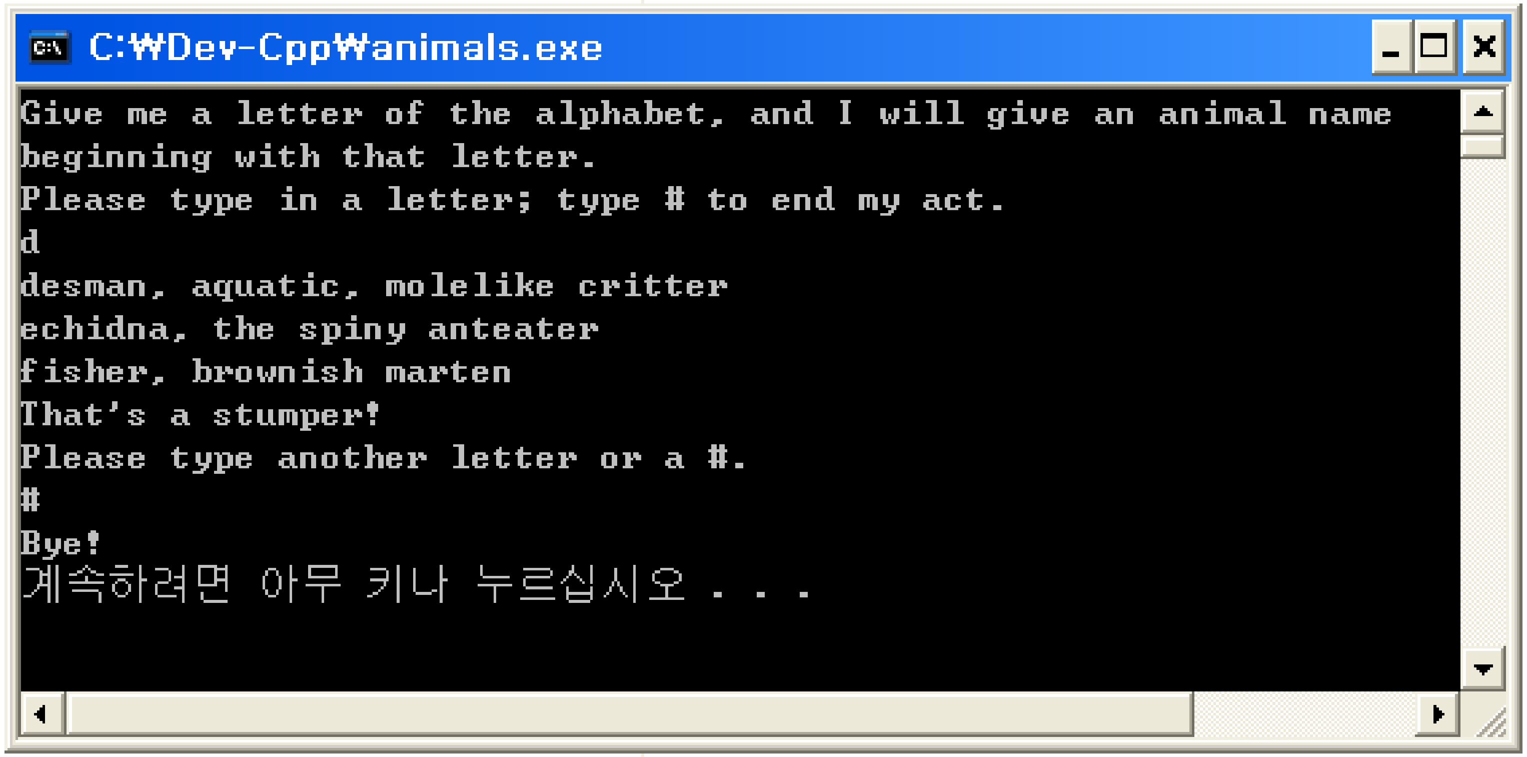
Using the `switch` Statement

- The expression in the parentheses following the word `switch` is evaluated.
- What about the `break` statement?
- It causes the program to break out of the `switch`.
- Skip to the next statement after the `switch`.

Multiple Choice: switch and break

Using the switch Statement

- if you removed all the `break` statements from the program and then ran the program using the letter d, you would get this exchange:

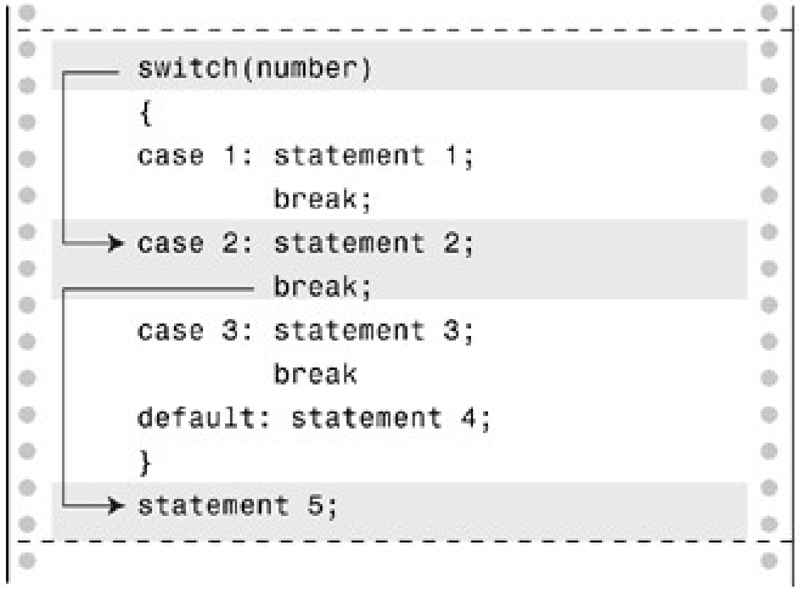


```
Give me a letter of the alphabet, and I will give an animal name
beginning with that letter.
Please type in a letter; type # to end my act.
d
desman, aquatic, molelike critter
echidna, the spiny anteater
fisher, brownish marten
That's a stumper!
Please type another letter or a #.
#
Bye!
계속하려면 아무 키나 누르십시오 . . .
```

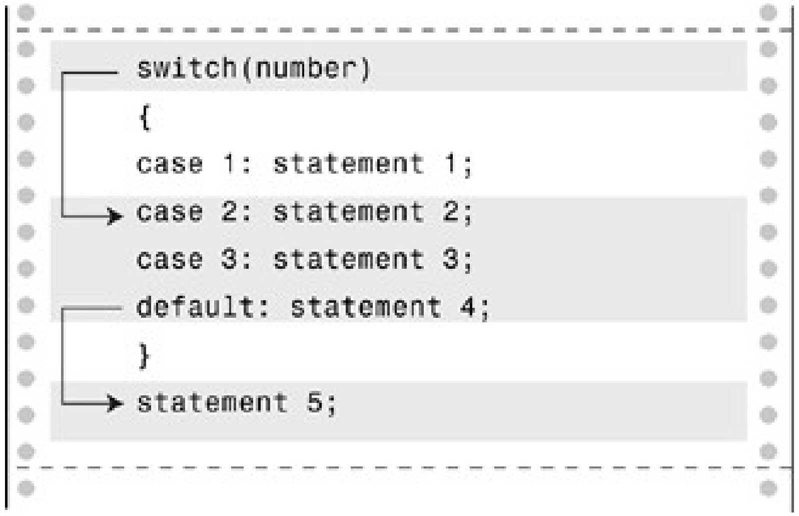
Multiple Choice: `switch` and `break`

Using the `switch` Statement

- Program flow in `switch`es, with and without `break`s.



```
switch(number)
{
case 1: statement 1;
        break;
case 2: statement 2;
        break;
case 3: statement 3;
        break
default: statement 4;
}
statement 5;
```



```
switch(number)
{
case 1: statement 1;
case 2: statement 2;
case 3: statement 3;
default: statement 4;
}
statement 5;
```

Multiple Choice: `switch` and `break`

Using the `switch` Statement

- The structure of a `switch`.

```
switch (integer expression)
{
        case constant1:
                  statements     optional
        case constant2:
                  statements     optional
        default :                optional
                  statements     optional
}
```

Multiple Choice: switch and break

Reading Only the First Character of a Line

- The other new feature incorporated into animals.c is how it reads input.
- When dab was entered, **only the first character was processed.**

```
while (getchar() != '\n')
    continue;            /* skip rest of input line */
```

– This loop reads characters from input up to and including the newline character generated by the Enter key.

Multiple Choice: switch and break

Reading Only the First Character of a Line

- Suppose a user starts out by pressing Enter so that the first character encountered is a newline.

```
if (ch == '\n')
    continue;
```

Multiple Choice: switch and break

The vowels.c Program (1/2)

- **Multiple case Labels**

```
#include <stdio.h>

int main(void)
{
    char ch;
    int a_ct, e_ct, i_ct, o_ct, u_ct;

    a_ct = e_ct = i_ct = o_ct = u_ct = 0;

    printf("Enter some text; enter # to quit.\n");
```

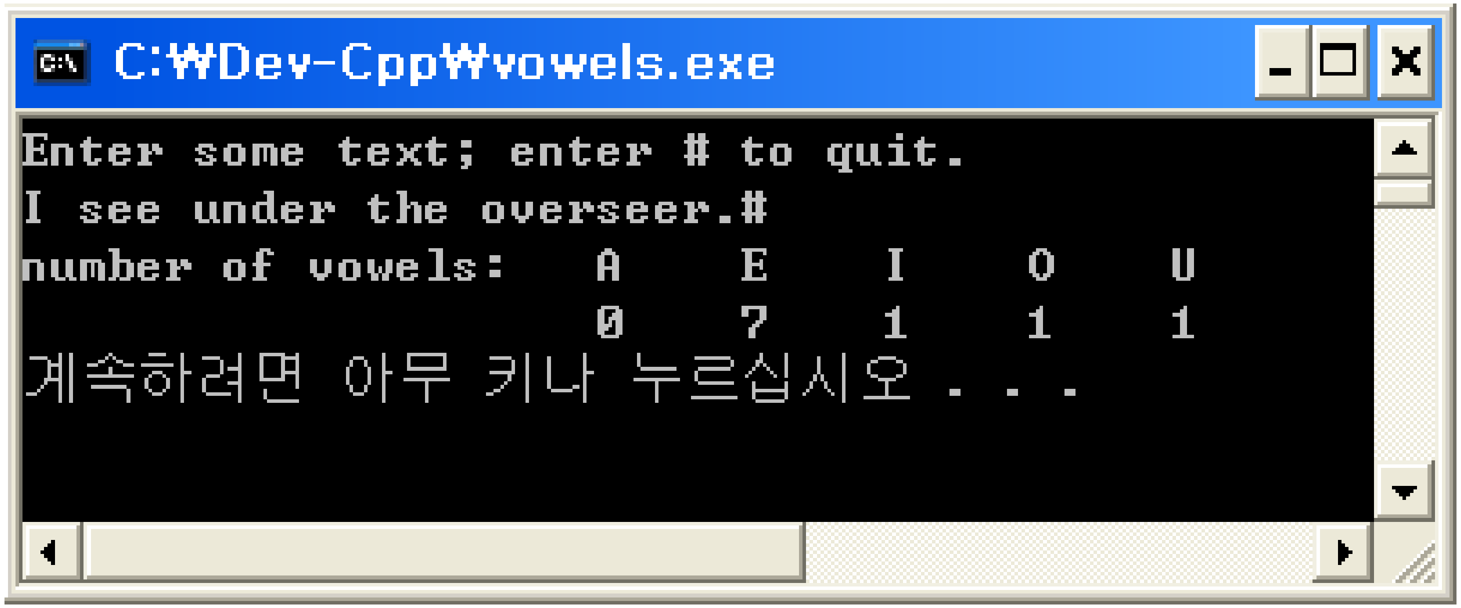
Multiple Choice: switch and break

The vowels.c Program (2/2)

```
    while ((ch = getchar()) != '#')
    {
        switch (ch)
        {
            case 'a' :
            case 'A' :  a_ct++;
                        break;
            case 'e' :
            case 'E' :  e_ct++;
                        break;
            case 'i' :
            case 'I' :  i_ct++;
                        break;
            case 'o' :
            case 'O' :  o_ct++;
                        break;
            case 'u' :
            case 'U' :  u_ct++;
                        break;
            default :   break;
          }                    /* end of switch  */
    }                          /* while loop end */
    printf("number of vowels:   A    E    I    O    U\n");
    printf("                  %4d %4d %4d %4d %4d\n",
          a_ct, e_ct, i_ct, o_ct, u_ct);
    return 0;
}
```

Multiple Choice: switch and break

The vowels.c Program (2/2)



C:₩Dev-Cpp₩vowels.exe

```
Enter some text; enter # to quit.
I see under the overseer.#
number of vowels:    A     E     I     O     U
                     0     7     1     1     1
계속하려면 아무 키나 누르십시오 . . .
```

Multiple Choice: switch and break

The vowels.c Program

- You can avoid multiple labels by using the `toupper()` function from the `ctype.h` family to convert all letters to uppercase before testing:

```
while ((ch = getchar()) != '#')
{
    ch = toupper(ch);
    switch (ch)
    {
      case 'A' :  a_ct++;
                  break;
      case 'E' :  e_ct++;
                  break;
      case 'I' :  i_ct++;
                  break;
      case 'O' :  o_ct++;
                  break;
      case 'U' :  u_ct++;
                  break;
      default :   break;
    }                               /* end of switch  */
}                                   /* while loop end */
```

Multiple Choice: switch and break

The vowels.c Program

- Or, if you want to leave ch unchanged, use the function this way:

```
switch(toupper(ch))
```

Multiple Choice: `switch` and `break`

`switch` and `if else`

- You can't use a `switch` if your choice is based on evaluating a float variable or expression.
- Nor can you conveniently use a `switch` if a variable must fall into a certain range.

```
if (integer < 1000 && integer > 2)
```

The `goto` Statement

The `goto` statement

- The **`goto`** statement has two parts
- The **`goto`** and a **label name**
- The label is named following the same convention used in naming a variable.

```
goto part2;
```

The `goto` Statement

- **The `goto` statement**
  - For the preceding statement to work, the function must contain another statement bearing the `part2` label.
  - This is done by beginning a statement with the label name followed by a colon:

```
part2: printf("Refined analysis:\n");
```

The goto Statement

Avoiding goto

- Handling an `if` situation that requires more than one statement:

```
if (size > 12)
    goto a;
goto b;

a: cost = cost * 1.05;
   flag = 2;
b: bill = cost * flag;
```

The goto Statement

Avoiding goto

- The standard C approach of using a compound statement or block is much easier to follow:

```
if (size > 12)
{
        cost = cost * 1.05;
        flag = 2;
}
bill = cost * flag;
```

The `goto` Statement

- **Avoiding `goto`**
  - Choosing from two alternatives:

```
if (ibex > 14)
   goto a;
sheds = 2;
goto b;
a: sheds= 3;
b: help = 2 * sheds;
```

The goto Statement

Avoiding goto

- Having the `if else` structure available allows C to express this choice more cleanly:

```
if (ibex > 14)
     sheds = 3;
else
     sheds = 2;

help = 2 * sheds;
```

The goto Statement

- **Avoiding goto**
  - Setting up an indefinite loop:

```
readin: scanf("%d", &score);
if (score > 0)
   goto stage2;
lots of statements;
goto readin;
stage2: more stuff;
```

The goto Statement

- **Avoiding goto**
  - Use a `while` loop instead:

```
scanf("%d", &score);
while (score <= 0)
{
         lots of statements;
         scanf("%d", &score);
}
```

The `goto` Statement

Avoiding `goto`

- Skipping to the end of a loop and starting the next cycle.
- Use `continue` instead.

- Leaving a loop.
- Use `break` instead.

- Actually, `break` and `continue` are specialized forms of `goto`.

- Leaping madly about to different parts of a program. In a word, don't!

The goto Statement

Avoiding goto

- There is one use of goto tolerated by many C practitioners—getting out of a nested set of loops if trouble shows up:

```
while (funct > 0)
     {
     for (i = 1, i <= 100; i++)
          {
          for (j = 1; j <= 50; j++)
               {
               statements galore;
               if (bit trouble)
                    goto help;
               statements;
               }
          more statements;
          }
     yet more statements;
     }
and more statements;
help : bail out;
```

Summary

- Keywords
- `if, else, switch, continue, break, case, default, goto`
- Operators
- `&& || ?:`
- Functions
- `getchar(), putchar(), the ctype.h family`
- How to use the `if` and `if else` statements and how to nest them
- Using logical operators to combine relational expressions into more involved test expressions

Summary

- C's conditional operator
- The `switch` statement
- The `break`, `continue`, and `goto` jumps
- Using C's character I/O functions: `getchar()` and `putchar()`
- The family of character-analysis functions provided by the `ctype.h` header file