Lecture 8: Selection Statements (Cont.)
Recap to previous lecture!

• If – else
• Relational Operators
• Equality Operators
• Logical Operators
The *if*-else Clause

```java
if ( expression )
{
    Statements - A
}
else
{
    Statements - B
}
```

- The statements - A are executed *if the expression has the value true*.
- The statements - B are executed *if the expression has the value false*. 
Nested If Else Clause

- It’s not unusual for if statements to be nested inside other if statements:

```plaintext
if (i > j) {
    if (i > k) {
        max = i;
    } else {
        max = k;
    }
} else {
    if (j > k) {
        max = j;
    }
}
```
When to use if scope

• What is the difference between:

```plaintext
if( i < 0 )
{
    i = i + 5;
    K = k +10;
}
```

```plaintext
if( i < 0 )
{
    i = i + 5;
    K = k +10;
}
```
The “Dangling else” Problem

• The indentation suggests that the else clause belongs to the outer if statement.

• However, C follows the rule that an else clause belongs to the nearest if statement that hasn’t already been paired with an else.
Cascaded if Statements

- A “cascaded” if statement is used to write a series of test conditions, stopping as soon as one of them is true.

```
if ( expression )
    statements
else if ( expression )
    statements
...
else if ( expression )
    statements
else
    statements
```
The **switch** Statement

- A cascaded `if` statement can be used to compare an expression against a series of values:

```c
if (grade == 4)
    printf("Excellent");
else if (grade == 3)
    printf("Good");
else if (grade == 2)
    printf("Average");
else if (grade == 1)
    printf("Poor");
else if (grade == 0)
    printf("Failing");
else
    printf("Illegal grade");
```
The switch Statement

- The `switch` statement is an alternative:

```c
int grade = 3;
switch (grade) {
    case 4:  printf("Excellent");
             break;
    case 3:  printf("Good");
             break;
    case 2:  printf("Average");
             break;
    case 1:  printf("Poor");
             break;
    case 0:  printf("Failing");
             break;
    default: printf("Illegal grade");
             break;
}
```
The `switch` Statement

- A `switch` statement may be easier to read than a cascaded `if` statement.
- The common form of the `switch` statement:

```
switch ( expression ) {
    case constant-expression : statements
    ...
    case constant-expression : statements
    default : statements
}
```
The switch Statement

• The word `switch` must be followed by an integer expression—the **controlling expression**—in parentheses.

• *Characters* are treated as integers in C and thus can be tested in `switch` statements.

• *Floating-point numbers and strings* don’t qualify, however.
The `switch` Statement

- Each case begins with a label of the form
  ```
  case constant-expression :
  ```
- A `constant expression` is much like an ordinary expression except that it can’t contain variables or function calls.
- After each case label comes any number of statements.
- *No braces are required* around the statements.
- The last statement in each group is normally `break`. 
The **switch** Statement

- *Duplicate* case labels aren’t allowed.

- The *order of the cases doesn’t matter*, and the *default* case doesn’t need to come last.

- Executing a *break* statement causes the program to “break” out of the *switch* statement; execution continues at the next statement after the *switch*. 
Program: Printing a Date

• Develop a program that formats the user-input date in the form of month/day/year form:
  Enter date (mm/dd/yy): 7/19/14
  Dated this 19th day of July, 2014.

• The program uses switch statements to add “th” (or “st” or “nd” or “rd”) to the day, and to print the month as a word instead of a number.
main(void)
{
    int month, day, year;

    printf("Enter date (mm/dd/yy): ");
    scanf("%d /%d /%d", &month, &day, &year);

    printf("Dated this %d", day);
    switch (day) {
        case 1: case 21: case 31:
            printf("st"); break;
        case 2: case 22:
            printf("nd"); break;
        case 3: case 23:
            printf("rd"); break;
        default:
            printf("th"); break;
    }

    printf(" day of ");
    switch (month) {
        case 1: printf("January"); break;
        case 2: printf("February"); break;
        ...
        case 11: printf("November"); break;
        case 12: printf("December"); break;
    }

    printf("", 20%d.\n", year);
    return 0;
}
Loops in C
Loops in C.

• A loop is a statement that repeatedly executes some other statements (the body of the loop) for a number of times (the iterations).

• Every loop has a *controlling expression*.

• Each time the loop body is executed (an *iteration* of the loop), the controlling expression is evaluated.
  – If the expression is false (integer zero), the loop stops.
C provides three loop statements:

– The *while statement* is used for loops whose controlling expression is tested *before* the loop body is executed.

– The *do statement* is used if the expression is tested *after* the loop body is executed.

– The *for statement* is convenient for loops that increment or decrement *a counting variable*.
The **while** Statement

- Using a **while** statement is the easiest way to set up a loop.
- The **while** statement has the form
  
  ```
  while ( expression )
  {
    statement
  }
  
  ```

- `expression` is the controlling expression; `statement` is the loop body.
while \( i < n \) /* controlling exp */
{
    i = i \times 2; /* loop body */
}

- When a `while` statement is executed, the controlling expression is evaluated *first*.

- If its value is nonzero (true), the loop body is executed and the expression is tested again. The process continues until the controlling expression eventually *has the value zero*. 
The while Statement

- A *trace* of the loop when \( n \) has the value 10 and \( i = 1 \):

```plaintext
i = 1;    \hspace{1cm} i \text{ is now } 1.
Is i < n? \hspace{1cm} Yes; continue.
i = i * 2; \hspace{1cm} i \text{ is now } 2.
Is i < n? \hspace{1cm} Yes; continue.
i = i * 2; \hspace{1cm} i \text{ is now } 4.
Is i < n? \hspace{1cm} Yes; continue.
i = i * 2; \hspace{1cm} i \text{ is now } 8.
Is i < n? \hspace{1cm} Yes; continue.
i = i * 2; \hspace{1cm} i \text{ is now } 16.
Is i < n? \hspace{1cm} No; exit from loop.
```
Develop C program that prints countdown messages from 10 to 0, in separate lines.
The while Statement

i = 10;
while (i > 0) {
    printf(" %d \n", i);
    i--;
}
Guess the output

```c
i = 10;
while (i > 0) {
    printf(" %d \n", i);
    i++;
}
printf(" done with the loop");
```
Infinite Loops

• A while statement won’t terminate if the controlling expression always has a nonzero value.

• C programmers sometimes deliberately create an infinite loop by using a nonzero constant as the controlling expression:
  while (1) ...

• A while loop of this form will execute forever unless its body contains a statement that transfers control out of the loop (e.g., break) or calls a function that causes the program to terminate.
Program: Printing a Table of Squares

• Develop a program that print a table of squares.
• The user specifies the number of entries in the table:

This program prints a table of squares. Enter number of entries in table: 5

1  1
2  4
3  9
4 16
5 25
Program: Counting even numbers

• Develop a program that counts even numbers between 1 to n, where n is user input.