Week 4: Control Structures - Repetition

BJ Furman
18FEB2010
The Plan for Today

- Review control structures
  - Sequence
  - Selection
  - Repetition
- Review relational operators, pseudocode, and flowcharting
- Repetition structures
  - While
  - Do/While
  - For
- Repetition structure example
Learning Objectives

- Explain the three basic types of control structures
- Explain how the three kinds of repetition structures work
- Apply the concept of repetition control structures to a practical problem
Control Structures - Review

All programs can be written in terms of three control structures (like building blocks)

- **Sequence**
  - ‘Built-in’ to C
    - Unless otherwise directed, one statement after the next is executed

- **Selection** (three types: IF, IF-ELSE, SWITCH)
  - Depending on a *condition*, *select* between one statement or another
    - If var1 is greater than 10, do *this*…, else do *that*…

- **Repetition** (three types: WHILE, DO-WHILE, FOR)
  - Depending on a *condition*, execute one or more statements *repeatedly*
Selection Structure

- Three kinds of selections structures
  - **IF** (also called, ‘single-selection’)
    - if *condition* is true
      - Perform action
    - if *condition* is false, action is skipped
  - **IF/ELSE** (also called, ‘double-selection’)
    - if *condition* is true
      - Perform action
    - else (if *condition* is false)
      - Perform a different action
  - **SWITCH** (also called ‘multiple-selection’)
    - Allows selection among many actions depending on the value of a variable or expression
Repetition Structure

- Often need to **repeat** an action or calculation
  - Ex. Find and print the distance traveled by an object dropped from a height at \( t = 0 \) sec each second over the next 10 seconds

- Repetition structures are often called ‘**loops**’

\[
\sum \vec{F} = ma
\]

\[
d = \frac{1}{2} gt^2 + v_0 t + d_0
\]

A dynamics problem (ME 101)
3 Types of Repetition Structures

- **while**
  - Tests a *condition* at the *beginning* of the loop
    - condition must first be true for the loop to run even once

- **do/while**
  - Tests a *condition* at the *end* of the loop
    - loop will run at least once

- **for**
  - Facilitates initializing and incrementing the variable that controls the loop
  - Especially helpful for:
    - Looping for a *known* number of times
    - Loops that *count* or that need to increment a variable
while Loop Structure

- General form of a while loop:

  ```
  while(condition) /* while condition not equal to zero */
    statement1; /* execute this statement */
  
  - When `condition` becomes false, looping stops and the next statement is executed
  ```

- Compound statement form:

  ```
  while(condition)
  {
    statement1;
    statement2;
    statement3;
  }
  ```
while Loop - Pseudocode

WHILE *condition* is TRUE, repeat:

  Statement1
  Statement2

END WHILE
while Loop - Flowchart View

- Statement is executed \textit{while} condition is true
  - Note that the condition must first be true in order for the statement to be executed even once
Program Development

- Define the problem
  - Calculate and print the distance traveled each second by an object dropped from a height in a standard gravitational field beginning at $t = 0$ sec over the next 10 seconds
  - Simplify
    - Assume $v_0$ and $d_0$ are both zero

- Inputs
- Outputs
  - time
  - distance

\[ d = \frac{1}{2} gt^2 \]
Solution Algorithm

Pseudocode

1. Declare variables
   - d (distance traveled)
   - t (time)
2. Initialize variables
3. While time is less than or equal to 10
   - calculate d
   - print time
   - print distance
   - increment time
4. End
Solution Code

- free_fall_d_vs_time.c
Practice - Procedure

Procedure

- Divide into groups of 4
- Introduce yourselves
- Together (all four in the group)
  - Define the problem
  - Determine inputs and outputs
- Split into pairs (sub-groups)
  - One pair – pseudocode the algorithm
  - Other pair – flowchart the algorithm
- Swap one from each pair and share pseudocode and flowchart

Write code from flowchart or pseudocode
Practice - Problem

- Write a program that calculates the average exam score for a class

  Specifications
  - Prompts user for the number of scores to average
  - Prompts user for each score
  - Echoes each score entered
  - Calculates the average score
  - Prints out the average score
Practice - Solution
do-while Loop Structure

- General form of a do/while loop:
  ```
  do
  statement1; /* execute this statement */
  while (condition); /* while condition is TRUE */
  ```

- When condition becomes false, looping stops, and the next statement after the while is executed
  - Note: statement1 will be executed at least once
  - Remember: to DO more than one statement in the loop, enclose the statements in curly braces `{ }`
    - called a compound statement
do-while Loop - Pseudocode

REPEAT:
  Statement1
  Statement2
WHILE condition is TRUE
do-while Loop – Flow Chart View

- statement is executed at least once
for Loop Structure

- General form of a **for** loop:
  ```
  for(expression1; expression2; expression3)
  statement1;          /* execute this statement */
  ```
  - **expression1** initializes the variable controlling the loop
    - `i = 0;`
  - **expression2** is the **condition** for continuing the loop
    - `i <= 10;`
  - **expression3** increments the control variable
    - `i++ /* same as i=i+1 */`
  - Note that there is **NO** semicolon after **expression3**! or after the closing parenthesis
  - To execute more than one statement in the **for** loop, enclose them in curly braces `{ }`
for Loop - Pseudocode

FOR n times, REPEAT:
    Statement1
    Statement2
END FOR
for Loop – Flow Chart View

expression1

Tests the loop control variable to see if it is time to quit looping: ex. i < 10;

expression2

T -> statement

F

expression3

Initializes the loop control variable: ex. i = 0;

Increments the loop control variable: ex. i++
for Loop Example

- double_it.c
break and continue Statements

- **break** immediately exit from a loop
  - Recall its use in the **switch** selection structure

```c
int x;
    for(x=1; x<=10; x++)
    {
        if(x == 5)
            break;
        printf("%d ", x);
    }
printf("Broke out at x == %d
",x);
```

```
1 2 3 4 5
```

- **continue** skips remaining statements in the body of a repetition structure

```c
int x, y;
    for(x=1; x<=10; x++)
    {
        if(x == 5)
            break;
        printf("%d ", x);
    }
printf("Skipped x == %d
",y);
```

```
1 2 3 4 6 7 8 9 10 Skipped x == 5
```

Adapted from Deitel & Deitel, C How to Program, 3rd ed., p. 119
Appendix – Recap of Selection Structures
Selection Structure

- Three kinds of selections structures
  - **if** (also called, ‘single-selection’)
    - if *condition* is true
      - Perform action
    - if *condition* is false, action is skipped
  - **if/else** (also called, ‘double-selection’)
    - if *condition* is true
      - Perform action
    - else (if *condition* is false)
      - Perform a different action
  - **switch** (also called ‘multiple-selection’)
    - Allows selection among many actions depending on the value of a variable or expression
IF statement (single-selection)

- Syntax

```plaintext
if (expression) /* if expression is TRUE (not equal to zero) */
    statement1; /* then execute this statement */
statement2; /* otherwise execute this statement */
```

- Notes

  - Indent statements
  - Can have multiple statements
    - Enclose a ‘block’ of statements using { } (curly braces)
```plaintext
if ( x <= 2 )
{
    statement1;
    statement2;
}
statement3;
```
IF statement example

- **Pseudocode** (notice indentation!)
  
  If speed is greater than 65 mph
  print “You’re speeding!”

- **C code**
  
  if(speed > 65)
      printf("You’re speeding!\n");

- **C code with statement block**
  
  if(speed > 65)
      /* statements below executed only if speed > 65 is true */
      {
          printf("You’re speeding!\n");
          printf("Slow down!\n");
          printf("Keep speed below 65 MPH\n");
      }
Single Selection IF - Flowchart

Speed > 65

TRUE
Print “You’re speeding!”

FALSE
IF-ELSE statement - Double Selection

Syntax

```c
if (expression) /* if expression is TRUE */
    statement1; /* execute this statement */
else /* else execute the following statement */
    statement2;
```

Notes:

- If `expression` is non-zero, `statement1` is executed, then the program continues with the statement after `statement2`, i.e., `statement2` is skipped.
- If `expression` is equal to zero, `statement1` is skipped and `statement2` is executed, then the program continues with the statement after `statement2`.
Double-Selection IF - Flowchart

Print “Within limit”

Speed > 65

TRUE

Print “Over speed limit!”

FALSE
IF-ELSE statement example

- Pseudocode (notice indentation!)
  
  If speed is greater than 65 mph
  print “Over speed limit!”
  else
  print “Within speed limit”

- C code

  if(speed > 65)  
    printf(“Over speed limit!\n”);  
  else  
    printf(“Within limit\n”);
Relational Operators

Important for constructing the decision expression

### Practice

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 &lt; 7</td>
<td>_____</td>
</tr>
<tr>
<td>5 &gt; 7</td>
<td>_____</td>
</tr>
<tr>
<td>7 &lt;= 7</td>
<td>_____</td>
</tr>
<tr>
<td>8 &gt;= 7</td>
<td>_____</td>
</tr>
<tr>
<td>5 == 5</td>
<td>_____</td>
</tr>
<tr>
<td>5 == 7</td>
<td>_____</td>
</tr>
<tr>
<td>5.0 == 5</td>
<td>_____</td>
</tr>
<tr>
<td>6 != 5</td>
<td>_____</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operations</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>::</td>
<td></td>
</tr>
<tr>
<td>() []</td>
<td>left to right</td>
</tr>
<tr>
<td>Function_name()</td>
<td>right to left</td>
</tr>
<tr>
<td>. -&gt;</td>
<td>left to right</td>
</tr>
<tr>
<td><code> </code> ! <code>+</code> - ` * &amp; (type) sizeof</td>
<td>right to left</td>
</tr>
<tr>
<td><code> </code> / <code>%</code> . <code>*</code> ./</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt;= &gt; &gt;=</td>
<td>left to right</td>
</tr>
<tr>
<td>== != &amp; ^</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>? :</td>
<td>right to left</td>
</tr>
<tr>
<td>+= -= *= /= %=</td>
<td>= &lt;&lt;= &gt;&gt;= &lt;&lt;= &gt;&gt;=</td>
</tr>
<tr>
<td>,</td>
<td>left to right</td>
</tr>
</tbody>
</table>

Adapted from H. Cheng chap04.ppt, slide 5
Logical operators for more complex decisions

- Logical AND operator \( \&\& \) (double ampersand)
  - ```
    if(switch1 == 0 && switch2 == 1)
    turn on the motor
  ```
  - The condition evaluates to TRUE if and only if BOTH expressions on either side of \( \&\& \) evaluate to TRUE
    - Note operator precedence
  - Otherwise condition evaluates to FALSE
    - Beware of ‘short circuit evaluation’
      - Make the condition most likely to be FALSE the left-most condition
Compound Condition - ||

- Logical operators for more complex decisions, cont.
  - Logical OR operator || (double vertical bar)
    - `if(switch1 == 0 || switch2 == 1)`
      - turn on the motor
    - The condition evaluates to TRUE if one or the other or both expressions on either side of && evaluate to TRUE
      - Note operator precedence
    - Otherwise condition evaluates to FALSE
      - Beware of ‘short circuit evaluation’
        - Make the condition most likely to be TRUE the left-most condition
Nesting selection structures

- Selection structures can be **stacked and nested** to handle more sophisticated decision/action functionality
  - Ex. Figuring grades
    - Pseudocode →

Notes:

- “an *else* is always associated with the nearest previous if”  
  (Darnell & Margolis, 1996)
- Use braces ({ }) to clarify the association of the *else* for other situations where the decision structure is more complicated

Grade Determination for Overall Percentage (OP)

- If student’s grade is greater than or equal to 90
  - Print ‘A’
- else
  - If student’s grade is greater than or equal to 80
    - Print ‘B’
  - else
    - If student’s grade is greater than or equal to 70
      - Print ‘C’
    - else
      - If student’s grade is greater than or equal to 60
        - Print ‘D’
      - else
        - Print ‘F’

Adapted from Deitel & Deitel, C How to Program, 3rd ed., p. 64
Nesting If/else – C Code – Two Ways

```c
if (grade > 90)
    printf(“A\n”);
else
    if (grade >= 80)
        printf(“B\n”);
    else
        if (grade >= 70)
            printf(“C\n”);
        else
            if (grade >= 60)
                printf(“D\n”);
            else
                printf(“F\n”);
```

Or

```c
if (grade > 90)
    printf(“A\n”);
else if (grade >= 80)
    printf(“B\n”);
else if (grade >= 70)
    printf(“C\n”);
else if (grade >= 60)
    printf(“D\n”);
else
    printf(“F\n”);
```

Adapted from Deitel & Deitel, C How to Program, 3rd ed., p. 64
#include <stdio.h>

int main()
{
    int user_sel; /* variable for user menu choice */
    char line[100]; /* container for line of user input */

    /* Show the user the menu of choices, and ask for choice */
    printf("1. Play game\n"
        "2. Load game\n"
        "3. Play multiplayer\n"
        "4. Exit\n";
    "Enter your selection: ");

    /* Get user input - more robust method than scanf */
    fgets(line, sizeof(line), stdin);
    sscanf(line, "%d", &user_sel);

    switch (user_sel)
    {
    case 1: /* Note the colon, not a semicolon */
        printf("\nOk, let's play the game...\n");
        /* call game function here */
        break;

    case 2: /* Note the colon, not a semicolon */
        printf("\nOk, loading the game...\n");
        /* call load game function here */
        break;

    case 3: /* Note the colon, not a semicolon */
        printf("\nOk, multiplayer game...\n");
        /* call multiplayer game function here */
        break;

    case 4: /* Note the colon, not a semicolon */
        printf("\nOk, thank you for playing!\n");
        break;

    /* Always should have a default case for unexpected cases */
    default: /* Note the colon, not a semicolon */
        printf("Error, bad input, quitting\n";
        break;
    }
}
Fig. 4.8 | switch multiple-selection statement with breaks.

Adapted from Deitel & Deitel, C How to Program, 6th ed., p. 111
Review
References


Nesting selection structures

- Selection structures can be **stacked** and **nested** to handle more sophisticated decision/action functionality

```c
/* File: ifc.c */
#include <stdio.h>
int main ()
{
    int i;
    i = 10;
    if(i==2 || i == 4)
    {
        printf("i = 2 or 4\n");
    }
    else if(i == 10)
    {
        printf("i = 10\n");
    }
    else
    {
        printf("i = %d\n", i);
    }
    return 0;
}
```

Adapted from H. Cheng chap05.ppt, slide 12
<table>
<thead>
<tr>
<th>Operations</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>::</td>
<td></td>
</tr>
<tr>
<td>() []</td>
<td>left to right</td>
</tr>
<tr>
<td>Function_name()</td>
<td>right to left</td>
</tr>
<tr>
<td>. -&gt;</td>
<td>left to right</td>
</tr>
<tr>
<td><code> </code> ` ++ -- + - * &amp; (type) sizeof</td>
<td>right to left</td>
</tr>
<tr>
<td>* / % . * ./</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>left to right</td>
</tr>
<tr>
<td>== !=</td>
<td>left to right</td>
</tr>
<tr>
<td>&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td>^</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td>^^</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>? :</td>
<td>right to left</td>
</tr>
<tr>
<td>+= -= *= /= %=</td>
<td>= &lt;&lt;= &gt;&gt;=</td>
</tr>
<tr>
<td>,</td>
<td>left to right</td>
</tr>
</tbody>
</table>

Adapted from H. Cheng chap04.ppt, slide 5