C How to Program

Looping structures

Dr. mhd. Mazen Al Mustafa

Chapter 5– Looping structure

Outline

5.1 The While Repetition Structure

5.2 Formulating Algorithms: Case Study 1 (Counter-Controlled Repetition)

5.3 Formulating Algorithms with Top-down, Stepwise Refinement: Case Study 2 (Sentinel-Controlled Repetition)

5.4 Formulating Algorithms with Top-down, Stepwise Refinement: Case Study 3 (Nested Control Structures)

5.5 The Essentials of Repetition

5.6 Counter-Controlled Repetition

5.7 The For Repetition Structure

5.8 The For Structure: Notes and Observations

5.9 Examples Using the For Structure

5.10 The Do/While Repetition Structure

5.11 The break and continue Statements

5.1 The **`while`** Repetition Structure

- Repetition structure
  - Programmer specifies an action to be repeated while some condition remains **`true`**
  - Psuedocode:

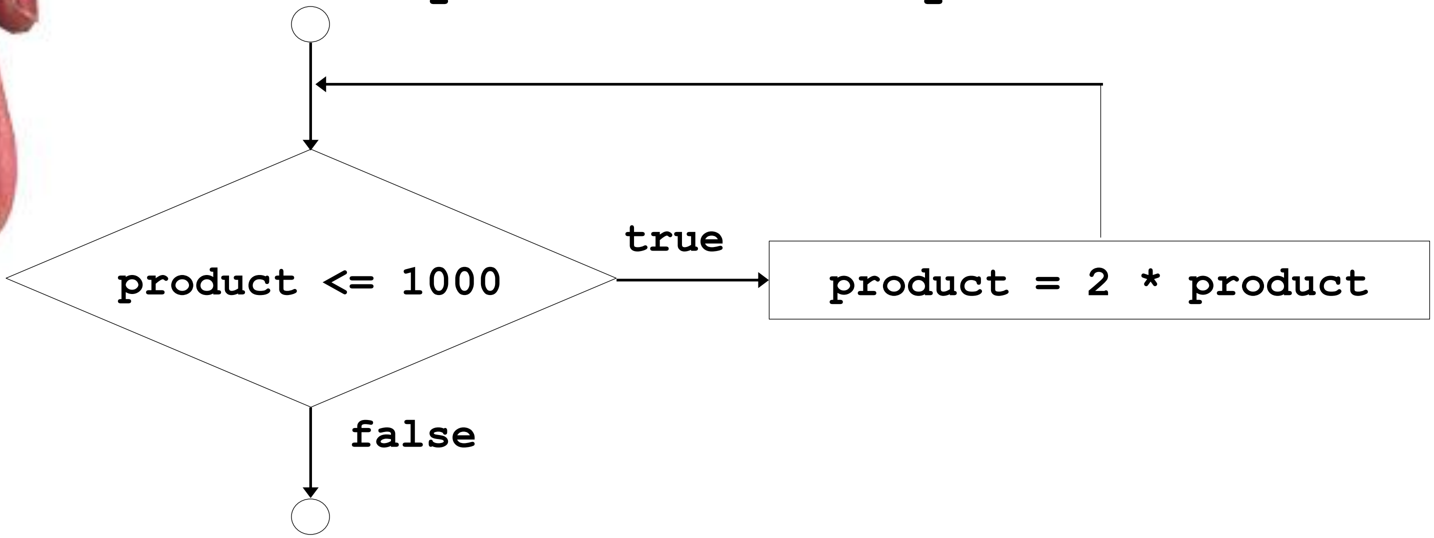
```
While there are more items on my shopping list
    Purchase next item and cross it off my list
```

  - **`while`** loop repeated until condition becomes **`false`**

5.1 The `while` Repetition Structure

- Example:

```
int product = 2;
while ( product <= 1000 )
     product = 2 * product;
```



5.2 Formulating Algorithms (Counter-Controlled Repetition)

- Counter-controlled repetition
  - Loop repeated until counter reaches a certain value
  - Definite repetition: number of repetitions is known
  - Example: A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz
  - Pseudocode:

```
Set total to zero
   Set grade counter to one

While grade counter is less than or equal to ten
   Input the next grade
   Add the grade into the total
   Add one to the grade counter

Set the class average to the total divided by ten
   Print the class average
```

```c
/* Fig. 3.6: fig03_06.c
   Class average program with
   counter-controlled repetition */
#include <stdio.h>

int main()
{
   int counter, grade, total, average;

   /* initialization phase */
   total = 0;
   counter = 1;

   /* processing phase */
   while ( counter <= 10 ) {
      printf( "Enter grade: " );
      scanf( "%d", &grade );
      total = total + grade;
      counter = counter + 1;
   }

   /* termination phase */
   average = total / 10;
   printf( "Class average is %d\n", average );

   return 0;   /* indicate program ended successfully */
}
```

1. Initialize Variables

2. Execute Loop

3. Output results

Program Output

```
Enter grade:  98
Enter grade:  76
Enter grade:  71
Enter grade:  87
Enter grade:  83
Enter grade:  90
Enter grade:  57
Enter grade:  79
Enter grade:  82
Enter grade:  94
Class average is 81
```

5.3 Formulating Algorithms with Top-Down, Stepwise Refinement

- Problem becomes:

  Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.

  - Unknown number of students
  - How will the program end?
- Use sentinel value
  - Also called signal value, dummy value, or flag value
  - Indicates “end of data entry.”
  - Loop ends when user inputs the sentinel value
  - Sentinel value chosen so it cannot be confused with a regular input (such as **`–1`** in this case)

5.3 Formulating Algorithms with Top-Down, Stepwise Refinement

- Top-down, stepwise refinement
  - Begin with a pseudocode representation of the *top*:

    Determine the class average for the quiz

  - Divide *top* into smaller tasks and list them in order:

    Initialize variables
    Input, sum and count the quiz grades
    Calculate and print the class average

- Many programs have three phases:
  - Initialization: initializes the program variables
  - Processing: inputs data values and adjusts program variables accordingly
  - Termination: calculates and prints the final results

5.3 Formulating Algorithms with Top-Down, Stepwise Refinement

- Refine the initialization phase from *Initialize variables* to:

```
Initialize total to zero
Initialize counter to zero
```

- Refine *Input, sum and count the quiz grades* to

```
Input the first grade (possibly the sentinel)
While the user has not as yet entered the sentinel
    Add this grade into the running total
    Add one to the grade counter
    Input the next grade (possibly the sentinel)
```

5.3 Formulating Algorithms with Top-Down, Stepwise Refinement

- Refine *Calculate and print the class average* to

```
If the counter is not equal to zero
    Set the average to the total divided by the counter
    Print the average
else
    Print “No grades were entered”
```

```c
/* Fig. 3.8: fig03_08.c
   Class average program with
   sentinel-controlled repetition */
#include <stdio.h>

int main()
{
   float average;             /* new data type */
   int counter, grade, total;

   /* initialization phase */
   total = 0;
   counter = 0;

   /* processing phase */
   printf( "Enter grade, -1 to end: " );
   scanf( "%d", &grade );

   while ( grade != -1 ) {
      total = total + grade;
      counter = counter + 1;
      printf( "Enter grade, -1 to end: " );
      scanf( "%d", &grade );
   }
```

1. Initialize Variables

2. Get user input

2.1 Perform Loop

```c

   /* termination phase */
   if ( counter != 0 ) {
      average = ( float ) total / counter;
      printf( "Class average is %.2f", average );
   }
   else
      printf( "No grades were entered\n" );

   return 0;   /* indicate program ended successfully */
}
```

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```

3. Calculate Average
   3.1 Print Results

Program Output

5.4 Nested control structures

- Problem
  - A college has a list of test results (**1** = pass, **2** = fail) for 10 students
  - Write a program that analyzes the results
    - If more than 8 students pass, print "Raise Tuition"
- Notice that
  - The program must process 10 test results
    - Counter-controlled loop will be used
  - Two counters can be used
    - One for number of passes, one for number of fails
  - Each test result is a number—either a **1** or a **2**
    - If the number is not a **1**, we assume that it is a **2**

5.4 Nested control structures

- Top level outline

    Analyze exam results and decide if tuition should be raised

- First Refinement

    Initialize variables

    Input the ten quiz grades and count passes and failures

    Print a summary of the exam results and decide if tuition should be raised

- Refine *Initialize variables* to

    Initialize passes to zero

    Initialize failures to zero

    Initialize student counter to one

5.4 Nested control structures

- Refine *Input the ten quiz grades and count passes and failures* to

```
While student counter is less than or equal to ten
   Input the next exam result
   If the student passed
      Add one to passes
   else
      Add one to failures
   Add one to student counter
```

- Refine *Print a summary of the exam results and decide if tuition should be raised* to

```
Print the number of passes
Print the number of failures
If more than eight students passed
   Print "Raise tuition"
```

```c
/* Fig. 3.10: fig03_10.c
   Analysis of examination results */
#include <stdio.h>

int main()
{
   /* initializing variables in declarations */
   int passes = 0, failures = 0, student = 1, result;

   /* process 10 students; counter-controlled loop */
   while ( student <= 10 ) {
      printf( "Enter result ( 1=pass,2=fail ): " );
      scanf( "%d", &result );

      if ( result == 1 )         /* if/else nested in while */
         passes = passes + 1;
      else
         failures = failures + 1;

      student = student + 1;
   }

   printf( "Passed %d\n", passes );
   printf( "Failed %d\n", failures );

   if ( passes > 8 )
      printf( "Raise tuition\n" );

   return 0;   /* successful termination */
}
```

1. Initialize variables

2. Input data and count passes/failures

3. Print results

Program Output

```
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 1
Enter Result (1=pass,2=fail): 2
Passed 6
Failed 4
```

5.5 The Essentials of Repetition

- Loop
  - Group of instructions computer executes repeatedly while some condition remains `true`
- Counter-controlled repetition
  - Definite repetition: know how many times loop will execute
  - Control variable used to count repetitions
- Sentinel-controlled repetition
  - Indefinite repetition
  - Used when number of repetitions not known
  - Sentinel value indicates "end of data"

5.6 Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires
  - The name of a control variable (or loop counter)
  - The initial value of the control variable
  - A condition that tests for the final value of the control variable (i.e., whether looping should continue)
  - An increment (or decrement) by which the control variable is modified each time through the loop

5.6 Essentials of Counter-Controlled Repetition

- Example:

```
int counter = 1;                // 
  initialization
while ( counter <= 10 ) { // repetition 
  condition
   printf( "%d\n", counter );
   ++counter;                // increment
}
```

  - The statement

```
int counter = 1;
```

    - Names **`counter`**
    - Declares it to be an integer
    - Reserves space for it in memory
    - Sets it to an initial value of **`1`**

5.7 The `for` Repetition Structure

- Format when using **`for`** loops

  `for` (*initialization*; *loopContinuationTest*; *increment*)
      statement

- Example:

```
for( int counter = 1; counter <= 10; counter++ )
   printf( "%d\n", counter );
```

  – Prints the integers from one to ten

No semicolon (`;`) after last expression

5.8 The `for` Repetition Structure

- For loops can usually be rewritten as while loops:

```
initialization;
while ( loopContinuationTest ) {
   statement;
   increment;
}
```

- Initialization and increment
  - Can be comma-separated lists
  - Example:

```
for (int i = 0, j = 0;  j + i <= 10; j++,
 i++)
   printf( "%d\n", j + i );
```

5.9 The **for** Structure: Notes and Observations

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions. If **x** equals **2** and **y** equals **10**

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to

```
for ( j = 2; j <= 80; j += 5 )
```

- Notes about the **for** structure:
  - "Increment" may be negative (decrement)
  - If the loop continuation condition is initially **false**
    - The body of the **for** structure is not performed
    - Control proceeds with the next statement after the **for** structure
  - Control variable
    - Often printed or used inside "for” body, but not necessary

```c
/* Fig. 4.5: fig04_05.c
   Summation with for */
#include <stdio.h>

int main()
{
   int sum = 0, number;

   for ( number = 2; number <= 100; number += 2 )
      sum += number;

   printf( "Sum is %d\n", sum );

   return 0;
}
```

```
Sum is 2550
```

1. Initialize variables
2. `for` repetition structure

5.10 The **do/while** Repetition Structure

- The **do/while** repetition structure
  - Similar to the **while** structure
  - Condition for repetition tested after the body of the loop is performed
    - All actions are performed at least once
  - Format:

```
do {
   statement;
} while ( condition );
```

5.10 The do/while Repetition Structure

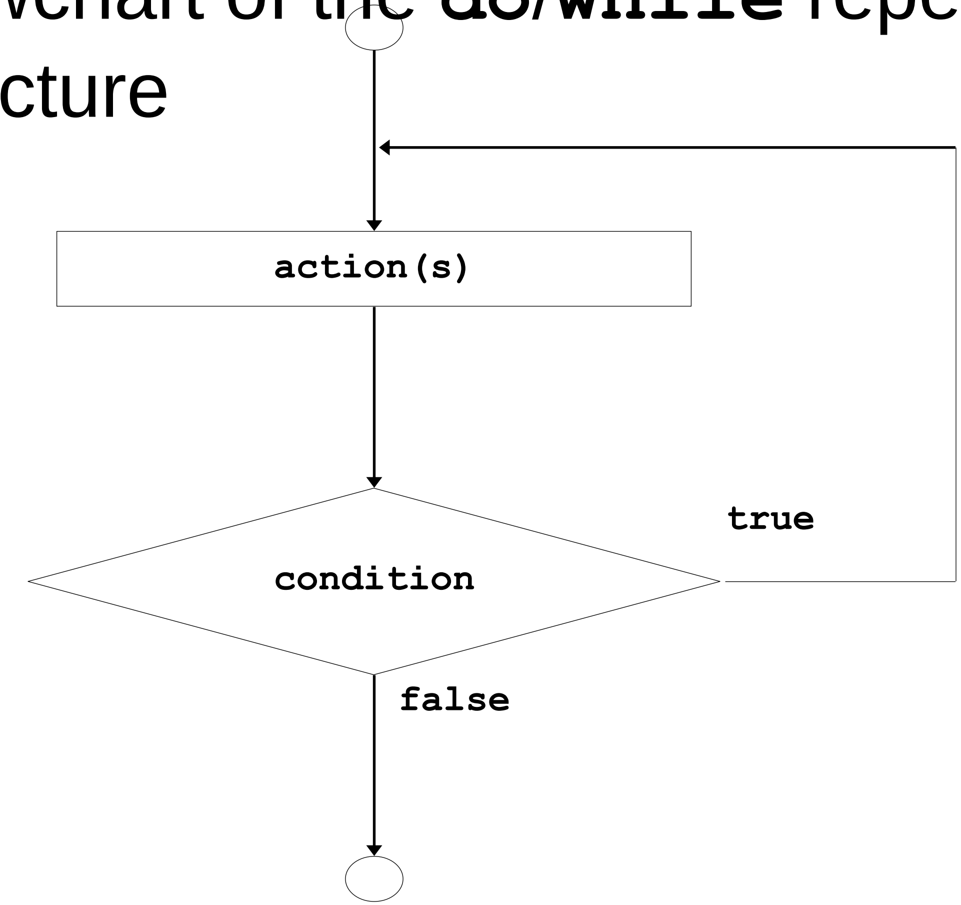
- Example (counter = 1):

```
do {
   printf( "%d  ", counter );
} while (++counter <= 10);
```

  - Prints the integers from 1 to 10

5.10 The `do/while` Repetition Structure

- Flowchart of the **`do/while`** repetition structure



```c
/* Fig. 4.9: fig04_09.c
   Using the do/while repetition structure */
#include <stdio.h>

int main()
{
   int counter = 1;

   do {
      printf( "%d  ", counter );
   } while ( ++counter <= 10 );

   return 0;
}
```

```
1  2  3  4  5  6  7  8  9  10
```

1. Initialize variable

2. Loop

3. Print

5.11 The `break` and `continue` Statements

- **`break`**
  - Causes immediate exit from a **`while`**, **`for`**, **`do/while`** or **`switch`** structure
  - Program execution continues with the first statement after the structure
  - Common usage of the **`break`** statement
    - Escape early from a loop
    - Skip the remainder of a **`switch`** structure

5.11 The `break` and `continue` Statements

- **`continue`**
  - Skips the remaining statements in the body of a **`while`**, **`for`** or **`do/while`** structure
    - Proceeds with the next iteration of the loop
  - **`while`** and **`do/while`**
    - Loop-continuation test is evaluated immediately after the **`continue`** statement is executed
  - **`for`**
    - Increment expression is executed, then the loop-continuation test is evaluated

```c
/* Fig. 4.12: fig04_12.c
   Using the continue statement in a for structure */
#include <stdio.h>

int main()
{
   int x;

   for ( x = 1; x <= 10; x++ ) {

      if ( x == 5 )
         continue;  /* skip remaining code in loop only
                       if x == 5 */

      printf( "%d ", x );
   }

   printf( "\nUsed continue to skip printing the value 5\n" );
   return 0;
}
```

1. Initialize variable

2. Loop

3. Print

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5
```

```c
/* Fig. 4.7: fig04_07.c
   Counting letter grades */
#include <stdio.h>

int main()
{
   int grade;
   int aCount = 0, bCount = 0, cCount = 0,
       dCount = 0, fCount = 0;

   printf(  "Enter the letter grades.\n"  );
   printf(  "Enter the EOF character to end input.\n"  );

   while ( ( grade = getchar() ) != EOF ) {

      switch ( grade ) {    /* switch nested in while */

         case 'A': case 'a':  /* grade was uppercase A */
            ++aCount;         /* or lowercase a */
            break;

         case 'B': case 'b':  /* grade was uppercase B */
            ++bCount;         /* or lowercase b */
            break;

         case 'C': case 'c':  /* grade was uppercase C */
            ++cCount;         /* or lowercase c */
            break;

         case 'D': case 'd':  /* grade was uppercase D */
            ++dCount;         /* or lowercase d */
            break;
```

1. Initialize variables

2. Input data

2.1 Use switch loop to update count

```c

         case 'F': case 'f':  /* grade was uppercase F */
            ++fCount;         /* or lowercase f */
            break;

         case '\n': case' ':  /* ignore these in input */
            break;

         default:        /* catch all other characters */
            printf( "Incorrect letter grade entered." );
            printf( " Enter a new grade.\n" );
            break;
      }
   }

   printf( "\nTotals for each letter grade are:\n" );
   printf( "A: %d\n", aCount );
   printf( "B: %d\n", bCount );
   printf( "C: %d\n", cCount );
   printf( "D: %d\n", dCount );
   printf( "F: %d\n", fCount );

   return 0;
}
```

2.1 Use switch loop to update count

3. Print results

Program Output

```
Enter the letter grades.
Enter the EOF character to end input.
A
B
C
C
A
D
F
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
B

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```