Chapter 3
Structured Program Development

Associate Prof. Yuh-Shyan Chen
Dept. of Computer Science and Information Engineering
National Chung-Cheng University

Outline

3.1 Introduction
3.2 Algorithms
3.3 Pseudocode
3.4 Control Structures
3.5 The If Selection Structure
3.6 The If/Else Selection Structure
3.7 The While Repetition Structure
3.8 Formulating Algorithms: Case Study 1 (Counter-Controlled Repetition)
3.9 Formulating Algorithms with Top-down, Stepwise Refinement: Case Study 2 (Sentinel-Controlled Repetition)
3.10 Formulating Algorithms with Top-down, Stepwise Refinement: Case Study 3 (Nested Control Structures)
3.11 Assignment Operators
3.12 Increment and Decrement Operators

3.1 Introduction

Before writing a program:
- Have a thorough understanding of problem
- Carefully planned approach for solving it

While writing a program:
- Know what “building blocks” are available
- Use good programming principles

3.2 Algorithms

Computing problems
- All can be solved by executing a series of actions in a specific order

Algorithm: procedure in terms of
- Actions to be executed
- Order in which these actions are to be executed

Program control
- Specify order in which statements are executed
3.3 Pseudocode

- Pseudocode
  - Artificial, informal language that helps us develop algorithms
  - Similar to everyday English
  - Not actually executed on computers
  - Helps us “think out” a program before writing it
    - Easy to convert into a corresponding C/C++ program
    - Consists only of executable statements

```c
int i ; /* Declarations are not executable statements */
```

3.4 Control Structures

- Sequential execution
  - Statements executed one after the other in the order written

- Transfer of control
  - When the next statement executed is not the next one in sequence
    - Overuse of `goto` led to many problems.
  - So called “structured programming”
    - “goto elimination”

Bohm and Jacopini

- Demonstrated that programs could be written without any `goto` statements
- Goto-less programming
- All programs written in terms of three control structures
  - Sequence structure: Built into C. Programs executed sequentially by default.
  - Selection structures: C has three types - `if`, `if/else`, and `switch`.
  - Repetition structures: C has three types - `while`, `do/while`, and `for`.
- These are C keywords

3.4 Control Structures (II)

- Flowchart
  - Graphical representation of an algorithm
    - Drawn using certain special-purpose symbols connected by arrows called `flowlines`.
    - Rectangle symbol (action symbol): indicates any type of action.
    - Oval symbol: indicates beginning or end of a program, or a section of code (circles).
  - Single-entry/single-exit control structures
    - Connect exit point of one control structure to entry point of the next (control-structure stacking).
    - Makes programs easy to build
The simplest flowchart

Repeatedly applying rule 2 (any rectangle can be replaced by two rectangles in sequence)

3.5 The if Selection Structure

- Selection structure:
  - Used to choose among alternative courses of action
  - Pseudocode: If student’s grade is greater than or equal to 60
    Print “Passed”
  - If condition true:
    - Print statement executed and program goes on to next statement.
    - If false, print statement is ignored and the program goes on to the next statement.
    - Indenting makes programs easier to read
  - C ignores whitespace characters.
  - Pseudocode statement in C:
    if (grade >= 60)
    printf( "Passed\n" );
  - C code corresponds closely to the pseudocode
3.5 The if Selection Structure (II)

- Diamond symbol (decision symbol) - indicates decision is to be made.
- Contains an expression that can be true or false.
- Test the condition, follow appropriate path.
- If structure is a single-entry/single-exit structure.

![Decision Structure Diagram]

Example:

```plaintext
grade >= 60
true: print "Passed"
false: print "Failed"
```

3.6 The if/else Selection Structure

- Only performs an action if the condition is true.
- A different action when condition is false.

Psuedocode:

```
if student's grade is greater than or equal to 60
    Print "Passed"
else
    Print "Failed"
```

C code:

```
if ( grade >= 60 )
    printf( "Passed\n" );
else
    printf( "Failed\n" );
```

3.6 The if/else Selection Structure (II)

- Ternary conditional operator (?:)

```
Takes three arguments (condition, value if true, value if false).
Our pseudocode could be written:
printf( "%s\n", grade >= 60 ? "Passed" : "Failed" );
```

```
grade >= 60 ? printf("Passed\n") : printf("Failed\n")
```

3.6 The if/else Selection Structure (III)

- Nested if/else structures
- Test for multiple cases by placing if/else selection structures inside if/else selection structures.

C code:

```
if ( student's grade is greater than or equal to 90 )
    printf( "A" );
else if ( student's grade is greater than or equal to 80 )
    printf( "B" );
else if ( student's grade is greater than or equal to 70 )
    printf( "C" );
else if ( student's grade is greater than or equal to 60 )
    printf( "D" );
else
    printf( "F" );
```

Once condition is met, rest of statements skipped.
Deep indentation usually not used in practice.
3.6 The if/else Selection Structure (IV)

- **Compound statement:**
  - Set of statements within a pair of braces
  - Example:
    ```c
    if ( grade >= 60 )
        printf( "Passed.\n" );
    else {
        printf( "Failed.\n" );
        printf( "You must take this course again.\n" );
    }
    
    Without the braces,
    printf( "You must take this course again.\n" );
    would be automatically executed

- **Block:** compound statements with declarations

Written by C

```c
if ( grade >= 90)
    printf ("A\n");
else
    if (grade >= 80)
        printf("B\n");
    else
        if (grade>= 60)
            printf("D \n");
        else
            printf("F\n");
```
3.7 The while Repetition Structure

- Repetition structure
  - Programmer specifies an action to be repeated while some condition remains true

- Pseudocode: 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

3.7 The while Repetition Structure (II)

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

3.8 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
3.9 Formulating Algorithms with Top-Down, Stepwise Refinement (Sentinel-Controlled Repetition)

- **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 know to end?
- **Use sentinel (哨兵) value**
  - Also called signal value, dummy value, or flag value
  - Indicates "end of data entry."
  - Loop ends when sentinel inputted
  - Sentinel value chosen so it cannot be confused with a regular input (such as -1 in this case)
- **Sentinel-controlled repetition is often called as indefinite repetition**

### 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
```

3.9 Formulating Algorithms with Top-Down, Stepwise Refinement (Sentinel-Controlled Repetition) (II)

- **Top-down, stepwise refinement**
  - Is essential to the development of well-structured programs
  - Begin with a pseudocode representation of the top:
    - Describe the class average for the quiz
  - (first refinement) 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
      - This Helps the breakup of programs for top-down refinement
3.9 Formulating Algorithms with Top-Down, Stepwise Refinement (III)

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)

Refine Calculate and print the class average to:

Set the average to the total divided by the counter
Print the average
else
Print “No grades were entered”

Class average program with sentinel-controlled repetition

```c
#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 );
  }

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

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

3.10 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
3.10 Nested control structures (II)

- **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

---

3.10 Nested control structures (III)

- **Refine**
  Input the ten quiz grades and count passes and failures
  While student counter is less than or equal to ten
    - 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
  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
", passes );
  printf( "Failed %d
", failures );

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

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

---

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
3.11 Assignment Operators

- Assignment operators abbreviate assignment expressions
  
  \[ c = c + 3; \]
  
  can be abbreviated as \[ c += 3; \]
  
  using the addition assignment operator.

- Statements of the form
  
  \[ variable = variable \text{ operator} \text{ expression}; \]
  
  can be rewritten as

  \[ variable \text{ operator} = \text{expression}; \]

- Examples of other assignment operators:
  
  \[
  \begin{align*}
  &d -= 4 \quad (d = d - 4) \\
  &e *= 5 \quad (e = e * 5) \\
  &f /= 3 \quad (f = f / 3) \\
  &g %= 9 \quad (g = g % 9)
  \end{align*}
  \]

3.12 Increment and Decrement Operators

- Increment operator \([++]\) - can be used instead of \([c += 1]\)
- Decrement operator \([--]\) - can be used instead of \([c -= 1]\).

Preincrement

- Operator is used before the variable \([++c] \) or \([--c]\)
- Variable is changed, then the expression it is in is evaluated

Postincrement

- Operator is used after the variable \([c++] \) or \([c--]\)
- Expression executes, then the variable is changed

- If \(c = 5\), then
  
  \[
  \begin{align*}
  &\text{printf}("%d", ++c); \\
  &\text{printf}("%d", c++);
  \end{align*}
  \]
- In either case, \(c\) now has the value of 6

3.12 Increment and Decrement Operators (II)

- When variable not in an expression
  
  Preincrementing and postincrementing have the same effect.

  \[
  \begin{align*}
  &+e; \\
  &cout << c; \\
  \end{align*}
  \]
  
  and

  \[
  c++; \\
  \text{cout} << \text{c};
  \]
  
  have the same effect.

Precedence and Associativity of the Operators (Fig. 3.14)
Example of Precedence and Associativity

x += y <= z ? ++ (a+b) * 4: c -- % 5 ;