An Introduction to Programming with C++

Fifth Edition

Chapter 7

The Repetition Structure

Objectives

- Include the repetition structure in pseudocode
- Include the repetition structure in a flowchart
- Code a pretest loop using the C++ `while` statement
- Initialize and update counters and accumulators
- Code a pretest loop using the C++ `for` statement

Concept Lesson

- Using the Repetition Structure
- Pretest Loops
- Using the `while` Statement to Code a Pretest Loop

Concept Lesson (continued)

- Using Counters and Accumulators
- Counter-Controlled Pretest Loops
- Using the `for` Statement to Code a Pretest Loop

Using the Repetition Structure

- **Repetition structure** repeatedly processes (a) program instruction(s) until a condition is met
  - Also called **loop**
  - **Pretest loop** (also called top-driven) ← Most common type
    - Condition evaluated *before* instructions are processed
      - Instructions may never be processed
  - **Posttest loop** (also called bottom-driven)
    - Condition evaluated *after* instructions are processed
      - Instructions within loop are always processed

Pretest Loops

- Not every problem requires a loop in its solution
- Most loops have a condition and a body
  - **Loop condition** determines the number of times instructions within loop are processed
    - In pretest loops, condition appears at the beginning
    - Must result in true or false value only

Pretest Loops (continued)

Problem description

At the beginning of each year, the president of Acme Hardware is paid a 10% bonus. The bonus is based on the amount of sales made by the company during the previous year. The payroll clerk wants a program that calculates and displays the bonus amount.

Input	Processing	Output
bonus rate (10%) sales	Processing items: none Algorithm: 1. enter the sales 2. bonus = sales * bonus rate 3. display the bonus	bonus

Figure 7-1: Problem description and IPO chart for Acme Hardware

Pretest Loops (continued)

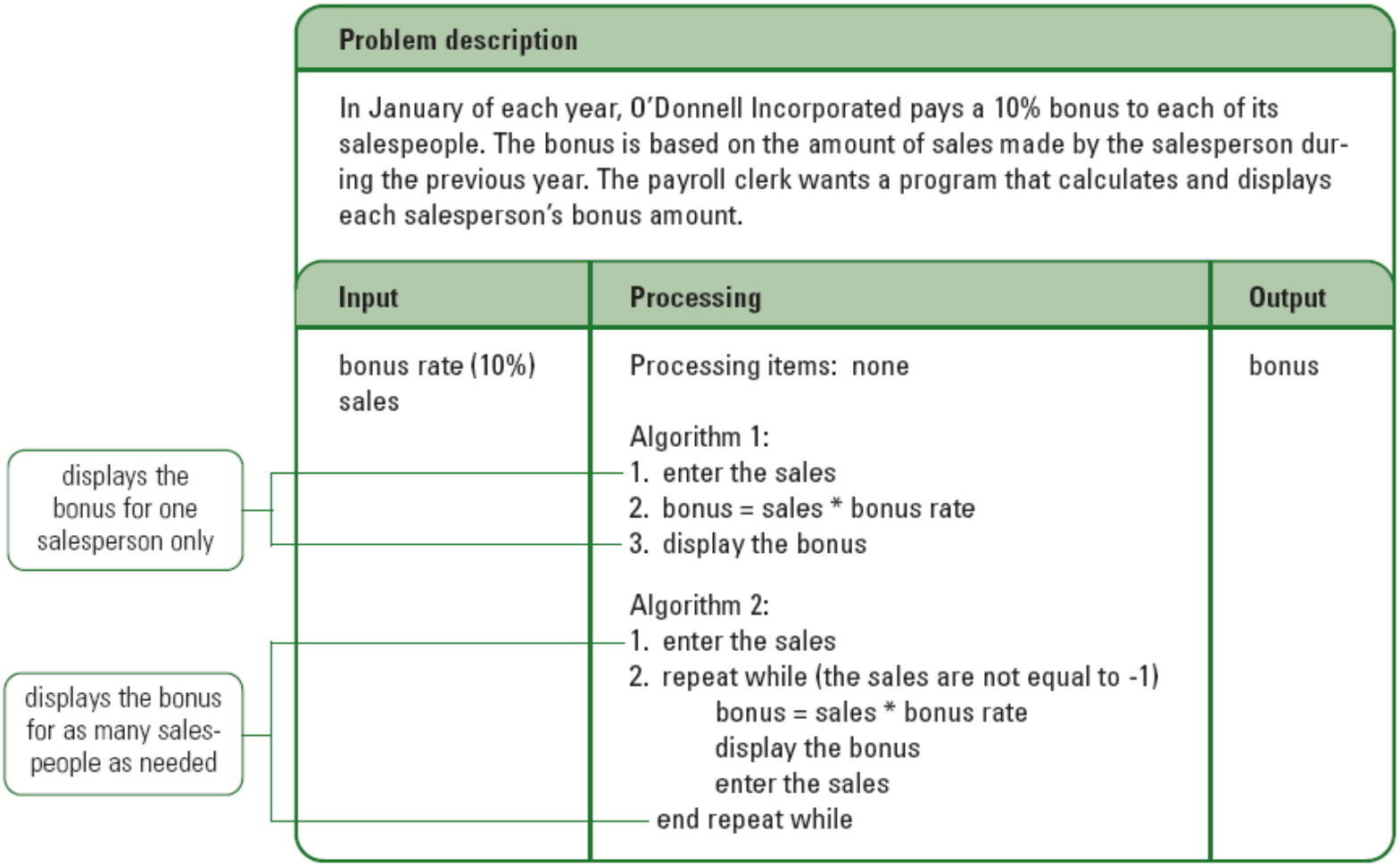


Figure 7-2: Problem description and IPO chart for O'Donnell Incorporated

Pretest Loops (continued)

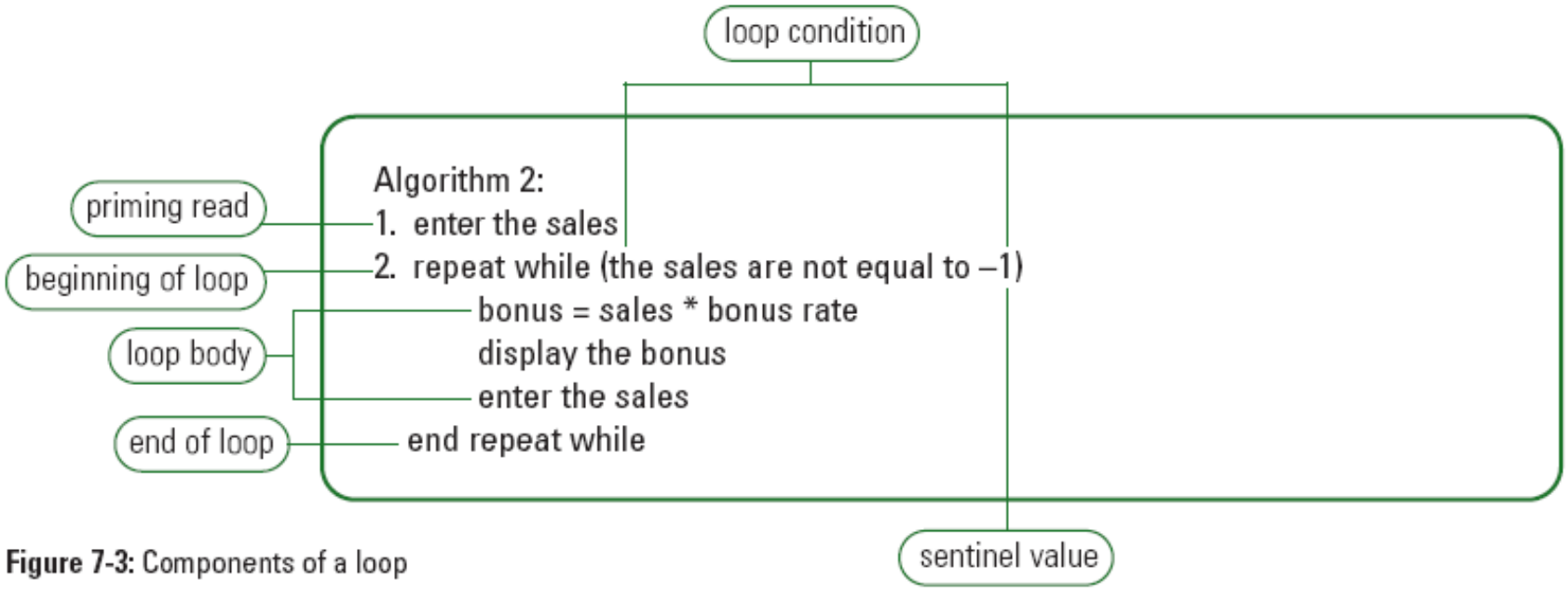


Figure 7-3: Components of a loop

Flowcharting a Pretest Loop

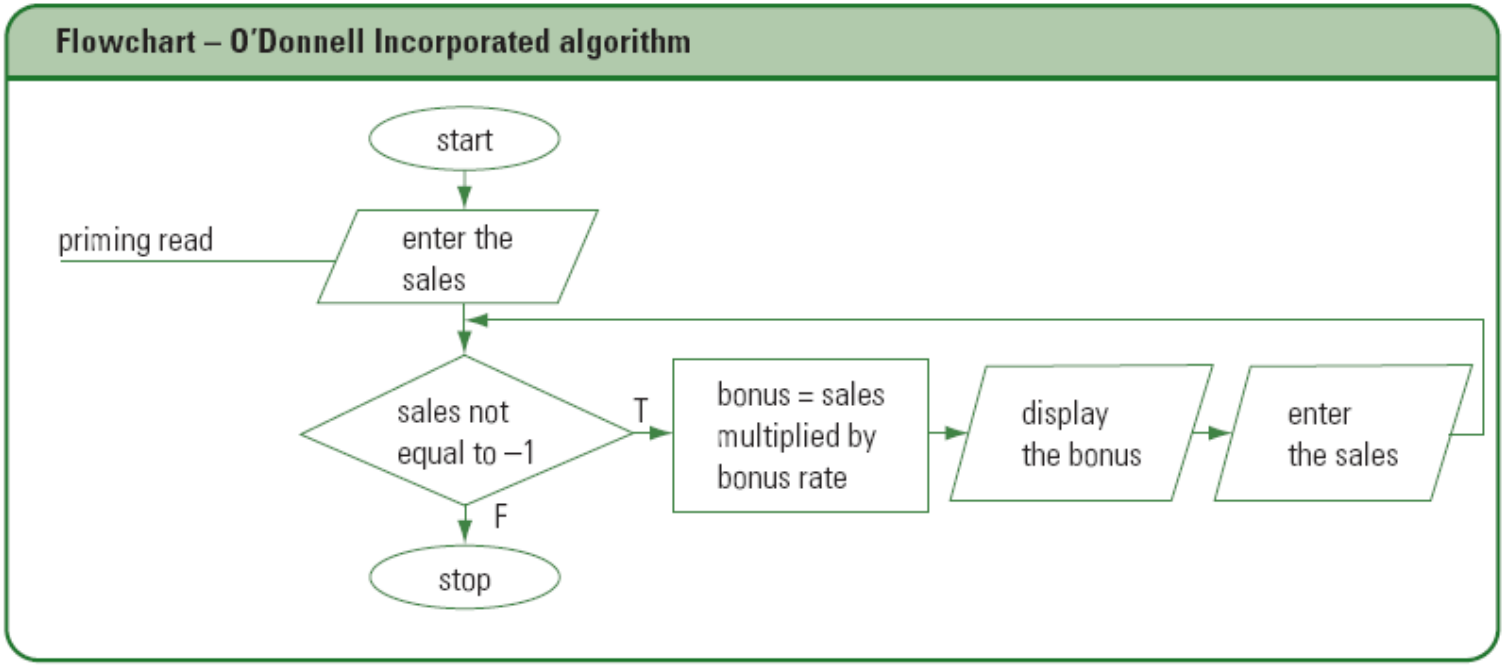


Figure 7-4: O'Donnell Incorporated algorithm shown in flowchart form

Flowcharting a Pretest Loop (continued)

bonus rate	sales	bonus
.10	10000	

Figure 7-5: First sales amount recorded in the desk-check table

bonus rate	sales	bonus
.10	10000	1000

Figure 7-6: First salesperson's bonus information recorded in the desk-check table

bonus rate	sales	bonus
.10	~~10000~~ 25000	~~1000~~ 2500

Figure 7-7: Second salesperson's bonus information recorded in the desk-check table

bonus rate	sales	bonus
.10	~~10000~~ ~~25000~~ –1	~~1000~~ 2500

Figure 7-8: Sentinel value recorded in the desk-check table

Using the `while` Statement to Code a Pretest Loop

Use the `while` Statement

Syntax

`while` (*loop condition*)
    one statement, or a statement block, to be processed as long as the loop condition evaluates to true
`//end while`

Example 1

```
int age = 0;
cout << "Enter age: ";
cin >> age;
while (age > 0)
{
     cout << "Age: " << age << endl;
     cout << "Enter age: ";
     cin >> age;
}    //end while
```

Figure 7-9: How to use the `while` statement

Using the `while` Statement to Code a Pretest Loop (continued)

Example 2

```
string name = "";
cout << "Name: ";
getline(cin, name);
while (name != "Y" && name != "y")
{
     cout << name << endl;
     cout << "Name: ";
     getline(cin, name);
}    //end while
```

Figure 7-9: How to use the `while` statement *(Continued)*

Pretest Loop Example – O'Donnell Incorporated Program

IPO chart information	C++ instructions
Input bonus rate (10%) sales	`const double RATE = .1;` `double sales = 0.0;`
Processing none	
Output bonus	`double bonus = 0.0;`

Figure 7-10: IPO chart information and C++ code for the O'Donnell Incorporated program

Pretest Loop Example – O'Donnell Incorporated Program (continued)

IPO chart information	C++ instructions
Algorithm 1. enter the sales	`cout << "First sales amount: ";` `cin >> sales;`
2. repeat while (the sales are not equal to –1)	`while (sales != -1.0)` `{`
bonus = sales * bonus rate	`bonus = sales * RATE;`
display the bonus	`cout << "Bonus: $" << bonus;` `cout << endl << endl;`
enter the sales	`cout << "Next sales amount: ";` `cin >> sales;`
end repeat while	`} //end while`

Figure 7-10: IPO chart information and C++ code for the O'Donnell Incorporated program *(Continued)*

Forgetting this statement turns loop into an **endless** or **infinite loop**

Pretest Loop Example – O'Donnell Incorporated Program (continued)

Figure 7-11: Sample run of the O'Donnell Incorporated program

Using Counters and Accumulators

- **Counter:** variable used for counting something
  - E.g., number of employees paid in a week
- **Accumulator:** variable used for accumulating
  - E.g., total dollar amount of a week's payroll
- **Initializing:** assigning a beginning value to counter or accumulator
  - Typically zero
- **Updating** (**incrementing**)**:** adding a number to counter or accumulator

Counter and Accumulator Examples – Sales Express Program

IPO chart information	C++ instructions
Input	
sales	`double sales = 0.0;`
Processing	
number of salespeople (counter)	`int totalPeople = 0;`
total sales (accumulator)	`double totalSales = 0.0;`

Figure 7-12: IPO chart information and C++ code for the Sales Express program

Counter and Accumulator Examples – Sales Express Program (continued)

IPO chart information	C++ instructions
Output average sales	`double average = 0.0;`
Algorithm 1. enter the sales	`cout << "First sales amount: ";` `cin >> sales;`
2. repeat while (the sales are greater than or equal to 0)	`while (sales >= 0.0)` `{`
add 1 to the number of salespeople	`totalPeople = totalPeople + 1;`
add the sales to the total sales	`totalSales = totalSales + sales;`
enter the sales	`cout << "Next sales amount: ";` `cin >> sales;`
end repeat while	`} //end while`
3. if (the number of salespeople is greater than 0)	`if (totalPeople > 0)` `{`
average sales = total sales / number of salespeople	`average = totalSales /` `static_cast<double>(totalPeople);`
display the average sales	`cout << "Average: " << average << endl;`
else	`}` `else`
display an error message	`cout << "No sales entered." << endl;`
end if	`//end if`

Figure 7-12: IPO chart information and C++ code for the Sales Express program *(Continued)*

Counter and Accumulator Examples – Sales Express Program (continued)

sales	totalPeople	totalSales	average
~~0.0~~ 30000.0	0	0.0	0.0

Figure 7-13: First sales amount recorded in the desk-check table

sales	totalPeople	totalSales	average
~~0.0~~ 30000.0	~~0~~ 1	~~0.0~~ 30000.0	0.0

Figure 7-14: Desk-check table showing the first update to the counter and accumulator variables

sales	totalPeople	totalSales	average
~~0.0~~ ~~30000.0~~ 40000.0	~~0~~ ~~1~~ 2	~~0.0~~ ~~30000.0~~ 70000.0	0.0

Figure 7-15: Desk-check table showing the second update to the counter and accumulator variables

sales	totalPeople	totalSales	average
~~0.0~~ ~~30000.0~~ ~~40000.0~~ –3	~~0~~ ~~1~~ 2	~~0.0~~ ~~30000.0~~ 70000.0	~~0.0~~ 35000.0

Figure 7-16: Completed desk-check table for the Sales Express program

Counter and Accumulator Examples – Sales Express Program (continued)

Figure 7-17: Sample run of the Sales Express program

Counter-Controlled Pretest Loops

- In previous programs, user controls loop termination by entering a sentinel value
- Program can also control the termination of loop
  - Typically, done using a counter
    - Counter-controlled loop
    - Example: Jasper Music Company program

Counter-Controlled Loop Example – Jasper Music Company Program

IPO chart information	C++ instructions
Input region's quarterly sales	`int regionSales = 0;`
Processing counter (1 through 3)	`int region = 1;`
Output total quarterly sales (accumulator)	`int totalSales = 0;`

Algorithm
1. repeat while (the counter is less than or equal to 3)
enter the region's quarterly sales
	`while (region <= 3)`
	`{`
	`cout << "Enter Region " << region << "'s quarterly sales: ";`
	`cin >> regionSales;`

add the region's quarterly sales to the total quarterly sales	`totalSales = totalSales + regionSales;`
add 1 to the counter end repeat while	`region = region + 1;` `} //end while`
2. display the total quarterly sales	`cout << "Total: " << totalSales << endl;`

Figure 7-18: IPO chart information and C++ code for the Jasper Music Company program

Counter-Controlled Loop Example – Jasper Music Company Program (continued)

regionSales	region	totalSales
0	1	0

Figure 7-19: Desk-check table showing initialization of variables

regionSales	region	totalSales
~~0~~ 2500	~~1~~ 2	~~0~~ 2500

Figure 7-20: Results of processing the loop instructions the first time

regionSales	region	totalSales
~~0~~ ~~2500~~ 6000	~~1~~ ~~2~~ 3	~~0~~ ~~2500~~ 8500

Figure 7-21: Results of processing the loop instructions the second time

regionSales	region	totalSales
~~0~~ ~~2500~~ ~~6000~~ 2000	~~1~~ ~~2~~ ~~3~~ 4	~~0~~ ~~2500~~ ~~8500~~ 10500

Figure 7-22: Results of processing the loop instructions the third time

Counter-Controlled Loop Example – Jasper Music Company Program (continued)

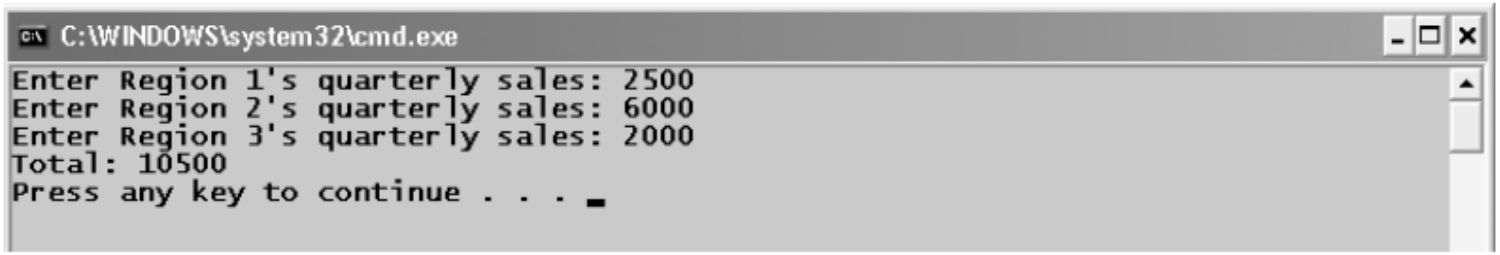


Figure 7-23: Sample run of the Jasper Music Company program

Using the `for` Statement to Code a Pretest Loop

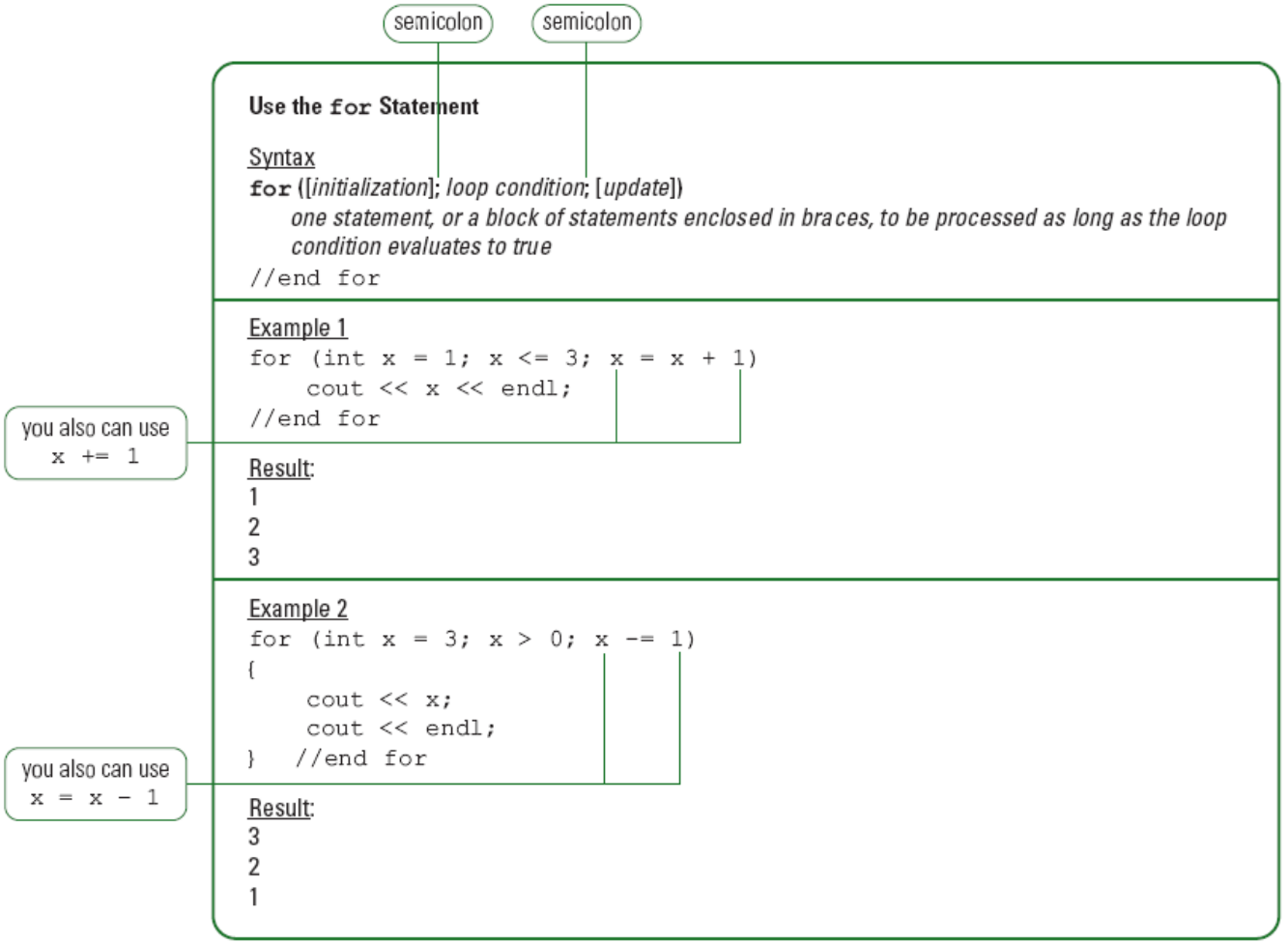


Figure 7-24: How to use the `for` statement

Using the `for` Statement to Code a Pretest Loop (continued)

Processing steps for Example 1 in Figure 7-24

1. The *initialization* argument (`int x = 1`) tells the computer to create a counter variable named `x` and initialize it to the number 1.
2. The *loop condition* argument (`x <= 3`) tells the computer to check whether the value in the `x` variable is less than or equal to 3. It is, so the computer processes the loop body instruction.
3. The loop body instruction tells the computer to display the `x` variable's value on the screen. In this case, it displays the number 1.
4. The *update* argument tells the computer to add the number 1 to the contents of the `x` variable, giving 2.
5. The *loop condition* argument (`x <= 3`) tells the computer to check whether the value in the `x` variable is less than or equal to 3. It is, so the computer processes the loop body instruction.
6. The loop body instruction tells the computer to display the `x` variable's value on the screen. In this case, it displays the number 2.
7. The *update* argument tells the computer to add the number 1 to the contents of the `x` variable, giving 3.
8. The *loop condition* argument (`x <= 3`) tells the computer to check whether the value in the `x` variable is less than or equal to 3. It is, so the computer processes the loop body instruction.
9. The loop body instruction tells the computer to display the `x` variable's value on the screen. In this case, it displays the number 3.
10. The *update* argument tells the computer to add the number 1 to the contents of the `x` variable, giving 4.
11. The *loop condition* argument (`x <= 3`) tells the computer to check whether the value in the `x` variable is less than or equal to 3. It's not, so the computer stops processing the `for` loop. Processing continues with the statement following the end of the loop.

Figure 7-25: Processing steps for the code shown in Example 1 in Figure 7-24

Using the `for` Statement to Code a Pretest Loop (continued)

Pseudocode

```
repeat for x = 1 to 3
    display x's value
end repeat for
```

Figure 7-26: Pseudocode for Example 1 in Figure 7-24

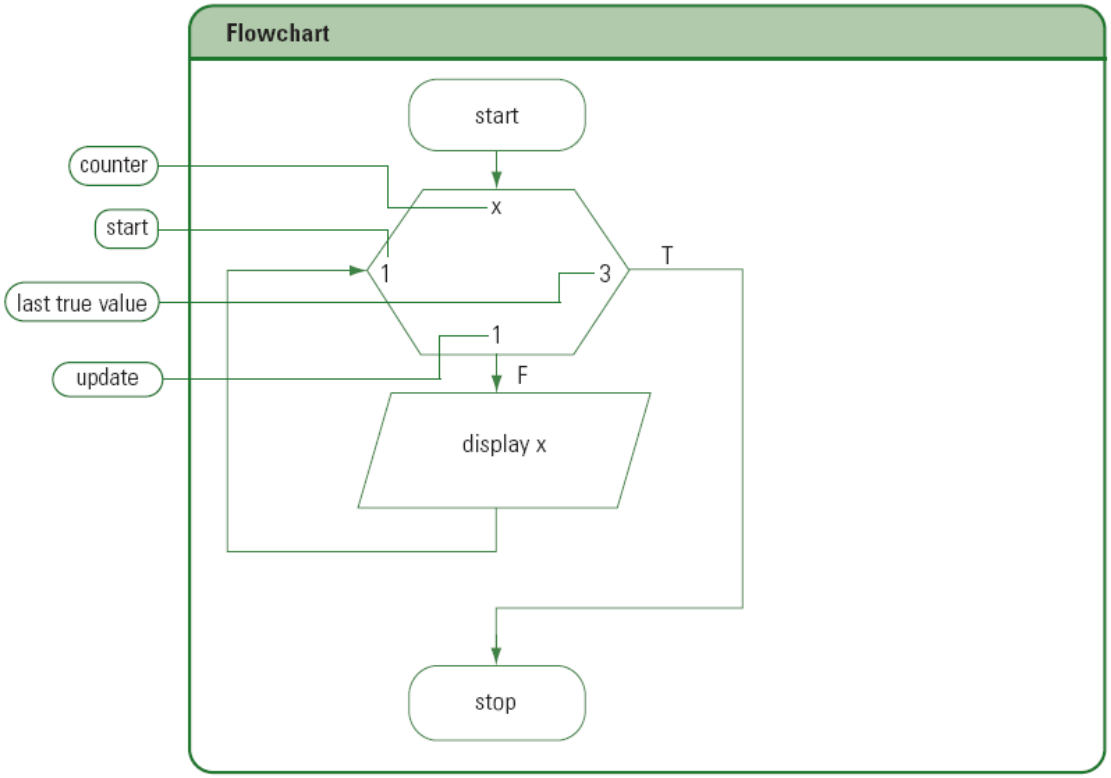


Figure 7-27: Flowchart for Example 1 in Figure 7-24

`for` Statement Example 1 – Holmes Supply Program

C++ code – Holmes Supply program

```
int main()
{
     int storeGross = 0;
     int totalGross = 0;

     for (int store = 1; store <= 3; store = store + 1)
     {
          cout << "Store " << store << " gross: ";
          cin >> storeGross;
          totalGross = totalGross + storeGross;
     } //end for

     cout << "Total: " << totalGross << endl;

     return 0;
}    //end of main function
```

Figure 7-28: C++ code for the `main()` function in the Holmes Supply program

for Statement Example 1 – Holmes Supply Program (continued)

storeGross	totalGross	store
0	0	1

Figure 7-29: Desk-check table showing the result of processing the *initialization* argument

storeGross	totalGross	store
~~0~~ 15000	~~0~~ 15000	~~1~~ 2

Figure 7-30: Desk-check table showing the result of processing the *update* argument for the first time

storeGross	totalGross	store
~~0~~ ~~15000~~ 25000	~~0~~ ~~15000~~ 40000	~~1~~ ~~2~~ 3

Figure 7-31: Desk-check table showing the result of processing the *update* argument for the second time

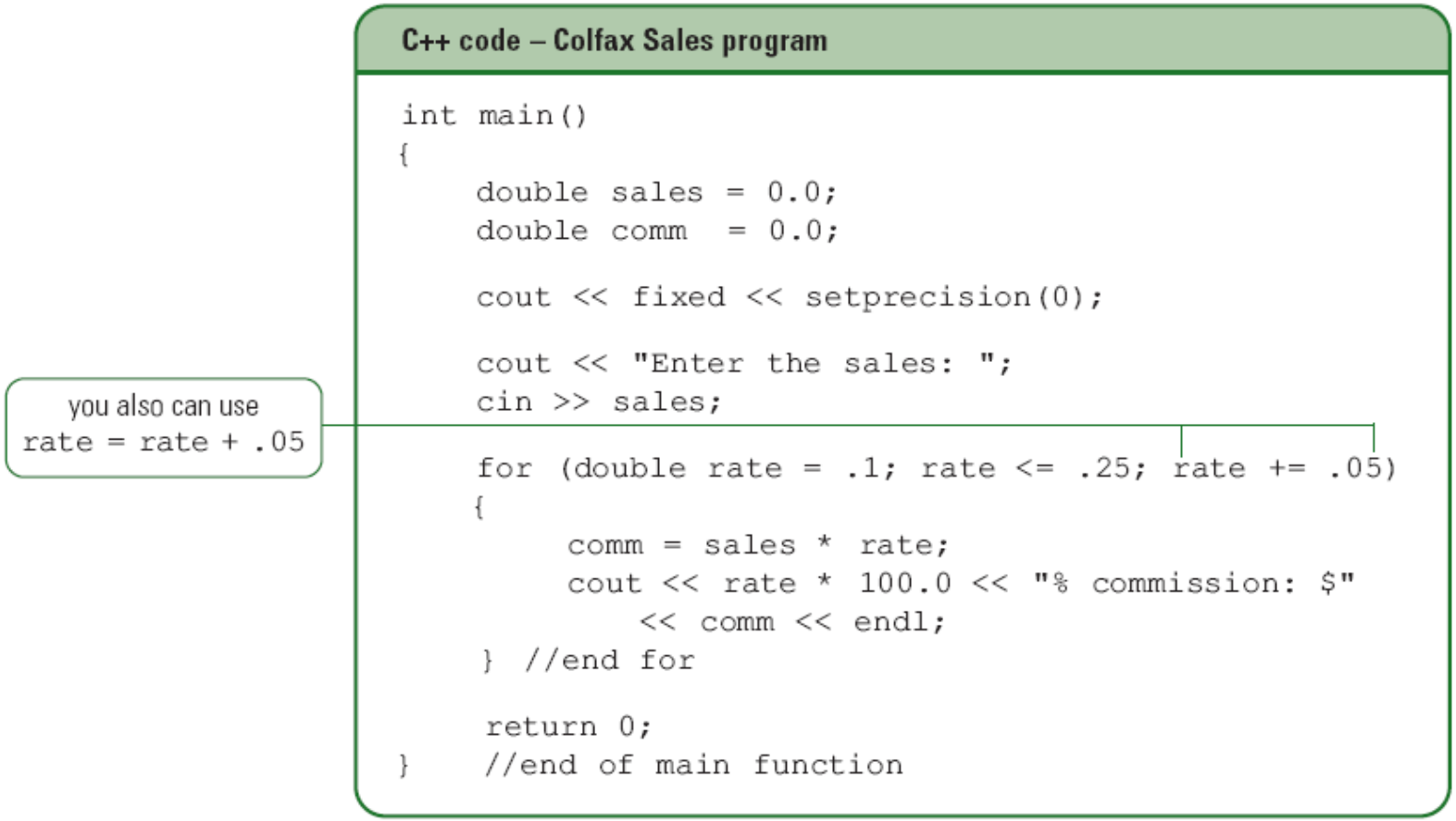
storeGross	totalGross	store
~~0~~ ~~15000~~ ~~25000~~ 60000	~~0~~ ~~15000~~ ~~40000~~ 100000	~~1~~ ~~2~~ ~~3~~ 4

Figure 7-32: Desk-check table showing the result of processing the *update* argument for the third time

`for` Statement Example 1 – Holmes Supply Program (continued)

Figure 7-33: Sample run of the Holmes Supply program

`for` Statement Example 2 – Colfax Sales Program



C++ code – Colfax Sales program

```
int main()
{
     double sales = 0.0;
     double comm  = 0.0;

     cout << fixed << setprecision(0);

     cout << "Enter the sales: ";
     cin >> sales;

     for (double rate = .1; rate <= .25; rate += .05)
     {
          comm = sales * rate;
          cout << rate * 100.0 << "% commission: $"
               << comm << endl;
     } //end for

     return 0;
}    //end of main function
```

Figure 7-34: C++ code for the `main()` function in the Colfax Sales program

`for` Statement Example 2 – Colfax Sales Program (continued)

Processing steps

1. Computer creates the `sales` variable and initializes it to 0.0.
2. Computer creates the `comm` variable and initializes it to 0.0.
3. Computer sets the appropriate format for floating-point numbers.
4. Computer prompts the user for the sales amount.
5. Computer stores the user's response (25000) in the `sales` variable.
6. Computer creates the `rate` variable and initializes it to .1 (*initialization* code).
7. Computer determines whether the `rate` variable's value is less than or equal to .25 (*loop condition* code). It is.
8. Computer calculates the commission and displays "10% commission: $2500" on the screen (loop body instructions).
9. Computer adds .05 to the value stored in the `rate` variable, giving .15 (*update* code).
10. Computer determines whether the `rate` variable's value is less than or equal to .25 (*loop condition* code). It is.
11. Computer calculates the commission and displays "15% commission: $3750" on the screen (loop body instructions).
12. Computer adds .05 to the value stored in the `rate` variable, giving .2 (*update* code).
13. Computer determines whether the `rate` variable's value is less than or equal to .25 (*loop condition* code). It is.
14. Computer calculates the commission and displays "20% commission: $5000" on the screen (loop body instructions).
15. Computer adds .05 to value stored in the `rate` variable, giving .25 (*update* code).
16. Computer determines whether the `rate` variable's value is less than or equal to .25 (*loop condition* code). It is.
17. Computer calculates the commission and displays "25% commission: $6250" on the screen (loop body instructions).
18. Computer adds .05 to the value stored in the `rate` variable, giving .3 (*update* code).
19. Computer determines whether the `rate` variable's value is less than or equal to .25 (*loop condition* code). It is not.
20. Loop ends and the computer processes the instruction following the end of the `for` statement.

Figure 7-35: Processing steps for the code shown in Figure 7-34 *(Continued)*

`for` Statement Example 2 – Colfax Sales Program (continued)

Figure 7-36: Sample run of the Colfax Sales program

`for` Statement Example 3 – Morgan Company Program

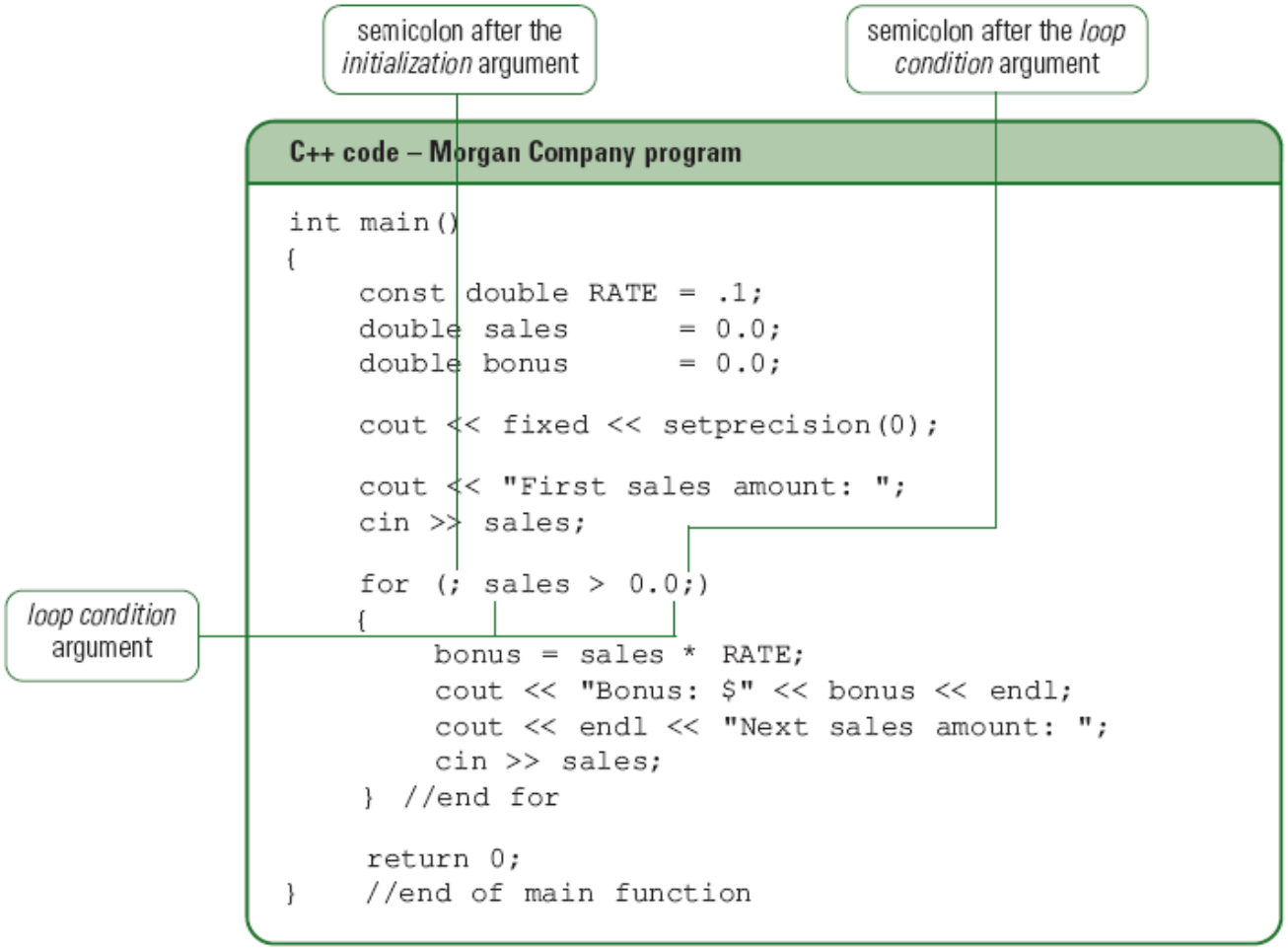


Figure 7-37: C++ code for the `main()` function in the Morgan Company program

`for` Statement Example 3 – Morgan Company Program (continued)

Processing steps

1. Computer creates the `RATE` constant and initializes it to .1.
2. Computer creates the `sales` variable and initializes it to 0.0.
3. Computer creates the `bonus` variable and initializes it to 0.0.
4. Computer sets the appropriate format for floating-point numbers.
5. Computer prompts the user for the sales amount.
6. Computer stores the user's response (4000) in the `sales` variable.
7. Computer determines whether the `sales` variable's value is greater than 0.0 (*loop condition* code). It is.
8. Computer calculates the bonus and displays "Bonus: $400" on the screen. It then prompts the user for another sales amount and stores the user's response (3000) in the `sales` variable (loop body instructions).
9. Computer determines whether the `sales` variable's value is greater than 0.0 (*loop condition* code). It is.
10. Computer calculates the bonus and displays "Bonus: $300" on the screen. It then prompts the user for another sales amount and stores the user's response (–1) in the `sales` variable (loop body instructions).
11. Computer determines whether the `sales` variable's value is greater than 0.0 (*loop condition* code). It is not.
12. Loop ends and the computer processes the instruction following the end of the `for` statement.

Figure 7-38: Processing steps for the code shown in Figure 7-37 *(Continued)*

for Statement Example 3 – Morgan Company Program (continued)

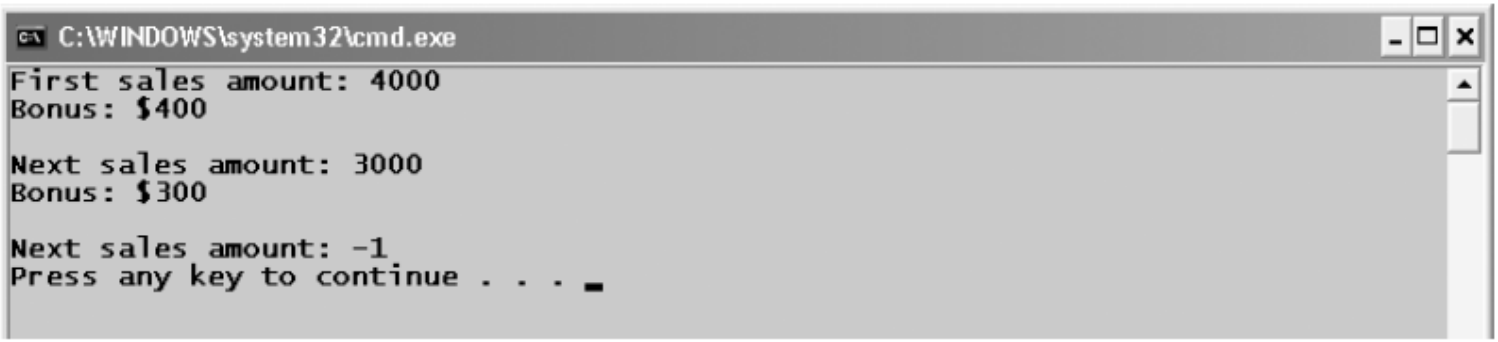


Figure 7-39: Sample run of the Morgan Company program

Summary

- Use repetition structure (loop) to repeatedly process program instruction(s) until condition is met
  - Pretest or posttest
  - Flowchart symbol is repetition/selection diamond
- Most loops have a condition and a body
- Sentinel value may be entered by user to end loop
- Priming read: input instruction above a pretest loop
- Counters and accumulators within loops used to calculate subtotals, totals, and averages

Application Lesson: Using the Repetition Structure in a C++ Program

- Lab 7.1: Stop and Analyze
- Lab 7.2
  - Create, test and verify the Professor Krelina program
- Lab 7.3
  - Modify the program created in Lab 7.2
    - Should allow user to enter four project scores and two test scores only
- Lab 7.4: Desk-Check Lab
- Lab 7.5: Debugging Lab