Tutorial letter 201/1/2012

Introduction to Programming II

COS1512

Semester 1

School of Computing

Solution to Assignment 1

Bar code

CONTENTS

1 INTRODUCTION

The purpose of this tutorial letter is to supply the solution for Assignment 1, and to indicate how you should interpret the results you obtained for this assignment. This assignment covers the work discussed in Chapters 4, 5, 6 and 9 of the study guide (Appendix F in Tutorial Letter 101), as well as the relevant sections in Chapters 4, 5, 6 and 9 of Savitch. Note that you should have included the input and output of all programs you were required to write.

The assessment and feedback given on your assignment, serve as an indication of your mastering of the study material. If you received a good mark, you can be confident that you are on the right track. If you did not receive a good mark, it is an indication that you need to revise your study method for COS1512.

CD	Prescribed software
COS1512/**101**/3/2012	First tutorial letter: General information, study programme, exam admission and assignments, software installation instructions, solution to assignment 3, study guide
COS1512/**102**/1/2012	Invitation to Workshop
COS1512/**201**/1/2012	This letter: Solution to Assignment 1

2. TUTORIAL MATTER DISTRIBUTED TO DATE

If you have not received all of the above-mentioned tutorial matter, please contact our DESPATCH DEPARTMENT:

Email: despatch@unisa.ac.za or
inf@unisa.ac.za (for international students)
SMS number: 43584
Phone number: (011) 471-3249/2248.

Or for any non-academic or registration queries:

Contact: Mr. Maswika Molepo
Email: molepmm@unisa.ac.za
Phone Number: (012)429 3111

Note that all the COS1512 tutorial letters will also be available from *myUnisa* under Official Study Material.

3 ALLOCATION OF MARKS

When we mark assignments, we comment on your answers. Many students make the same mistakes and consequently we discuss general problems in the tutorial letters. It is, therefore, important to work through the tutorial letters and to make sure you understand our solutions and where you went wrong.

The maximum number of marks you could obtain for Assignment 1 is 50. This is converted to a percentage. If you for instance obtained 25 marks for Assignment 1, you received 50% for Assignment 1. This percentage in turn contributes a weight of 20% to the year mark, as can be seen in the summary of the weights allocated to the assignments for COS1512 below.

Assignment number	Weight
1	20%
2	80%
3	0%

Question 2 has not been marked. However, 5 marks were awarded if you *attempted* this question (note that this is not the way in which questions will be marked in the examination!). We include complete solutions for *all* questions.

The marks you received for question 1 were determined on the following basis:

Question not done	0/5
Question attempted, but the program does not work at all	2/5
A good attempt, but there are a few problems with your answer	4/5
The program works correctly and produces the correct output	5/5

The marks you received for questions 3 and 4 were determined on the following basis:

Question not done	0/15
Question attempted, but the program does not work at all	5/15
A good attempt, but there are a few problems with your answer	12/15
The program works correctly and produces the correct output	15/15

In other words, you can obtain a maximum of 5 marks for question 1, and maximum of 15 marks for question 3 and 4.

For question 5 you can obtain a maximum of 10 marks:

- ½ mark for each of the questions 5(a)(i – viii) (4 marks in total)
- for question 5(b), ½ mark for each correct diagram that corresponds to the body of the program, and 2 marks for the output. (6 marks in total)

Note that not all mistakes are *corrected* – but we will provide informative comments.

If you did not include the program output for question 1, it means there are “a few problems with your answer” and the maximum you can get is then 4/5. If you did not include the program output for questions 3 and 4, it means there are “a few problems with your answer” and the maximum you can get is then 12/15.

We did not award any marks to assignments submitted more than four weeks after the due date. However, we still provided informative comments.

4 SOLUTION TO ASSIGNMENT

Question 1 [5 marks]

Function overloading is the practice of declaring the same function with different signatures. This means that the same function name will be used with different numbers of parameters and/or parameters of different types. *But overloading of functions with different* ***return*** *types is not allowed.* In this question you had to write an overloaded function named `calcSalary` with either one parameter of type `char` or two parameters of type `char` and `int` respectively. Although it was not specified in the question, our program displays only two digits after the decimal point.

Program listing

```
//Ass 1 question 1
#include <iostream>
using namespace std;
const double PERM_PER_HOUR_SALES = 30.70;
const double PERM_PER_HOUR_STOCK = 28.50;
const double TEMP_PER_HOUR_MERCH = 17.90;
const double TEMP_PER_HOUR_COUNT = 16.80;
const int PERM_HOURS = 192;

//overloaded function for calculation of permanent employee salary
//Pre: type of duties
//Post: returned value is monthly salary
double calcSalary (char duties);

//overloaded function for calculation of temporary employee salary
//Pre: type of duties and number of hours
//Post: returned value is monthly salary
double calcSalary (char duties, int hours);

int main()
{
    char answer;
    int hours;
    double salary;

    do
    {
        cout << "Permanent or temporary worker - enter 'p' or 't':" << endl;
        cin >> answer;
    }while (answer != 'p' && answer != 't');

    if (answer == 'p')
    {
        do
        {
          cout << "Please key in duties - enter 's' or 'c':" << endl;
          cin >> answer;
```

```
        }while(answer!= 's' && answer != 'c');

            salary = calcSalary (answer);
    }
    else if (answer == 't')
    {
        do
        {
          cout << "Please key in duties - enter 'm' or 'c':" << endl;
          cin >> answer;

        }while(answer!= 'm' && answer != 'c');
        do
        {
          cout << "Does the employee work 60 or 102 hours?:" << endl;
          cin >> hours;
        }while(hours!= 60 && hours != 102);

         salary = calcSalary (answer, hours);
    }

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << endl <<"The monthly salary is R" << salary << endl;
    return 0;
}

//overloaded function permanent employee salary
double calcSalary (char duties)
{
     if (duties == 's')
          return (PERM_PER_HOUR_SALES * PERM_HOURS);
     else
          return (PERM_PER_HOUR_STOCK * PERM_HOURS);
}

//overloaded function temporary employee salary
double calcSalary (char duties, int hours)
{
     if (duties == 'm')
          return (TEMP_PER_HOUR_MERCH * hours);
     else
          return (TEMP_PER_HOUR_COUNT * hours);
}
```

Input and corresponding output version 1:

```
Permanent or temporary worker - enter 'p' or 't':
p
Please key in duties - enter 's' or 'c':
c

The monthly salary is R5472.00
Press any key to continue . . .
```

Input and corresponding output version 2:

```
Permanent or temporary worker - enter 'p' or 't':
t
Please key in duties - enter 'm' or 'c':
m
Does the employee work 60 or 102 hours?:
102

The monthly salary is R1825.80
Press any key to continue . . .
```

Question 2 [5 marks- This question has not been marked. A max of 5 marks was awarded if you *attempted* this question]

In this question you had to change two dates (mm yyyy) to months, ensure that both dates are valid, and calculate the difference between the two dates in terms of number of months, in order to calculate what was required in the question. We used the `assert` macro as follows to insure that the dates are valid:

```
assert(month1 >= 1 && month1 <= 12);
assert(year1 >= 1975 && year1 <= 2010);
```

The same applies for the second date:

```
assert(month2 >= 1 && month2 <= 12);
assert(year2 >= 1975 && year2 <= 2010);
```

Also note how we make sure that if `year1` and `year2` are the same year, that `month1` must then be less than `month2`:

```
if (year1 == year2)
        assert(month1 < month2);
    else
        assert(year1 < year2);
```

Note the definition of `assert()` that had to be included:

```
#include <cassert>
```

We expected you to check the validity of the dates with the `assert` macro. This is of course also possible without the `assert` macro.

Typically the `assert` macro is used to identify program errors during development. The argument given to `assert` should be chosen so that it holds true only if the program is operating as intended. The macro evaluates the `assert` argument and, if the argument expression is false the program execution is halted. No action is taken if the argument is true.

Program listing

```
//Ass 1 question 2
#include <iostream>
#include <cassert>
using namespace std;

int calcMonths(int month, int year) //convert a date to a number of months
{
    int totalmonths = month;
    totalmonths += year * 12;
    return totalmonths;
}
int main()
{
    int month1, month2, year1, year2;
    int bonus = 0, yearsService, monthsService;
    int difference, totMonth1, totMonth2;
    cout << "Key in the start date (e.g. 7 1986) : " ;
    cin >> month1 >> year1;
    assert(month1 >= 1 && month1 <= 12);
    assert(year1 >= 1975 && year1 <= 2010);
    totMonth1 = calcMonths(month1, year1);
    cout << endl<< "Key in the end date(e.g. 3 1999) : " ;
    cin >> month2 >> year2;
    if (year1 == year2)
        assert(month1 < month2);
    else
        assert(year1 < year2);
    assert(month2 >= 1 && month2 <= 12);
    assert(year2 >= 1975 && year2 <= 2010);
    totMonth2 = calcMonths(month2, year2);
    //calculate total nr of months that employee was in service
    difference = totMonth2 - totMonth1;
    //convert service in months back to years and months for display
    yearsService = difference / 12;
    monthsService = difference % 12;
    //calculate bonus - employee gets R500 for first 60 months
    if (difference >= 60){
       bonus += 500;
       difference -= 60;
    }
```

```
    //and R5 for each month thereafter
    bonus += difference * 5;
    cout << "The employee was in service for " << yearsService << " year(s) and
    “ << monthsService << " month(s) " << endl;
    cout << "The bonus payable is R" << bonus << endl;

        return 0;
}
```

Input and corresponding output version 1:

```
Key in the start date (e.g. 7 1986) : 3  1999

Key in the end date(e.g. 3 1999) : 8 2013
Assertion failed: year2 >= 1975 && year2 <= 2010, file
C:\unisa\cos1512\q2_s1.cpp, line 28

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.
```

Input and corresponding output version 2:

```
Key in the start date (e.g. 7 1986) : 3 1976

Key in the end date(e.g. 3 1999) : 8 2010
The employee was in service for 34 year(s) and 5 month(s)
The bonus payable is R2265
Press any key to continue . . .
```

Input and corresponding output version 3:

```
Key in the start date (e.g. 7 1986) : 3 2010

Key in the end date(e.g. 3 1999) : 6 2010
The employee was in service for 0 year(s) and 3 month(s)
The bonus payable is R15
Press any key to continue . . .
```

Question 3 [15 marks]

Discussion:

The first step is to create the input file. We created the input file `precedence.dat` by using the DevC++ editor and creating a new source file, entering the data, and saving it as a file with an extension of `.dat`. (You could also have used Notepad.) Save your input file in the same directory where you save your program.

The header file for the `fstream` library has to be added to the code with the `#include <fstream>` directive for file processing.

Though the question does not specify that the output should appear on the console window, we include code to display the contents of the output file. The output file can also be viewed with DevC++. This file is opened with `File | Open Project or File` and by selecting the correct file in the correct directory.

We open the input file `precedence.dat` and the output file, `results.dat`, by associating the file names with the `ifstream` and `outstream` variables, as shown in the statements below:

```
string outName = "results.dat";
string inName = "precedence.dat";

infile.open(inName.c_str());
outfile.open(outName.c_str());

if (!infile)
{
    cout << "Cannot open file " << inName << " Aborting!" << endl;
    exit(1);
}
if (!outfile)
{
   cout << "Cannot open file " << outName << " Aborting!" << endl;
   exit(1);
}
```

Note that if you create the input file in a directory different from the one where you save your program, you need to specify the path as well, when specifying the filename, e.g.

```
C:\unisa\cos112\datafiles\precedence.dat
```

When using a file it is essential to test whether the file is available. The following code segment tests whether the first input file is available before attempting to extract from it:

```
if (!infile)
{
    cout << "Cannot open file " << inName << " Aborting!" << endl;
    exit(1);
}
```

In the above code segment the program is terminated immediately when `exit(1)` is invoked. We need to add

```
#include <cstdlib>
```

to our program i.e. the header file for the library that contains the `exit()` function.

The function with prototype `void process(ifstream& infile, ofstream& outfile)` is then called. This function reads the data from the input file line by line, calculating the value of y in the following assignment statement:

```
y = 14 * 9 – 8 / 2 + 10 / 3 – 8 + 4;
```

The program uses the combination number as the parameter for the `switch` statement. The result is then written to the output file `results.dat` as was requested in the question.

Note that when file streams are passed as arguments to a function, all the input- and output-file streams need to be reference parameters for the function, since they will all be modified, as can be seen from the prototype given above. The file streams should be passed by reference rather than by value since the internal state of a file stream object may change with an open operation even if the contents of the file have not changed. When you declare a reference parameter, the function call will pass the *memory address* of the actual parameter, instead of copying the parameter value into the formal parameter.

The next step is to close the output file. It has originally been opened as an output file. However, we now want to read it to see if the data was written correctly, which means it now has to serve as an input file, and we have to open it as an `ifstream`:

```
outfile.close();
ifstream inout;
inout.open(outName.c_str());

if (!inout)
{
   cout << "Cannot open file " << outName << " Aborting!" << endl;
  exit(1);
 }
```

Program Listing:

```
#include <iostream> // for screen/keyboard i/o
#include <fstream>  // for file
#include <string>   //for string manipulation
#include <cstdlib> // for exit, assert

using namespace std;

//void process(ifstream& infile)
//Pre: input file is open
//     input file is a data file containing operator preference selections
//Post:records of output file are displayed

//This function is written to test that the contents of results.dat is in
//fact what we intend it to be.
void checkFile(ifstream& infile)
{
   cout << endl <<"Displaying contents of results.dat : "<< endl <<endl;
   while(infile)
   {
      string temp;   //temporary string to hold records of output file
      getline(infile, temp, '\n');
      cout << temp <<endl;
   }
}
```

```
//This function opens the data file precedence.dat containing the precedence
//selection of each member in the group

void process(ifstream& infile, ofstream& outfile)
{
    string name ;       //name of team member
    int combo;          //combination of precedence chosen
    char oper1;         //read operator with highest precedence
    char oper2;         //read operator with 2nd highest precedence
    char oper3;         //read operator with 2nd lowest precedence
    char oper4;         //read operator with lowest precedence
    int y = 0;

    outfile.setf(ios::fixed);
    outfile.precision(2);

    while(infile >> name >> combo >> oper1 >> oper2 >> oper3 >> oper4)
    {
        switch (combo)
        {
           case 1:
                y = (14 * 9) - ((8 / 2) + (10 / 3)) - (8 + 4);
                break;
           case 2:
                y = (14 * ((9 - (8 / 2))) + ((10 / 3) - 8) + 4);
                break;
           case 3:
                y = 14 * (9 - ((8 / 2) + (10 / 3)) - (8 + 4));
                break;
           case 4:
                y = (((14 * 9) - 8) / 2) + (10 / (3 - 8)) + 4;
                break;
           case 7:
                y = (14 * (9 - 8)) /( 2 + 10) / ((3 - 8) + 4);
                break;
           case 8:
                y = 14 * ((9 - 8) / (2 + 10) / (3 - (8 + 4)));
                break;
           default:
              cout << "Other combinations not developed yet" << endl;
        }
        outfile << name << ' ' << oper1 << ' ' << oper2 << ' ';
        outfile  << oper3 << ' ' << oper4 << " y = " << y << endl;
        }
}

int main()
{
   ifstream infile;
```

```
   ifstream inout;
   ofstream outfile;
   string outName = "results.dat";
   string inName = "precedence.dat";

   infile.open(inName.c_str());
   outfile.open(outName.c_str());

   if (!infile)
   {
       cout << "Cannot open file " << inName << " Aborting!" << endl;
       exit(1);
   }
   if (!outfile)
   {
      cout << "Cannot open file " << outName << " Aborting!" << endl;
      exit(1);
   }
   process(infile, outfile);

   outfile.close();
   infile.close();

   inout.open(outName.c_str());

   if (!inout)
   {
      cout << "Cannot open file " << outName << " Aborting!" << endl;
      exit(1);
   }

   checkFile(inout);
   inout.close();
   return 0;
}
```

Input and corresponding output :

```
Displaying contents of results.dat:

Peter / + - * y = -140
Dahne + - / * y = 0
Renee - * + / y = -1
Charl * / + - y = 107
Johan / - * + y = 56
Sipho * - / + y = 61

Press any key to continue . . .
```

Question 4 [15 marks]

Discussion:
The purpose of the program is to read a file character by character. The file contains a nursery rhyme. Your task was to ignore the leading blanks in each line, and write an output file with all lines left justified and all characters in upper case.

We did not write a function to perform this task – it is performed by the main function. We did, however write a function to read and display the input file, as well as the output file that was created from the input file. It is good programming practice to put the code that does all the reading and writing together.

When implementing our solution, once again the first step is to create the input file. Since the purpose of the program is to process files, the `#include <fstream>` and `#include <cstdlib>` directives have to be included in the program in order to use the files and to test that they exist before they are used. The names of the input and output files are requested from the user.

In this program we process the input file as a text file. A text file is typically processed in the following way:

```
char ch;
infile.get(ch);
while (!infile.eof())
{
        //process ch
        infile.get(ch);
}
```

Compare this to the typical way to read a file containing values that should be processed as `int`, `string` or `float` (as in Question 3), e.g. a file containing one `int` followed by a string on each line:

```
int value;
string name;
while (infile >> value >> name)
{
     //process value and name
}
```

After having created the output file, we added some code to read the input file and display the contents, as well as some code to do the same for the output file. Note that after the output file is created and closed, it now acts as an input file if we want to read and display its contents. We therefore need an `ifstream` definition. For this purpose we defined:

```
ifstream indisplay;
ifstream outdisplay;
```

The first declaration will represent the original input file. The second declaration will represent the created output file as an input file.

Program listing:

```
//Ass 1 question 4
#include <iostream> // for screen/keyboard i/o
```

```
#include <fstream> // for file
#include <cstdlib> // for exit

using namespace std;

// Precondition:
// The input file is a text file.
// The input file has been opened.
//
// Postcondition:
// The output file is a text file.
// The output file has been opened.
// Output file will contain all lines left justified
// and all characters in upper case

//This function displays the output file created

void checkFile(ifstream& infile)
{
   while(infile)
   {
      string temp;   //temp string to hold lines of previously created

      //output file
      getline(infile, temp, '\n');
      cout << temp <<endl;
    }
}

int main()
{

  ifstream infile;
  ofstream outfile;
  ifstream indisplay;
  ifstream outdisplay;

  string inName, outName;
  bool init_blanks = true; //will get rid of all initial blanks while true

  cout << endl << "Enter the input file name. " << endl;
  cin >> inName;
  cout << endl << "Enter the output file name. " << endl
       << "WARNING: ANY EXISTING FILE WITH THIS NAME WILL"
       <<" BE ERASED." << endl;
  cin >> outName;

  infile.open(inName.c_str());
  if (infile.fail())
```

```
{
    cout << "Cannot open file "
         << inName << " Aborting!" << endl;
    exit(1);
}

outfile.open(outName.c_str());
if (outfile.fail())
{
    cout << "Cannot open file "
         << outName << " Aborting!" << endl;
    exit(1);
}

char ch;
infile.get(ch);
ch = toupper(ch);

while(!infile.eof())
{

    if (ch == '\n')
    {
       outfile << endl;
       init_blanks = true;
    }
    else
    {
       if (ch != ' ')
       {
          init_blanks = false;  //As soon as the first non-blank character
          outfile << ch;        //has been read, there cannot be leading
       }                        //blanks anymore
       else
       {
          if (ch == ' ' && (!init_blanks)) //blank character only gets
              outfile << ch;               //written if not a leading blank
       }                                   //character
    }
    infile.get(ch);
    ch = toupper(ch);
}       //end while !infile.eof

infile.close();
outfile.close();

//This part was not required, but it is always a good idea to read the file
//that was created to make sure it is correct.
//We first read the original input file, and display its content
```

```
  indisplay.open(inName.c_str());
  if (indisplay.fail())
  {
      cout << "Cannot open file "
           << inName << " Aborting!" << endl;
      exit(1);
  }
      cout << endl << endl << "The contents of the input file is : "
           << endl << endl;
      checkFile(indisplay);
      indisplay.close();

  //Now we read the file that was created as an input file and display the
  //content
  outdisplay.open(outName.c_str());
  if (outdisplay.fail())
  {
      cout << "Cannot open file "
           << outName << " Aborting!" << endl;
      exit(1);
  }
      cout << endl<<"The contents of the created output file is : "
           << endl << endl;
      checkFile(outdisplay);
      outdisplay.close();

    return 0;
  }
```

Input and corresponding output 1:

```
Please enter the input file name.
wrongfilename.txt
Cannot open file wrongfilename.txt Aborting!
Press any key to continue . . .
```

Input and corresponding output 2:

```
Enter the input file name.
rhyme.txt

Enter the output file name.
WARNING: ANY EXISTING FILE WITH THIS NAME WILL BE ERASED.
changed.txt

The contents of the input file is :

        Chubby Little Snowman

       A chubby little snowman,
           Had a carrot nose.
```

```
          Along came a bunny
      And what do you suppose?

       That hungry little bunny
      Was looking for his lunch.
  He ate that little snowman's nose:
   Nibble, Nibble, Crunch, Crunch!

The contents of the created output file is :

CHUBBY LITTLE SNOWMAN

A CHUBBY LITTLE SNOWMAN,
HAD A CARROT NOSE.
ALONG CAME A BUNNY
AND WHAT DO YOU SUPPOSE?

THAT HUNGRY LITTLE BUNNY
WAS LOOKING FOR HIS LUNCH.
HE ATE THAT LITTLE SNOWMAN'S NOSE:
NIBBLE, NIBBLE, CRUNCH, CRUNCH!

Press any key to continue . . .
```

Another Solution

Since there is frequently more than one way to solve a problem and write a program, we include a very elegant solution submitted by one of your fellow students. This solution uses the `putback()` function to return a character to the file once it finds a non-blank character (extracted with `get()` from the file):

```
    //Remove leading blanks in first line
    fin.get(next);
    while (next == ' ')
    {
        fin.get(next);
    }
    fin.putback(next);
```

If we run the program below with the same input file as in the previous solution, we will get exactly the same output file as before.

Program listing:

```
//Chubby little snowman
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    char next;
    ifstream fin;
```

```
    ofstream fout;

    //Open input and output files and test both for successful opening
    //If file opening was not successful display error message and exit program
    fin.open("rhyme.txt");
    if (fin.fail())
    {
        cout << "Input file opening failed.\n";
        exit(1);
    }

    fout.open("changed.txt");
    if (fout.fail())
    {
        cout << "output file opening failed.\n";
        exit(1);
    }

    //Remove leading blanks in first line
    fin.get(next);
    while (next == ' ')
    {
        fin.get(next);
    }
    fin.putback(next);

    //Loop through the rest of the file until end remove leading blanks
    //and also convert all characters to uppercase and output results in
    //output file
    while (!fin.eof())
    {
       //Check for end of line and remove leading blanks in next line
       if(next == '\n')
       {
          fin.get(next);
          while(next == ' ')
          {
             fin.get(next);
          }
          fin.putback(next);
       }

        fin.get(next);
        fout.put(toupper(next));
     }

return 0;
}
```

Question 5 [10 marks]

(i) `double *fPtr1, *fPtr2;`
(ii) `fPtr1 = &total;`
(iii) `cout << “The contents of fPtr1 is “ << fPtr1 ;`
(iv) `fPtr2 = &interest;`
(v) `*fPtr1 += *fPtr2;`
(vi) `fPtr2 = fPtr1;`
(vii) `cout << “The contents of fPtr2 is “ << fPtr2 ;`
(viii) `cout << “The value of the object pointed to by fPtr1 is “ << *fPtr1;`

b) We reflect the state of memory diagrammatically after the statements above each diagram have been executed:

```
p = &x;
q = &y;
```

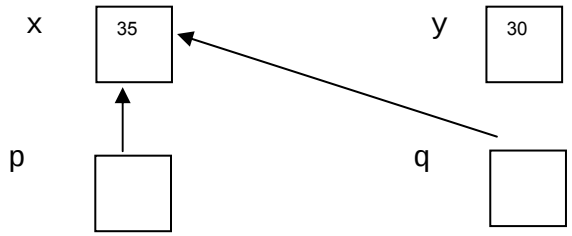

```
*p = 20;
*q += 10;
```


```
cout << x << “ “ << y << endl;
cout << *q << “ “ << *p << endl;

20 30
30 20

*p = *q + 5;
```

```
cout << *p << " " << *q << endl;
```

```
35 30
```

```
q = p;
```




```
cout << *p << " " << *q << endl;
```

```
35 35
```

Discussion:
The last diagram shows the final state of memory when the last output statement is executed.
The output after the statement will therefore be:

```
20 30
30 20
35 30
35 35
```

It should have been fairly straightforward to follow the diagrams above. If you did not manage to follow these diagrams, study chapter 9 in Savitch again, and ensure that you know what each of the statements used in this program segment means.
