Consolidated Assignment 8 Report

This report contains the graded results for the newest of each exercise submitted to the assignment checker prior to 5/27/2020 12:04:22 PM PDT.

Student Name: Shaun Chemplavil
Student ID: U08713628
Contact e-mail: shaun.chemplavil@gmail.com
C/C++ Programming I (Section 146359)

Submitted:
- Exercise 0: 5/1/2020 7:56:16 AM PDT
- Exercise 1: 5/8/2020 8:37:49 AM PDT
- Exercise 2: 5/8/2020 8:38:00 AM PDT
- Exercise 3: 5/8/2020 8:38:11 AM PDT

Score (out of 20 possible): 18

Never use the numeric value of a character if it can be represented another way (literally or simple escape sequence).

THIS WAS SENT FROM A NOTIFICATION-ONLY ADDRESS THAT CANNOT ACCEPT INCOMING MAIL.
For help please contact the instructor at the email address provided on the "Announcements" page of the course website. The assignment checker DOES NOT GRADE your submissions but merely reports on issues so you can correct them and resubmit, thereby avoiding unnecessary credit loss. ALL GRADING IS DONE MANUALLY BY THE INSTRUCTOR after the assignment deadline based solely upon the NEWEST submission of each exercise. BE WARY of correcting minor issues after the deadline because a late deduction will usually be much greater than a minor issue deduction.

From: Shaun Chemplavil <mailto:shaun.chemplavil@gmail.com>
Subject: C1A8E0_U08713628
Submitted: 5/1/2020 7:56:16 AM PDT
Course: C/C++ Programming I (Section 146359)
Student's name: Shaun Chemplavil
Contact email: shaun.chemplavil@gmail.com
Student ID: U08713628
Assignment 8, Exercise 0
Exercise point value: 6
File submitted:
C1A8E0_Quiz.txt

NOTE: The assignment checker does not check the correctness of quiz answers for this assignment.

Your submission has been accepted and will be graded manually by the instructor. You may resubmit it as many times as you wish before the assignment deadline. BE WARY of correcting minor issues after the deadline because a late deduction will usually be much greater than a minor issue deduction.

-2

Shaun Chemplavil U08713628
shaun.chemplavil@gmail.com
C/C++ Programming I : Fundamental Programming Concepts
146359 Raymond L. Mitchell, Jr., M.S.
05/01/2020
C1A8E0_Quiz.txt
Answers to Quiz

1. E
2. A
3. D
4. D <---E
5. D
6. B <---A

THIS WAS SENT FROM A NOTIFICATION-ONLY ADDRESS THAT CANNOT ACCEPT INCOMING MAIL.
For help please contact the instructor at the email address provided on the "Announcements" page of the course website. The assignment checker DOES NOT GRADE your submissions but merely reports on issues so you can correct them and resubmit, thereby avoiding unnecessary credit loss. ALL GRADING IS DONE MANUALLY BY THE INSTRUCTOR after the assignment deadline based solely upon the NEWEST submission of each exercise. BE WARY of correcting minor issues after the deadline because a late deduction will usually be much greater than a minor issue deduction.

From: Shaun Chemplavil <mailto:shaun.chemplavil@gmail.com>
Subject: C1A8E1_U08713628
Submitted: 5/8/2020 8:37:49 AM PDT
Course: C/C++ Programming I (Section 146359)
Student's name: Shaun Chemplavil
Contact email: shaun.chemplavil@gmail.com
Student ID: U08713628
Assignment 8, Exercise 1
Exercise point value: 4
Files submitted:
C1A8E1_SavingsAccount.h
C1A8E1_SavingsAccount.cpp
C1A8E1_main.cpp

"Compile-time" results:
No "compile-time" issues;
"Run-time" results:
Program ran - No errors detected during preliminary testing (SEE ATTACHMENT);

```
1   //
2   // Shaun Chemplavil U08713628
3   // shaun.chemplavil@gmail.com
4   // C / C++ Programming I : Fundamental Programming Concepts
5   // 146359 Raymond L. Mitchell Jr.
6   // 05 / 08 / 2020
7   // C1A8E1_SavingsAccount.h
8   // Win10
9   // Visual C++ 19.0
10  //
11  // This header file contains the definition of the SavingsAccount class
12  //
13
14  #ifndef C1A8E1_SAVINGSACCOUNT_H
15  #define C1A8E1_SAVINGSACCOUNT_H
16
17  #include <iostream>
18
19  const double PERCENT_TO_DECIMAL = 0.01;
20
21  class SavingsAccount
22  {
23     int type;
24     string ownerName;
25     long IDnbr;
26     double balance, closurePenaltyPercent;
27
28  public:
29
30     void GetInitialValues();
31     inline void DisplayValues() const;
32
33     inline double CalculatePenalty() const
34     {
35        // Calculate and save total fine for closing account
36        double fine = balance * closurePenaltyPercent * PERCENT_TO_DECIMAL;
37
38        return fine;
39     }
40  };
41
42  // Member function to display account information
43  inline void SavingsAccount::DisplayValues() const
44  {
45     std::cout <<
46        "Account type: " << type << "\n"
47        "Owner name: " << ownerName << "\n"
48        "ID number: " << IDnbr << "\n"
49        "Account balance: " << balance << "\n"
50        "Account closure penalty percent: " << closurePenaltyPercent << "\n";
51  }
52
53  #endif
```

```cpp
//
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C / C++ Programming I : Fundamental Programming Concepts
// 146359 Raymond L. Mitchell Jr.
// 05 / 08 / 2020
// C1A8E1_main.cpp
// Win10
// Visual C++ 19.0
//
// This program asks the user to enter account information to populate a
//  SavingsAccount class variable, and then displays the dollar amount
//  penalty to close the account
//

#include <iostream>
using namespace std;

#include "C1A8E1_SavingsAccount.h"

int main()
{
   SavingsAccount account;

   // Prompt user and set SavingsAccount member variables
   account.GetInitialValues();
   // Display member variable values
   account.DisplayValues();

   // Calculate and display the dollar amount of an account closure penalty
   cout << "Account closure penalty: " << account.CalculatePenalty() << "\n";

   return 0;
}
```

```
1
//
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C / C++ Programming I : Fundamental Programming Concepts
// 146359 Raymond L. Mitchell Jr.
// 05 / 08 / 2020
// C1A8E1_SavingsAccount.cpp
// Win10
// Visual C++ 19.0
//
// This function allows a user to set the SavingsAccount class member variables
//

#include <iostream>
#include <string>

using namespace std;

#include "C1A8E1_SavingsAccount.h"

void SavingsAccount::GetInitialValues()
{
   // Prompt the user to set SavingsAccount member variables
   cout << "Enter the account type: ";
   cin >> type;
   cout << "Enter the owner name: ";
   // Ignore leading whitespace
   cin >> ws;
   // Capture all remaining characters until '\n' is detected
   getline(cin, ownerName);
   cout << "Enter the ID number: ";
   cin >> IDnbr;
   cout << "Enter the account balance: ";
   cin >> balance;
   cout << "Enter the account closure penalty percentage: ";
   cin >> closurePenaltyPercent;
}
```

```
********** C1 ASSIGNMENT 8 EXERCISE 1 AUTOMATIC PROGRAM RUN RESULTS **********

************* THE RESULTS BELOW HAVE BEEN PARTIALLY CHECKED AND **************
*************   NO ERRORS WERE FOUND.  HOWEVER, THIS DOES NOT   **************
*************  NECESSARILY MEAN THAT THERE ARE NO ERRORS.  THE  **************
*************  INSTRUCTOR WILL DO A MORE THOROUGH CHECK DURING  **************
*************                  MANUAL GRADING.                    **************


------------------------------ START OF 1ST RUN ------------------------------

Enter the account type: 80
Enter the owner name: Big Spender
Enter the ID number: 123456789
Enter the account balance: .20
Enter the account closure penalty percentage: 1.3
Account type: 80
Owner name: Big Spender
ID number: 123456789
Account balance: 0.2
Account closure penalty percent: 1.3
Account closure penalty: 0.0026

------------------------------- END OF 1ST RUN -------------------------------

------------------------------ START OF 2ND RUN ------------------------------

Enter the account type: 9
Enter the owner name: Elvis Clone
Enter the ID number: 2345780
Enter the account balance: 300e1
Enter the account closure penalty percentage: 15
Account type: 9
Owner name: Elvis Clone
ID number: 2345780
Account balance: 3000
Account closure penalty percent: 15
Account closure penalty: 450

------------------------------- END OF 2ND RUN -------------------------------

------------------------------ START OF 3RD RUN ------------------------------

Enter the account type: 32767
Enter the owner name: Geezer
Enter the ID number: 789
Enter the account balance: .40
Enter the account closure penalty percentage: 1.3
Account type: 32767
Owner name: Geezer
ID number: 789
Account balance: 0.4
Account closure penalty percent: 1.3
Account closure penalty: 0.0052

------------------------------- END OF 3RD RUN -------------------------------
```

THIS WAS SENT FROM A NOTIFICATION-ONLY ADDRESS THAT CANNOT ACCEPT INCOMING MAIL.
For help please contact the instructor at the email address provided on the "Announcements" page of the course website. The assignment checker DOES NOT GRADE your submissions but merely reports on issues so you can correct them and resubmit, thereby avoiding unnecessary credit loss. ALL GRADING IS DONE MANUALLY BY THE INSTRUCTOR after the assignment deadline based solely upon the NEWEST submission of each exercise. BE WARY of correcting minor issues after the deadline because a late deduction will usually be much greater than a minor issue deduction.

From: Shaun Chemplavil <mailto:shaun.chemplavil@gmail.com>
Subject: C1A8E2_U08713628
Submitted: 5/8/2020 8:38:00 AM PDT
Course: C/C++ Programming I (Section 146359)
Student's name: Shaun Chemplavil
Contact email: shaun.chemplavil@gmail.com
Student ID: U08713628
Assignment 8, Exercise 2
Exercise point value: 4
File submitted:
C1A8E2_main.c

"Compile-time" results:
No "compile-time" issues;
"Run-time" results:
Program ran - No errors detected during preliminary testing (SEE ATTACHMENT);

```
1  //
2  // Shaun Chemplavil U08713628
3  // shaun.chemplavil@gmail.com
4  // C / C++ Programming I : Fundamental Programming Concepts
5  // 146359 Raymond L. Mitchell Jr.
6  // 05 / 08 / 2020
7  // C1A8E2_main.c
8  // Win10
9  // Visual C++ 19.0
10 //
11 // This program will run at the command line with two space separated arguments
12 //  first argument specifies a text file name, and the second specifies the
13 //  number of lines in a group
14 //
15
16 #include <stdio.h>
17 #include <stdlib.h>
18
19 #define BUFSIZE 256
20 #define EXPECTED_ARGS 3
21 #define NEW_LINE_ASCII 10
22 #define FILENAME_ARG  1
23 #define DISP_LINE_ARG  2
24
25 void ErrorAndExit(const char *filename)
26 {
27    fprintf(stderr, "%s  failed to open \n", filename);
28    exit(EXIT_FAILURE);
29 }
30
31 int main(int argc, char *argv[])
32 {
33    // Check to make ensure correct amount of arguments,
34    if (argc != EXPECTED_ARGS)
35    {
36       fputs("Unexpected number of arguments \n", stderr);
37       exit(EXIT_FAILURE);
38    }
39
40    int   numDispLines, lineCnt = 0;
41    char *filename, buf[BUFSIZE];
42    FILE *source;
43
44    // Store Arguments
45    filename = argv[FILENAME_ARG];
46    numDispLines = atoi(argv[DISP_LINE_ARG]);
47
48    // Open file containing we want to display
49    if ((source = fopen(filename, "r")) == NULL)
50       ErrorAndExit(filename);
51
52    // Display requested amount of lines
53    while (lineCnt < numDispLines)
54    {
55       // store line from source file into buffer
56       fgets(buf, (int)sizeof(buf), source);
57
58       // break loop if End of File reached
59       if (feof(source))
60          break;
61
```

Never use the numeric value of a character if it can be represented another way (literally or simple escape sequence).

```
62          // display line from buffer
63          printf("%s", buf);
64
65          // once we have displayed the request amount of lines
66          // check if user has entered '\n' to indicate if the same amount of lines
67          // (until EOF reached) should be displayed
68          if ((++lineCnt == numDispLines) && (getchar() == NEW_LINE_ASCII))
69             lineCnt = 0;
70       }
71
72       fclose(source);
73
74       return(EXIT_SUCCESS);
75    }
```

```
********** C1 ASSIGNMENT 8 EXERCISE 2 AUTOMATIC PROGRAM RUN RESULTS **********

************* THE RESULTS BELOW HAVE BEEN PARTIALLY CHECKED AND **************
*************   NO ERRORS WERE FOUND.  HOWEVER, THIS DOES NOT   **************
*************  NECESSARILY MEAN THAT THERE ARE NO ERRORS.  THE  **************
*************  INSTRUCTOR WILL DO A MORE THOROUGH CHECK DURING  **************
*************                  MANUAL GRADING.                    **************


------------------------------ PURPOSE OF 1ST RUN ------------------------------
Verify file line grouping.
---------------------- COMMAND LINE ARGUMENTS FOR 1ST RUN ----------------------
TestFile1.txt 3
------------------------------ START OF 1ST RUN ------------------------------

The number-sign or "stringizing" operator (#) converts macro
parameters (after expansion) to string constants. It is used
only with macros that take arguments. If it precedes a formal

parameter in the macro definition, the actual argument passed
by the macro invocation is enclosed in quotation marks and
treated as a string literal. The string literal then replaces

each occurrence of a combination of the stringizing operator
and formal parameter within the macro definition.


White space preceding the first token of the actual argument
and following the last token of the actual argument is ignored.
Any white space between the tokens in the actual argument is

reduced to a single white space in the resulting string literal.
Thus, if a comment occurs between two tokens in the actual
argument, it is reduced to a single white space. The resulting

string literal is automatically concatenated with any adjacent
string literals from which it is separated only by white space.



------------------------------- END OF 1ST RUN -------------------------------

------------------------------ PURPOSE OF 2ND RUN ------------------------------
Verify file line grouping.
---------------------- COMMAND LINE ARGUMENTS FOR 2ND RUN ----------------------
TestFile1.txt 11
------------------------------ START OF 2ND RUN ------------------------------

The number-sign or "stringizing" operator (#) converts macro
parameters (after expansion) to string constants. It is used
only with macros that take arguments. If it precedes a formal
parameter in the macro definition, the actual argument passed
by the macro invocation is enclosed in quotation marks and
treated as a string literal. The string literal then replaces
each occurrence of a combination of the stringizing operator
and formal parameter within the macro definition.

White space preceding the first token of the actual argument
and following the last token of the actual argument is ignored.
```

```
Any white space between the tokens in the actual argument is
reduced to a single white space in the resulting string literal.
Thus, if a comment occurs between two tokens in the actual
argument, it is reduced to a single white space. The resulting
string literal is automatically concatenated with any adjacent
string literals from which it is separated only by white space.



------------------------------- END OF 2ND RUN -------------------------------

----------------------------- PURPOSE OF 3RD RUN -----------------------------
Verify file line grouping.
--------------------- COMMAND LINE ARGUMENTS FOR 3RD RUN ---------------------
TestFile1.txt 5
------------------------------ START OF 3RD RUN ------------------------------

The number-sign or "stringizing" operator (#) converts macro
parameters (after expansion) to string constants. It is used
only with macros that take arguments. If it precedes a formal
parameter in the macro definition, the actual argument passed
by the macro invocation is enclosed in quotation marks and

treated as a string literal. The string literal then replaces
each occurrence of a combination of the stringizing operator
and formal parameter within the macro definition.

White space preceding the first token of the actual argument
+

------------------------------- END OF 3RD RUN -------------------------------

----------------------------- PURPOSE OF 4TH RUN -----------------------------
Verify file line grouping.
--------------------- COMMAND LINE ARGUMENTS FOR 4TH RUN ---------------------
TestFile9.txt 3
------------------------------ START OF 4TH RUN ------------------------------

A_11111111111111111111111111111111111111111111111111111111111111111111111111111111
B_22222222222222222222222222222222222222222222222222222222222222222222222222222222
C_33333333333333333333333333333333333333333333333333333333333333333333333333333333

D_44444444444444444444444444444444444444444444444444444444444444444444444444444444
E_55555555555555555555555555555555555555555555555555555555555555555555555555555555
F_66666666666666666666666666666666666666666666666666666666666666666666666666666666

G_77777777777777777777777777777777777777777777777777777777777777777777777777777777
H_88888888888888888888888888888888888888888888888888888888888888888888888888888888
I_99999999999999999999999999999999999999999999999999999999999999999999999999999999

J_00000000000000000000000000000000000000000000000000000000000000000000000000000000
K_11111111111111111111111111111111111111111111111111111111111111111111111111111111
L_22222222222222222222222222222222222222222222222222222222222222222222222222222222

M_33333333333333333333333333333333333333333333333333333333333333333333333333333333
N_44444444444444444444444444444444444444444444444444444444444444444444444444444444
0_55555555555555555555555555555555555555555555555555555555555555555555555555555555

P_66666666666666666666666666666666666666666666666666666666666666666666666666666666
Q_77777777777777777777777777777777777777777777777777777777777777777777777777777777
```

```
R_8888888888888888888888888888888888888888888888888888888888888888888888888888

S_9999999999999999999999999999999999999999999999999999999999999999999999999999
T_0000000000000000000000000000000000000000000000000000000000000000000000000000
U_1111111111111111111111111111111111111111111111111111111111111111111111111111

V_2222222222222222222222222222222222222222222222222222222222222222222222222222
W_3333333333333333333333333333333333333333333333333333333333333333333333333333
X_4444444444444444444444444444444444444444444444444444444444444444444444444444

Y_5555555555555555555555555555555555555555555555555555555555555555555555555555
Z_6666666666666666666666666666666666666666666666666666666666666666666666666666
@_7777777777777777777777777777777777777777777777777777777777777777777777777777

#_8888888888888888888888888888888888888888888888888888888888888888888888888888
$_9999999999999999999999999999999999999999999999999999999999999999999999999999
%_0000000000000000000000000000000000000000000000000000000000000000000000000000

&_1111111111111111111111111111111111111111111111111111111111111111111111111111
?_2222222222222222222222222222222222222222222222222222222222222222222222222222


-------------------------------- END OF 4TH RUN --------------------------------

------------------------------ PURPOSE OF 5TH RUN ------------------------------
Verify file line grouping.
---------------------- COMMAND LINE ARGUMENTS FOR 5TH RUN ----------------------
TestFile9.txt 11
------------------------------- START OF 5TH RUN -------------------------------

A_1111111111111111111111111111111111111111111111111111111111111111111111111111
B_2222222222222222222222222222222222222222222222222222222222222222222222222222
C_3333333333333333333333333333333333333333333333333333333333333333333333333333
D_4444444444444444444444444444444444444444444444444444444444444444444444444444
E_5555555555555555555555555555555555555555555555555555555555555555555555555555
F_6666666666666666666666666666666666666666666666666666666666666666666666666666
G_7777777777777777777777777777777777777777777777777777777777777777777777777777
H_8888888888888888888888888888888888888888888888888888888888888888888888888888
I_9999999999999999999999999999999999999999999999999999999999999999999999999999
J_0000000000000000000000000000000000000000000000000000000000000000000000000000
K_1111111111111111111111111111111111111111111111111111111111111111111111111111

L_2222222222222222222222222222222222222222222222222222222222222222222222222222
M_3333333333333333333333333333333333333333333333333333333333333333333333333333
N_4444444444444444444444444444444444444444444444444444444444444444444444444444
0_5555555555555555555555555555555555555555555555555555555555555555555555555555
P_6666666666666666666666666666666666666666666666666666666666666666666666666666
Q_7777777777777777777777777777777777777777777777777777777777777777777777777777
R_8888888888888888888888888888888888888888888888888888888888888888888888888888
S_9999999999999999999999999999999999999999999999999999999999999999999999999999
T_0000000000000000000000000000000000000000000000000000000000000000000000000000
U_1111111111111111111111111111111111111111111111111111111111111111111111111111
V_2222222222222222222222222222222222222222222222222222222222222222222222222222

W_3333333333333333333333333333333333333333333333333333333333333333333333333333
X_4444444444444444444444444444444444444444444444444444444444444444444444444444
Y_5555555555555555555555555555555555555555555555555555555555555555555555555555
Z_6666666666666666666666666666666666666666666666666666666666666666666666666666
@_7777777777777777777777777777777777777777777777777777777777777777777777777777
#_8888888888888888888888888888888888888888888888888888888888888888888888888888
```

```
$_99999999999999999999999999999999999999999999999999999999999999999999999999
%_00000000000000000000000000000000000000000000000000000000000000000000000000
&_11111111111111111111111111111111111111111111111111111111111111111111111111
?_22222222222222222222222222222222222222222222222222222222222222222222222222


------------------------------ END OF 5TH RUN ------------------------------

---------------------------- PURPOSE OF 6TH RUN ----------------------------
Verify file line grouping.
--------------------- COMMAND LINE ARGUMENTS FOR 6TH RUN ---------------------
TestFile9.txt 5
------------------------------ START OF 6TH RUN ------------------------------

A_11111111111111111111111111111111111111111111111111111111111111111111111111
B_22222222222222222222222222222222222222222222222222222222222222222222222222
C_33333333333333333333333333333333333333333333333333333333333333333333333333
D_44444444444444444444444444444444444444444444444444444444444444444444444444
E_55555555555555555555555555555555555555555555555555555555555555555555555555

F_66666666666666666666666666666666666666666666666666666666666666666666666666
G_77777777777777777777777777777777777777777777777777777777777777777777777777
H_88888888888888888888888888888888888888888888888888888888888888888888888888
I_99999999999999999999999999999999999999999999999999999999999999999999999999
J_00000000000000000000000000000000000000000000000000000000000000000000000000
+

------------------------------ END OF 6TH RUN ------------------------------

---------------------------- PURPOSE OF 7TH RUN ----------------------------
Verify that program detects an input file open failure.
--------------------- COMMAND LINE ARGUMENTS FOR 7TH RUN ---------------------
bad//file//a 5
------------------------------ START OF 7TH RUN ------------------------------

bad//file//a  failed to open



------------------------------ END OF 7TH RUN ------------------------------

---------------------------- PURPOSE OF 8TH RUN ----------------------------
Verify that program detects an input file open failure.
--------------------- COMMAND LINE ARGUMENTS FOR 8TH RUN ---------------------
bad//file//b 5
------------------------------ START OF 8TH RUN ------------------------------

bad//file//b  failed to open



------------------------------ END OF 8TH RUN ------------------------------

---------------------------- PURPOSE OF 9TH RUN ----------------------------
Verify that program detects too many arguments.
--------------------- COMMAND LINE ARGUMENTS FOR 9TH RUN ---------------------
ARGV1 ARGV2 ARGV3 5
------------------------------ START OF 9TH RUN ------------------------------

Unexpected number of arguments
```

```
------------------------------ END OF 9TH RUN ------------------------------

---------------------------- PURPOSE OF 10TH RUN ----------------------------
Verify that program detects too few arguments.
--------------------- COMMAND LINE ARGUMENTS FOR 10TH RUN ---------------------
 5
----------------------------- START OF 10TH RUN -----------------------------

Unexpected number of arguments



------------------------------ END OF 10TH RUN ------------------------------
```

THIS WAS SENT FROM A NOTIFICATION-ONLY ADDRESS THAT CANNOT ACCEPT INCOMING MAIL.
For help please contact the instructor at the email address provided on the "Announcements" page of the course website. The assignment checker DOES NOT GRADE your submissions but merely reports on issues so you can correct them and resubmit, thereby avoiding unnecessary credit loss. ALL GRADING IS DONE MANUALLY BY THE INSTRUCTOR after the assignment deadline based solely upon the NEWEST submission of each exercise. BE WARY of correcting minor issues after the deadline because a late deduction will usually be much greater than a minor issue deduction.

From: Shaun Chemplavil <mailto:shaun.chemplavil@gmail.com>
Subject: C1A8E3_U08713628
Submitted: 5/8/2020 8:38:11 AM PDT
Course: C/C++ Programming I (Section 146359)
Student's name: Shaun Chemplavil
Contact email: shaun.chemplavil@gmail.com
Student ID: U08713628
Assignment 8, Exercise 3
Exercise point value: 6
File submitted:
C1A8E3_main.cpp

"Compile-time" results:
No "compile-time" issues;
"Run-time" results:
Program ran - No errors detected during preliminary testing (SEE ATTACHMENT);

```
1  //
2  // Shaun Chemplavil U08713628
3  // shaun.chemplavil@gmail.com
4  // C / C++ Programming I : Fundamental Programming Concepts
5  // 146359 Raymond L. Mitchell Jr.
6  // 05 / 08 / 2020
7  // C1A8E3_main.cpp
8  // Win10
9  // Visual C++ 19.0
10 //
11 // This program will take command line arguments to 'find and replace' items
12 // within a text file the output will be in a new text file
13 //
14
15 #include <iostream>
16 #include <fstream>
17
18 using namespace std;
19
20 const int BUFSIZE = 256;
21 const int EXPECTED_ARGS = 5;
22 const int SOURCE_ARG = 1;
23 const int DESTIN_ARG = 2;
24 const int SEARCH_ARG = 3;
25 const int REPLAC_ARG = 4;
26
27 void ErrorAndQuit(const char *myString)
28 {
29    cerr << "\"" << myString << "\" :File access error!\n";
30    exit(EXIT_FAILURE);
31 }
32
33 int main(int argc, char *argv[])
34 {
35    // Check to make ensure correct amount of arguments,
36    if (argc != EXPECTED_ARGS)
37    {
38       cerr << "Unexpected number of arguments!\n";
39       exit(EXIT_FAILURE);
40    }
41
42    char buf[BUFSIZE], *sourceFile, *destFile, *searchString, *replaceString;
43
44    size_t numOff;
45
46    sourceFile = argv[SOURCE_ARG];
47    destFile = argv[DESTIN_ARG];
48    searchString = argv[SEARCH_ARG];
49    replaceString = argv[REPLAC_ARG];
50
51    // open SOURCE_FILE in "read" mode
52    ifstream source(sourceFile);
53    if (!source.is_open())
54       ErrorAndQuit(sourceFile);
55
56    // open destination file in "append" mode
57    ofstream destination(destFile, ios_base::app);
58    if (!destination.is_open())
59       ErrorAndQuit(destFile);
60
61    // Determine to offset applied to "write" pointer when searchString found
```

What does this accomplish?

```
62    numOff = strlen(searchString);
63
64    // Read each line from source
65    while (source.getline(buf, sizeof(buf)) && !source.eof())
66    {
67       char *pSrch, *pline;
68
69       // Place character from beginning of line until searchString location
70       for (pline = &buf[0]; pSrch = strstr(pline, searchString);)
71       {
72          // Write characters up to searchString
73          destination.write(pline, pSrch - pline);
74
75          // Write replaceString instead
76          destination << replaceString;
77          //Offset pline to location after searchString
78          pline = pSrch + numOff;
79       }
80
81       // Write remaining character of line and include '\n'
82       destination << pline << '\n';
83    }
84
85    // Close open files
86    source.close();
87    destination.close();
88
89    return(EXIT_SUCCESS);
90 }
```

********** C1 ASSIGNMENT 8 EXERCISE 3 AUTOMATIC PROGRAM RUN RESULTS **********

********** THE RESULTS BELOW HAVE BEEN PARTIALLY CHECKED AND **********
********** NO ERRORS WERE FOUND. HOWEVER, THIS DOES NOT **********
********** NECESSARILY MEAN THAT THERE ARE NO ERRORS. THE **********
********** INSTRUCTOR WILL DO A MORE THOROUGH CHECK DURING **********
********** MANUAL GRADING. **********

---------- PURPOSE OF 1ST RUN ----------
Verify "find and replace".
---------- COMMAND LINE ARGUMENTS FOR 1ST RUN ----------
TestFile1.txt TestFile1_modified1.txt the "John Galt?"
---------- START OF 1ST RUN ----------

<<YOUR CORRECTLY MODIFIED FILE>>

The number-sign or "stringizing" operator (#) converts macro
parameters (after expansion) to string constants. It is used
only with macros that take arguments. If it precedes a formal
parameter in John Galt? macro definition, John Galt? actual argument passed
by John Galt? macro invocation is enclosed in quotation marks and
treated as a string literal. The string literal John Galt?n replaces
each occurrence of a combination of John Galt? stringizing operator
and formal parameter within John Galt? macro definition.

White space preceding John Galt? first token of John Galt? actual argument
and following John Galt? last token of John Galt? actual argument is ignored.
Any white space between John Galt? tokens in John Galt? actual argument is
reduced to a single white space in John Galt? resulting string literal.
Thus, if a comment occurs between two tokens in John Galt? actual
argument, it is reduced to a single white space. The resulting
string literal is automatically concatenated with any adjacent
string literals from which it is separated only by white space.

---------- END OF 1ST RUN ----------

---------- PURPOSE OF 2ND RUN ----------
Verify "find and replace".
---------- COMMAND LINE ARGUMENTS FOR 2ND RUN ----------
TestFile1.txt TestFile1_modified2.txt "string literal" TESTING
---------- START OF 2ND RUN ----------

<<YOUR CORRECTLY MODIFIED FILE>>

The number-sign or "stringizing" operator (#) converts macro
parameters (after expansion) to string constants. It is used
only with macros that take arguments. If it precedes a formal
parameter in the macro definition, the actual argument passed
by the macro invocation is enclosed in quotation marks and
treated as a TESTING. The TESTING then replaces
each occurrence of a combination of the stringizing operator
and formal parameter within the macro definition.

White space preceding the first token of the actual argument
and following the last token of the actual argument is ignored.
Any white space between the tokens in the actual argument is
reduced to a single white space in the resulting TESTING.
Thus, if a comment occurs between two tokens in the actual

```
argument, it is reduced to a single white space. The resulting
TESTING is automatically concatenated with any adjacent
TESTINGs from which it is separated only by white space.


------------------------------ END OF 2ND RUN ------------------------------

---------------------------- PURPOSE OF 3RD RUN ----------------------------
Verify "find and replace".
--------------------- COMMAND LINE ARGUMENTS FOR 3RD RUN ---------------------
TestFile1.txt  TestFile1_modified3.txt  d  "X y Z"
------------------------------ START OF 3RD RUN ------------------------------

<<YOUR CORRECTLY MODIFIED FILE>>

The number-sign or "stringizing" operator (#) converts macro
parameters (after expansion) to string constants. It is useX y Z
only with macros that take arguments. If it preceX y Zes a formal
parameter in the macro X y Zefinition, the actual argument passeX y Z
by the macro invocation is encloseX y Z in quotation marks anX y Z
treateX y Z as a string literal. The string literal then replaces
each occurrence of a combination of the stringizing operator
anX y Z formal parameter within the macro X y Zefinition.

White space preceX y Zing the first token of the actual argument
anX y Z following the last token of the actual argument is ignoreX y Z.
Any white space between the tokens in the actual argument is
reX y ZuceX y Z to a single white space in the resulting string literal.
Thus, if a comment occurs between two tokens in the actual
argument, it is reX y ZuceX y Z to a single white space. The resulting
string literal is automatically concatenateX y Z with any aX y Zjacent
string literals from which it is separateX y Z only by white space.


------------------------------ END OF 3RD RUN ------------------------------

---------------------------- PURPOSE OF 4TH RUN ----------------------------
Verify "find and replace".
--------------------- COMMAND LINE ARGUMENTS FOR 4TH RUN ---------------------
mFile3.txt  TestFile1_modified4.txt  input  output
------------------------------ START OF 4TH RUN ------------------------------

<<EMPTY FILE - YOUR RESULTS WERE CORRECT>>


------------------------------ END OF 4TH RUN ------------------------------

---------------------------- PURPOSE OF 5TH RUN ----------------------------
Verify that program detects an input file open failure.
--------------------- COMMAND LINE ARGUMENTS FOR 5TH RUN ---------------------
bad//file//a  TestFile1_modified4.txt  1  2
------------------------------ START OF 5TH RUN ------------------------------

"bad//file//a" :File access error!

------------------------------ END OF 5TH RUN ------------------------------

---------------------------- PURPOSE OF 6TH RUN ----------------------------
Verify that program detects an output file open failure.
--------------------- COMMAND LINE ARGUMENTS FOR 6TH RUN ---------------------
```

```
TestFile1.txt  bad//file//b  3  4
------------------------------ START OF 6TH RUN ------------------------------

"bad//file//b" :File access error!

------------------------------- END OF 6TH RUN -------------------------------

----------------------------- PURPOSE OF 7TH RUN -----------------------------
Verify that program detects too many arguments.
--------------------- COMMAND LINE ARGUMENTS FOR 7TH RUN ---------------------
ARGV1 ARGV2 ARGV3 ARGV4 ARGV5 ARGV6
------------------------------ START OF 7TH RUN ------------------------------

Unexpected number of arguments!

------------------------------- END OF 7TH RUN -------------------------------

----------------------------- PURPOSE OF 8TH RUN -----------------------------
Verify that program detects too few arguments.
--------------------- COMMAND LINE ARGUMENTS FOR 8TH RUN ---------------------
ARGV1 ARGV2
------------------------------ START OF 8TH RUN ------------------------------

Unexpected number of arguments!

------------------------------- END OF 8TH RUN -------------------------------
```