Consolidated Assignment 8 Report

This report contains the graded results for the newest of each exercise submitted to the assignment checker prior to 3/9/2022 12:05:59 AM PST.

Student Name: Phillip Ward
Student ID: U09339367
Contact email: phillip.ward@seagate.com
C/C++ Programming I (Section 162461)

Submitted:
   Exercise 0: 3/6/2022 2:49:14 PM PST
   Exercise 1: 3/5/2022 12:23:51 PM PST
   Exercise 2: 3/6/2022 2:55:53 PM PST

Score (out of 20 possible): 14.7

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 "Home" page of the course's Canvas website. The assignment checker DOES NOT GRADE your submissions but merely reports on issues so you can avoid credit loss by making corrections and resubmitting. ALL GRADING IS DONE MANUALLY BY THE INSTRUCTOR after the assignment deadline based solely upon the NEWEST submission of each exercise that was submitted BEFORE THE ASSIGNMENT DEADLINE. NO CREDIT will be given for anything submitted after the deadline.

From: Phillip Ward <mailto:phillip.ward@seagate.com>
Subject: C1A8E0_162461_U09339367
Submitted: 3/6/2022 2:49:14 PM PST
Course: C/C++ Programming I (Section 162461)
Student's name: Phillip Ward
Contact email: phillip.ward@seagate.com
Student ID: U09339367
Assignment 8, Exercise 0 (003285666M01005X40285)
Exercise point value: 6
File submitted:
C1A8E0_Quiz.txt

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

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. NO CREDIT will be given for anything submitted after the deadline.

-4

Phillip Ward U09339367
Phillip.Ward@seagate.com
C/C++ Programming I
162461 Ray Mitchell
03/06/2022
C1A8E0_Quiz.txt
Quiz Answers

1. C <---E
2. E <---A
3. B <---D
4. B <---E
5. D
6. 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 "Home" page of the course's Canvas website. The assignment checker DOES NOT GRADE your submissions but merely reports on issues so you can avoid credit loss by making corrections and resubmitting. ALL GRADING IS DONE MANUALLY BY THE INSTRUCTOR after the assignment deadline based solely upon the NEWEST submission of each exercise that was submitted BEFORE THE ASSIGNMENT DEADLINE. NO CREDIT will be given for anything submitted after the deadline.

From: Phillip Ward <mailto:phillip.ward@seagate.com>
Subject: C1A8E1_162461_U09339367
Submitted: 3/5/2022 12:23:51 PM PST
Course: C/C++ Programming I (Section 162461)
Student's name: Phillip Ward
Contact email: phillip.ward@seagate.com
Student ID: U09339367
Assignment 8, Exercise 1 (001301924M01005X17301)
Exercise point value: 6
Files submitted:
C1A8E1_main.cpp
C1A8E1_SavingsAccount.cpp
C1A8E1_SavingsAccount.h

"Static analysis" results:
No "static" issues;
"Runtime" results:
Program ran - No errors detected during preliminary testing (SEE ATTACHMENT);

```
//
// Phillip Ward U09339367
// Phillip.Ward@seagate.com
// C/C++ Programming I
// 162461 Ray Mitchell
// 03/03/2022
// C1A8E1_SavingsAccount.h
// Win10
// g++ 11.2.0
//
// Header file for class and function declarations
//
#ifndef C1A8E1_SAVINGSACCOUNT_H
#define C1A8E1_SAVINGSACCOUNT_H
#include <iostream>
const double ONE_HUNDRETH = 0.01;
//class declaration
class SavingsAccount
{
private:
    int type;
    std::string ownerName;
    long IDnbr;
    double balance, closurePenaltyPercent;
public:
    void GetInitialValues(void);
    inline void DisplayValues();
    double CalculatePenalty()
    {
        return(closurePenaltyPercent * balance * ONE_HUNDRETH);
    }
};

inline void SavingsAccount::DisplayValues()
{
    //input SavingsAccount values
    std::cout << "\nAccount type: " << type << "\n";
    std::cout << "Owner name: " << ownerName << "\n";
    std::cout << "ID number: " << IDnbr << "\n";
    std::cout << "Account balance: " << balance << "\n";
    std::cout << "Account closure penalty percent: "
              << closurePenaltyPercent << "\n";
}

#endif
```

```
1  //
2  // Phillip Ward U09339367
3  // Phillip.Ward@seagate.com
4  // C/C++ Programming I
5  // 162461 Ray Mitchell
6  // 03/02/2022
7  // C1A8E1_main.cpp
8  // Win10
9  // g++ 11.2.0
10 //
11 // This file contains a program to input and display bank account attributes
12 //
13
14 #include <iostream>
15 #include "C1A8E1_SavingsAccount.h"
16 using namespace std;
17 int main()
18 {
19     SavingsAccount testAccount;
20     //Initialize the savings account
21     testAccount.GetInitialValues();
22     //Display the bank account values
23     testAccount.DisplayValues();
24     //calculate the penalty
25     double returnVal = testAccount.CalculatePenalty();
26     //display the calculated penalty
27     cout << "Account closure penalty: " << returnVal << "\n";
28     return(0);
29 }
```

```
1  //
2  // Phillip Ward U09339367
3  // Phillip.Ward@seagate.com
4  // C/C++ Programming I
5  // 162461 Ray Mitchell
6  // 03/02/2022
7  // C1A8E1_SavingsAccount.cpp
8  // Win10
9  // g++ 11.2.0
10 //
11 // This file contains a function to initialize a bank account
12 //
13
14 #include <iostream>
15 #include <string>
16 #include "C1A8E1_SavingsAccount.h"
17 using namespace std;
18 void SavingsAccount::GetInitialValues()
19 {
20     //get type
21     cout << "input type: ";
22     cin >> type;
23     //get owner name
24     cout << "input owner name: ";
25     cin >> ws;
26     getline(cin, ownerName);
27     //get ID number
28     cout << "input ID number: ";
29     cin >> IDnbr;
30     //get balance
31     cout << "input balance: ";
32     cin >> balance;
33     //get closer penalty
34     cout << "input penalty percent: ";
35     cin >> closurePenaltyPercent;
36 }
```

```
********** 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 ------------------------------

input type: 80
input owner name: Big Spender
input ID number: 123456789
input balance: .20
input penalty percent: 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 ------------------------------

input type: 9
input owner name: Elvis Clone
input ID number: 2345780
input balance: 300e1
input penalty percent: 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 ------------------------------

input type: 32767
input owner name: Geezer
input ID number: 789
input balance: .40
input penalty percent: 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 "Home" page of the course's Canvas website. The assignment checker DOES NOT GRADE your submissions but merely reports on issues so you can avoid credit loss by making corrections and resubmitting. ALL GRADING IS DONE MANUALLY BY THE INSTRUCTOR after the assignment deadline based solely upon the NEWEST submission of each exercise that was submitted BEFORE THE ASSIGNMENT DEADLINE. NO CREDIT will be given for anything submitted after the deadline.

From: Phillip Ward <mailto:phillip.ward@seagate.com>
Subject: C1A8E2_162461_U09339367
Submitted: 3/6/2022 2:55:53 PM PST
Course: C/C++ Programming I (Section 162461)
Student's name: Phillip Ward
Contact email: phillip.ward@seagate.com
Student ID: U09339367
Assignment 8, Exercise 2 (001485429M01005X81485)
Exercise point value: 8
File submitted:
C1A8E2_main.cpp

"Static analysis" results:
No "static" issues;
"Runtime" results:
Program ran - No errors detected during preliminary testing (SEE ATTACHMENT);

```
1   //
2   // Phillip Ward U09339367
3   // Phillip.Ward@seagate.com
4   // C/C++ Programming I
5   // 162461 Ray Mitchell
6   // 03/05/2022
7   // C1A8E2_main.cpp
8   // Win10
9   // g++ 11.2.0
10  //
11  // A program that finds and replaces user input phrases
12  //
13
14  #include <iostream>
15  #include <fstream>
16  #include <cstring>
17  using namespace std;
18  const int BUF_SIZE = 100;
19  const int IN_FILE_ARG = 1;
20  const int OUT_FILE_ARG = 2;
21  const int SEARCH_ARG = 3;
22  const int REPLACE_ARG = 4;
23  const int TOTAL_ARGS = 5;
24  int main(int argc, char *argv[])
25  {
26      //check for correct # of input args
27      if (argc != TOTAL_ARGS)
28      {
29          cerr << "Incorrect number of input arguments";
30          exit(1);
31      }
32      //declare and open in file
33      ifstream inFile(argv[IN_FILE_ARG]);
34      char charBuf[BUF_SIZE];
35      //check if the input file is open
36      if (inFile.is_open())
37      {
38          //declare and open the out file
39          ofstream outFile(argv[OUT_FILE_ARG]);
40          //check if it opened properly
41          if (outFile.is_open())
42          {
43              size_t jumpTo = strlen(argv[SEARCH_ARG]);
44              //While there are still characters in the infile
45              while (!inFile.eof())
46              {
47                  char *cp1, *cp2;
48                  inFile.getline(charBuf, BUF_SIZE);//get a line at a time
49                  charBuf[strcspn(charBuf, "\n")] = '\0';//strip the newline char
50                  //declare a pointer to the start of the character buffer and
51                  //find the first character of the search string
52                  for (cp1 = charBuf; cp2 = strstr(cp1, argv[SEARCH_ARG]);)
53                  {
54                      //write the beginning of the line
55                      //through the start of the search phrase
56                      outFile.write(cp1, cp2 - cp1);
57                      //Write the replace phrase
58                      outFile << argv[REPLACE_ARG];
59                      //move the pointer to after the replace phrase
60                      cp1 = cp2 + jumpTo;
61                  }
```

When reading a file, an "end of file" condition occurs during the read operation itself if at least the requested number of number of characters are not available. Reading and then using what you "assume" was read without an intervening test is always wrong and the results are unpredictable.

-1

There will not be a \n if the line is read by getline.

-0.3

```
62                  //write the rest of the line
63                  if (!inFile.eof())
64                  {
65                      outFile << cp1 << "\n";
66                  }
67              }
68          }
69          else
70          {
71              cerr << "File: " << argv[OUT_FILE_ARG] << " could not be opened\n";
72              exit(1);
73          }
74          //close the out file
75          outFile.close();
76      }
77      else
78      {
79          cerr << "File: " << argv[IN_FILE_ARG] << " could not be opened\n";
80          exit(1);
81      }
82      //Close the in file
83      inFile.close();
84
85      return(0);
86  }
```

```
********** 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 ------------------------------

File: bad//file//a could not be opened

------------------------------ 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 ------------------------------

File: bad//file//b could not be opened

------------------------------- 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 ------------------------------

Incorrect number of input 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 ------------------------------

Incorrect number of input arguments
------------------------------- END OF 8TH RUN -------------------------------

------------------------------ PURPOSE OF 9TH RUN ------------------------------
Check if the result of opening a file is tested before another file is opened.
---------------------- COMMAND LINE ARGUMENTS FOR 9TH RUN ----------------------
TestFile1.txt  TestFile1_modified1.txt  the  "John Galt?"
------------------------------ START OF 9TH RUN ------------------------------

<<TEST PASSED>>

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