Ray Mitchell, U99999999
MeanOldTeacher@MeanOldTeacher.com
C/C++ Programming I
Section 146359, Ray Mitchell
June 25, 2019
C1A8E0_Quiz.txt
Quiz Answers

1. E
2. A
3. D
4. E
5. D
6. A

C1A8E0 Explanations

In addition to the course book references cited below, these topics are also covered in the live lectures (in-class students) and the recorded lectures (online students).

1. **E** Note 10.1; The FILE data type is defined using a **typedef** statement in the *stdio.h* and *cstdio* standard header files. It represents an implementation-dependent structure type that is used to represent the parameters are necessary to control file operations on a particular implementation.

2. **A** Note 5.11; *savings* is an uninitialized automatic variable and as such, contains garbage. Because it is a structure variable all of its members contain garbage.

3. **D** Notes 10.2, 10.4A; On systems that permit files to be opened in the "text" mode, all files except temporary files will be opened in that mode unless the "binary" mode is explicitly specified. On systems not supporting the text mode all files will be opened in the binary mode. When a file is opened in the text mode the newline character, *'\n'*, is typically translated into the two-character sequence *'\r' '\n'* when written into the file. In the binary mode this translation never occurs and only the *'\n'* character itself is written.

4. **E** Note 10.5; The *ungetc* function is used to push a character back into an input stream for later use, but an implementation is only required to allow the pushback of 1 character in a row. That is, on such an implementation a previously pushed back character must be read before another push back is allowed. However, some implementations permit multiple pushbacks in a row. The *scanf* function internally uses *ungetc* to push back any input character it can't use because it does not match the requirements of the current conversion specification. In the code in this quiz question the first *scanf* reads and accepts the input characters *65FF* because they match the requirements for hexadecimal characters. However, when it then reads the *'z'* it discovers that it does not match those requirements and internally calls *ungetc* to push it back. The next statement in the code explicitly calls *ungetc* to push back the character *'A'*, but since a character has already been pushed back in *scanf* and that character has not yet been read, the new pushback may or may not succeed, depending upon the implementation. Thus, the next character in input stream will either the *z* that the first *scanf* pushed back or the *A* from the explicit ungetc. As a result, what gets output is implementation dependent.

5. **D** Notes 4.3A & 4.3B; If EOF occurs it occurs when a read or write is attempted. Thus, for all read operations it must be checked after that read and before any data that is "presumed" to have been read is actually used. Testing for EOF before the read is attempted or after the data that is "presumed" to have been read has already been used is of no value whatsoever.

6. **B** Notes 7.4, 10.4A, & 10.4B; Any attempt to open a file must be tested for success before the program tries to use that file. The *is_open* method is one of the most thorough ways to accomplish this task, although other options are available.

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A8E1_SavingsAccount.h
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains the definitions of class SavingsAccount and inline
12  // member function DisplayValues, which displays the class object's values.
13  //
14
15  #ifndef C1A8E1_SAVINGSACCOUNT_H
16  #define C1A8E1_SAVINGSACCOUNT_H
17
18  #include <iostream>
19  #include <string>
20
21  const double PERCENT_MULT = .01;
22
23  class SavingsAccount
24  {
25  private:
26     int type;
27     std::string ownerName;
28     long IDnbr;
29     double balance, closurePenaltyPercent;
30  public:
31     void GetInitialValues();
32     void DisplayValues() const;
33     // Calculate and return the penalty incurred when an account is closed.
34     double CalculatePenalty() const
35     {return closurePenaltyPercent * PERCENT_MULT * balance;}
36  };
37
38  //
39  // Display the values in the SavingsAccount object.
40  //
41  inline void SavingsAccount::DisplayValues() const
42  {
43     std::cout <<
44        "Account type: " << type <<
45        "\nOwner name: " << ownerName <<
46        "\nID number: " << IDnbr <<
47        "\nAccount balance: " << balance <<
48        "\nAccount closure penalty percent: " << closurePenaltyPercent << "\n";
49  }
50
51  #endif
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A8E1_main.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains function main, which uses the user-defined
12  // SavingsAccount class to collect and display account information.
13  //
14
15  #include <iostream>
16  #include <cstdlib>
17  #include "C1A8E1_SavingsAccount.h"
18
19  //
20  // Test the SavingsAccount class by creating a SavingsAccount object,
21  // prompting the user for initial values, then displaying the contents
22  // of that object, including the account closure penalty calculation.
23  //
24  int main()
25  {
26     // Exercise the SavingsAccount class by storing and displaying values.
27     SavingsAccount clientA;
28
29     clientA.GetInitialValues();
30     std::cout << "\n";
31     clientA.DisplayValues();
32     std::cout << "Account closure penalty: " << clientA.CalculatePenalty()
33        << '\n';
34
35     return EXIT_SUCCESS;
36  }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A8E1_SavingsAccount.cpp
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains the definition of the GetInitialValues member function
12  // of class SavingsAccount.  Its purpose is to prompt the user for various
13  // values and store them in the class object.
14  //
15
16  #include <iostream>
17  #include <string>
18  using std::cin;
19  using std::cout;
20  using std::ws;
21  #include "C1A8E1_SavingsAccount.h"
22
23  //
24  // Prompt the user for savings account values and store them into the
25  // SavingsAccount object.
26  //
27  void SavingsAccount::GetInitialValues()
28  {
29     // Read in initalization values for the SavingsAccount class and
30     // store them directly into the class object's data members.
31     cout << "Enter account type: ";
32     cin >> type;
33     cout << "Enter owner name: ";
34     cin >> ws;     // skip whitespace (the \n from the previous cin)
35     getline(cin, ownerName);
36     cout << "Enter ID number: ";
37     cin >> IDnbr;
38     cout << "Enter account balance: ";
39     cin >> balance;
40     cout << "Enter account closure penalty percent: ";
41     cin >> closurePenaltyPercent;
42  }
```

```
1   //
2   // Ray Mitchell, U99999999
3   // MeanOldTeacher@MeanOldTeacher.com
4   // C/C++ Programming I
5   // Section 146359, Ray Mitchell
6   // June 25, 2019
7   // C1A8E2_main.c
8   // Windows 10 Professional
9   // Visual Studio 2019 Professional
10  //
11  // This file contains functions main and ErrorOut.  The purpose/operation
12  // of each is described in the block comment preceding it.
13  //
14
15  #include <stdlib.h>
16  #include <stdio.h>
17  #include <string.h>
18
19  #define BUFSIZE 256           // size of input buffer
20  #define ARGUMENTS_EXPECTED 3  // command line args expected
21  #define INFILE_ARG_IX 1       // index of input file name
22  #define LINE_COUNT_ARG_IX 2   // index of line count
23
24  //
25  // Display the string in <myString> followed by a system error message,
26  // then terminate the program with an error code.
27  //
28  void ErrorOut(const char *myString)    // generic "error message and die"
29  {
30     fprintf(stderr, "File \"%s\" didn't open.\n", myString);
31     exit(EXIT_FAILURE);                 // terminate program with failure code
32  }
33  //
34  // Using information obtained from the command line, display the specified
35  // number of lines from the specified text file, then pause.  Each time
36  // the Enter key is pressed the next set of lines is displayed.  The
37  // program terminates when all lines in the file have been displayed or
38  // when a key other than the Enter key is pressed then the Enter key
39  // is pressed.
40  //
41  int main(int argc, char *argv[])
42  {
43     char buf[BUFSIZE];                              // input buffer
44     int lines, maxLines;
45     FILE *fp;                                       // source file pointer
46     char *infileName, *lineCountArg;
47
48     if (argc != ARGUMENTS_EXPECTED)                 // number of cmd. line args
49     {
50        fprintf(stderr, "Syntax is: pgmName fileName lineCount\n");
51        exit(EXIT_FAILURE);
52     }
53     infileName = argv[INFILE_ARG_IX];
54     lineCountArg = argv[LINE_COUNT_ARG_IX];
55
56     if ((fp = fopen(infileName, "r")) == NULL)      // open input file
57        ErrorOut(infileName);                        // open error
58     maxLines = atoi(lineCountArg);                  // get the integer value
59     for (lines = 0; fgets(buf, BUFSIZE, fp) != NULL;)//lint !e668
60     {
61        fputs(buf, stdout);                          // print it to stdout
```

```
62          if (buf[strlen(buf) - 1] == '\n')            // end of full line
63          {
64             if (++lines == maxLines)                  // if page is full
65             {
66                if (getchar() != '\n')                 // if terminate program
67                   break;                              // exit loop
68                lines = 0;                             // lines on page count = 0
69             }
70          }
71       }
72       fclose(fp);                                      // close input file
73
74       return EXIT_SUCCESS;
75    }
```

```
//
// Ray Mitchell, U99999999
// MeanOldTeacher@MeanOldTeacher.com
// C/C++ Programming I
// Section 146359, Ray Mitchell
// June 25, 2019
// C1A8E3_main.cpp
// Windows 10 Professional
// Visual Studio 2019 Professional
//
// This file contains functions main and ErrorOut.  The purpose/operation
// of each is described in the block comment preceding it.
//

#include <cstdlib>
#include <cstring>
#include <fstream>
#include <iostream>

const int ARGUMENTS_EXPECTED = 5;   // command line args expected
const int INFILE_ARG_IX = 1;        // index of input file name
const int OUTFILE_ARG_IX = 2;       // index of output file name
const int SEARCHSTRING_ARG_IX = 3;  // index of string to find
const int NEWSTRING_ARG_IX = 4;     // index of replacement string
const int BUFSIZE = 256;            // size of input buffer

//
// Display the string in <myString> followed by a system error message,
// then terminate the program with an error code.
//
void ErrorOut(const char *myString) // generic "error message and die"
{
   std::cerr << "File \"" << myString << "\" didn't open!\n";
   std::exit(EXIT_FAILURE);         // terminate program
}

//
// Create a modified version of the text file named in argv[INFILE_ARG_IX]
// giving it the name in argv[OUTFILE_ARG_IX].  The new file will contain
// the input file's original text except that all occurrences of the
// character sequence specified in argv[SEARCHSTRING_ARG_IX] will be
// replaced by the character sequence in argv[NEWSTRING_ARG_IX].
//
int main(int argc, char *argv[])
{
   if (argc != ARGUMENTS_EXPECTED)        // number of cmd. line args
   {
      std::cerr <<
         "Syntax is: pgmName infileName outfileName searchString newString\n";
      exit(EXIT_FAILURE);
   }

   char *infileName = argv[INFILE_ARG_IX];
   char *outfileName = argv[OUTFILE_ARG_IX];
   char *searchString = argv[SEARCHSTRING_ARG_IX];
   char *newString = argv[NEWSTRING_ARG_IX];

   std::ifstream inFile(infileName);      // open input file
   if (!inFile.is_open())                 // test the opening
      ErrorOut(infileName);               // open error
   std::ofstream outFile(outfileName);    // open output file
```

```
62      if (!outFile.is_open())                 // test the opening
63         ErrorOut(outfileName);               // open error
64
65      int searchStringLength = (int)std::strlen(searchString);
66      char lineBuf[BUFSIZE];                  // input line buffer
67      while (inFile.getline(lineBuf, BUFSIZE))
68      {
69         // Find substring to replace.
70         char *cp1, *cp2;
71         for (cp1 = lineBuf; cp2 = std::strstr(cp1, searchString);)
72         {
73            // copy up to substring
74            outFile.write(cp1, std::streamsize(cp2 - cp1));
75            outFile << newString;             // write substitute string
76            cp1 = cp2 + searchStringLength;   // move ahead in source string
77         }
78         outFile << cp1 << '\n';              // copy remainder of line
79      }
80      inFile.close();                         // close input file
81      outFile.close();                        // close output file
82
83      return EXIT_SUCCESS;
84   }
```