

Exercise 0 (6 points – 1 point per question – No program required)

1. C
2. C
3. B
4. C
5. E
6. E

References and 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. Note 9.14B; Before a non-**static** method of a class can be called an object of that class must exist. Such objects can be created by simple declarations or dynamically, just like any other objects. Then that object can be used to access the public members by using either the dot operator or the arrow operator, as appropriate.
2. Note 4.3A; EOF is required to be of type **int** and have a negative value, but any assumptions about what that negative value actually is are non-portable. Although not required to do so, virtually all common implementations represent type **char** using fewer bits than type **int**, which results in the inability of type **char** to represent the full bit pattern used to represent EOF. Because of this and because implementations are free to implement type **char** as either **signed char** or **unsigned char**, the result of trying to compare a value represented as type **char** with the value of EOF is unpredictable and unreliable. If this is attempted false indications of equality can occur if type **char** is implemented as a **signed char** while false indications of inequality can occur if type **char** is implemented as an **unsigned char**.
3. 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.
4. Notes 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.
5. 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.
6. Note 10.5; The *ungetc* function is used to push a character back into an open input stream so it can be read later. 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 character is pushed back or the additional pushbacks will fail. However, some implementations permit the pushback of multiple characters in a row. The *scanf* function uses *ungetc* to push back any character it reads from an input stream but 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 the 23a portion of the 23abz input string in an attempt to form a decimal integral value for its conversion specification, %d. However, it finds that the 'a' is not a valid character for a decimal integral value so it internally calls *ungetc* to push it back. The next

1 statement in the code explicitly calls *ungetc* to push back the character 'a'. However, since a
2 character has already been pushed back by the *scanf* and that character has not yet been
3 read, the new pushback may or may not fail, depending upon the implementation. Thus, the
4 input stream will either contain *abz* or *aabz* and the following *scanf* will read either *ab* or *aab*.

Exercise 1 (4 points – C++ Program)

```
***** FILE C1A8E1_SavingsAccount.h *****
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains the definitions of class SavingsAccount and inline
// member function DisplayValues, which displays the class object's values.
//
#ifndef C1A8E1_SAVINGSACCOUNT_H
#define C1A8E1_SAVINGSACCOUNT_H

#include <iostream>
#include <string>

const double PERCENT_MULT = .01;

class SavingsAccount
{
private:
    int accountType;
    std::string ownerName;
    long IDnbr;
    double accountBalance, accountClosurePenaltyPercent;
public:
    void GetInitialValues();
    void DisplayValues() const;
    // Calculate and return the penalty incurred when an account is closed.
    double CalculatePenalty() const
    { return accountClosurePenaltyPercent * PERCENT_MULT * accountBalance; };
};

//
// Display the values in the SavingsAccount object.
//
inline void SavingsAccount::DisplayValues() const
{
    std::cout << "Account type is: " << accountType << "\nOwner name is: "
        << ownerName << "\nID number is: " << IDnbr << "\nAccount balance is: "
        << accountBalance << "\nAccount closure penalty percent is: "
        << accountClosurePenaltyPercent << '\n';
}

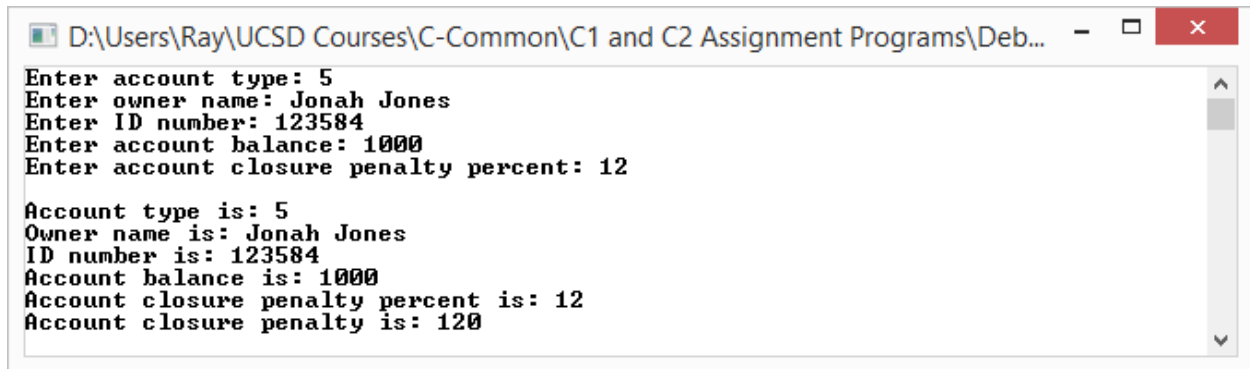
#endif

***** FILE C1A8E1_SavingsAccount.cpp *****
//
// ...the usual title block Student/Course/Assignment/Compiler information goes here...
//
// This file contains the definition of the GetInitialValues member function
// of class SavingsAccount. Its purpose is to prompt the user for various
// values and store them in the class object.
//
#include <iostream>
```

```
1  #include <string>
2  using std::cin;
3  using std::cout;
4  using std::ws;
5  #include "C1A8E1_SavingsAccount.h"
6
7  //
8  // Prompt the user for savings account values and store them into the
9  // SavingsAccount object.
10 //
11 void SavingsAccount::GetInitialValues()
12 {
13     // Read in initialization values for the SavingsAccount class and
14     // store them directly into the class object's data members.
15     cout << "Enter account type: ";
16     cin >> accountType;
17     cout << "Enter owner name: ";
18     cin >> ws;    // skip whitespace (the \n left from the previous cin)
19     getline(cin, ownerName);
20     cout << "Enter ID number: ";
21     cin >> IDnbr;
22     cout << "Enter account balance: ";
23     cin >> accountBalance;
24     cout << "Enter account closure penalty percent: ";
25     cin >> accountClosurePenaltyPercent;
26 }
27
28 ***** FILE C1A8E1_main.cpp *****
29 //
30 // ...the usual title block Student/Course/Assignment/Compiler information goes here...
31 //
32 // This file contains function main, which uses the user-defined
33 // SavingsAccount class to collect and display account information.
34 //
35
36 #include <iostream>
37 #include <cstdlib>
38 #include "C1A8E1_SavingsAccount.h"
39
40 //
41 // Test the SavingsAccount class by creating a SavingsAccount object,
42 // prompting the user for initial values, then displaying the contents
43 // of that object, including the account closure penalty calculation.
44 //
45 int main()
46 {
47     // Exercise the SavingsAccount class by storing and displaying values.
48     SavingsAccount clientA;
49
50     clientA.GetInitialValues();
51     std::cout << "\n";
52     clientA.DisplayValues();
53     std::cout << "Account closure penalty is: " << clientA.CalculatePenalty() << '\n';
54
55     return EXIT_SUCCESS;
56 }
```

C1A8E1 Screen Shots are on the next page...

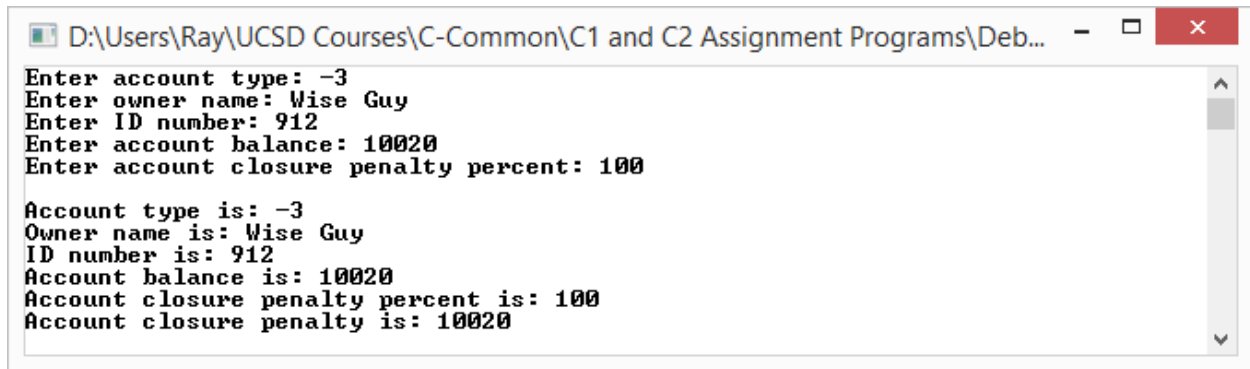
C1A8E1 Screen Shots



A screenshot of a Windows application window titled "D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Deb...". The window contains a text area with the following text:

```
Enter account type: 5
Enter owner name: Jonah Jones
Enter ID number: 123584
Enter account balance: 1000
Enter account closure penalty percent: 12

Account type is: 5
Owner name is: Jonah Jones
ID number is: 123584
Account balance is: 1000
Account closure penalty percent is: 12
Account closure penalty is: 120
```



A screenshot of a Windows application window titled "D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Deb...". The window contains a text area with the following text:

```
Enter account type: -3
Enter owner name: Wise Guy
Enter ID number: 912
Enter account balance: 10020
Enter account closure penalty percent: 100

Account type is: -3
Owner name is: Wise Guy
ID number is: 912
Account balance is: 10020
Account closure penalty percent is: 100
Account closure penalty is: 10020
```

Exercise 2 (4 points – C Program)

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

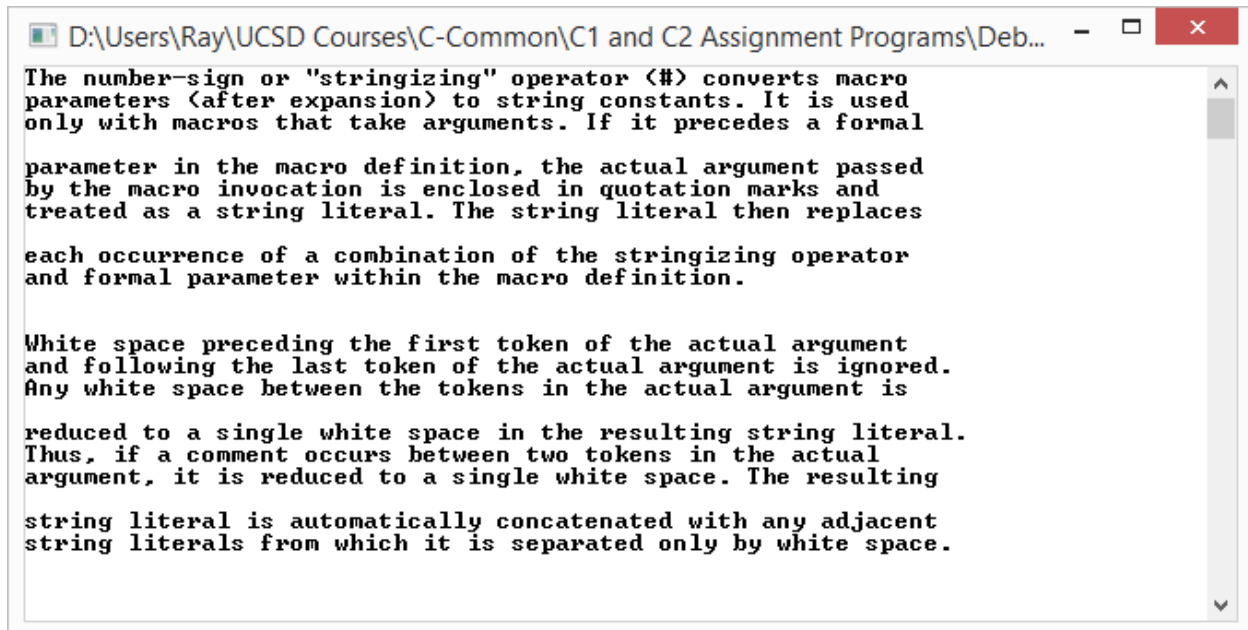
```

1      if (++lines == maxLines)          /* page is full */
2      {
3          if (getchar() != '\n')        /* if terminate program */
4          {
5              fputs("\n<<Program terminated...>>\n", stdout);
6              break;                    /* exit for loop */
7          }
8          lines = 0;                    /* lines on page count = 0 */
9      }
10     }
11 }
12 fclose(fp);                          /* close input file */
13
14 return EXIT_SUCCESS;
15 }

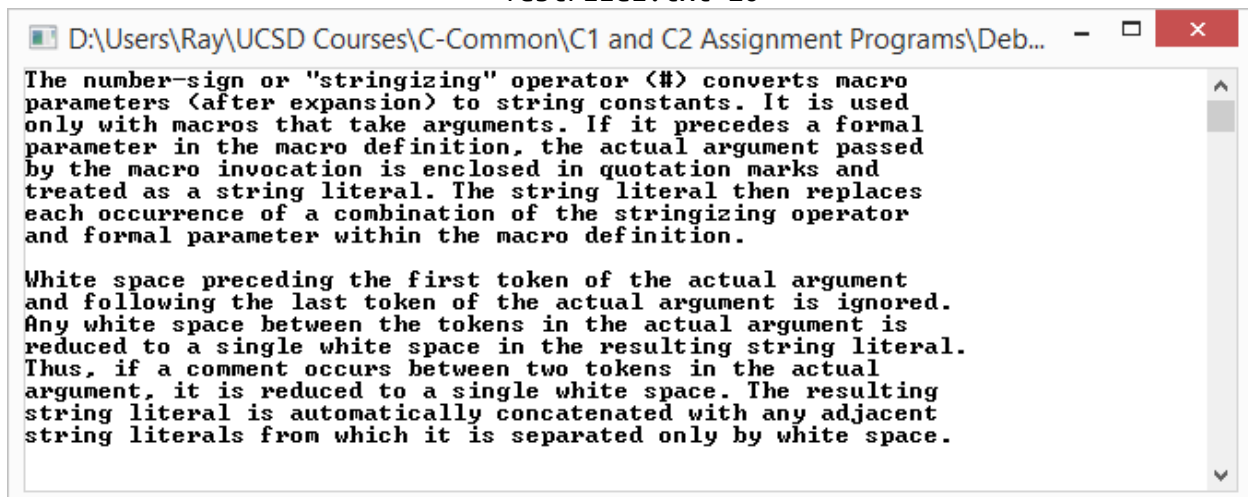
```

C1A8E2 Screen Shots

TestFile1.txt 3



TestFile1.txt 20



Exercise 3 (6 points – C++ Program)

```
1 //
2 // ...the usual title block Student/Course/Assignment/Compiler information goes here...
3 //
4 // This file contains functions:
5 //   main: Replaces all occurrences of command line specified string with
6 //         another command line specified string within a command line specified
7 //         text file;
8 //   ErrorOut: Terminates the program with an error message and error code;
9 //
10
11
12
13 #include <cstdlib>
14 #include <cstring>
15 #include <fstream>
16 #include <iostream>
17
18 const int ARGUMENTS_EXPECTED = 5; // command line args expected
19 const int INFILE_ARG_IX = 1;      // index of input file name
20 const int OUTFILE_ARG_IX = 2;     // index of input file name
21 const int SEARCHSTRING_ARG_IX = 3; // index of string to find
22 const int NEWSTRING_ARG_IX = 4;   // index of replacement string
23 const int BUFSIZE = 256;          // size of input buffer
24
25 //
26 // Display the string in <myString> followed by a system error message,
27 // then terminate the program with an error code.
28 //
29 void ErrorOut(const char *myString) // generic "error message and die"
30 {
31     std::cerr << "File \"" << myString << "\" didn't open!\n";
32     std::exit(EXIT_FAILURE);        // terminate program
33 }
34
35 //
36 // Create a modified version of the text file named in argv[INFILE_IX]
37 // giving it the name in argv[OUTFILE_IX]. The new file will contain
38 // the input file's original text except that all occurrences of the
39 // character sequence specified in argv[SEARCHSTRING_IX] will be
40 // replaced by the character sequence in argv[NEWSTRING_IX].
41 //
42 int main(int argc, char *argv[])
43 {
44     const char * const INFILE_NAME = argv[INFILE_ARG_IX];
45     const char * const OUTFILE_NAME = argv[OUTFILE_ARG_IX];
46     const char * const SEARCH_STRING = argv[SEARCHSTRING_ARG_IX];
47     const char * const NEW_STRING = argv[NEWSTRING_ARG_IX];
48
49     if (argc != ARGUMENTS_EXPECTED) // number of cmd. line args
50     {
51         std::cerr <<
52             "Syntax is: pgmName inFileName outFileName searchString newString\n";
53         exit(EXIT_FAILURE);
54     }
55
56     char lineBuf[BUFSIZE];          // input line buffer
57     int searchStringLength;          // length of search substring
```



```
1  std::ifstream inFile;           // input file object
2  std::ofstream outFile;          // output file object
3
4  inFile.open(INFILE_NAME);        // open input file
5  if (!inFile.is_open())           // test the opening
6      ErrorOut(INFILE_NAME);       // open error
7  outFile.open(OUTFILE_NAME);      // open output file
8  if (!outFile.is_open())          // test the opening
9      ErrorOut(OUTFILE_NAME);      // open error
10
11  searchStringLength = (int)std::strlen(SEARCH_STRING);
12  while (inFile.getline(lineBuf, BUFSIZE))
13  {
14      // Find substring to replace.
15      char *inPtr, *cp;
16      for (inPtr = lineBuf; cp = std::strstr(inPtr, SEARCH_STRING);)
17      {
18          // copy up to substring
19          outFile.write(inPtr, std::streamsize(cp - inPtr));
20          outFile << NEW_STRING;    // write substitute string
21          inPtr = cp + searchStringLength; // move ahead in source string
22      }
23      outFile << inPtr << '\n';    // copy remainder of line
24  }
25  inFile.close();                  // close input file
26  outFile.close();                 // close output file
27
28  return EXIT_SUCCESS;
29 }
```

C1A8E3 Input/Output File Contents

Original

```
1 The number-sign or "stringizing" operator (#) converts macro
2 parameters (after expansion) to string constants. It is used
3 only with macros that take arguments. If it precedes a formal
4 parameter in the macro definition, the actual argument passed
5 by the macro invocation is enclosed in quotation marks and
6 treated as a string literal. The string literal then replaces
7 each occurrence of a combination of the stringizing operator
8 and formal parameter within the macro definition.
9
10 White space preceding the first token of the actual argument
11 and following the last token of the actual argument is ignored.
12 Any white space between the tokens in the actual argument is
13 reduced to a single white space in the resulting string literal.
14 Thus, if a comment occurs between two tokens in the actual
15 argument, it is reduced to a single white space. The resulting
16 string literal is automatically concatenated with any adjacent
17 string literals from which it is separated only by white space.
18
19
```

C1A8E3 Input/Output File Contents continues on the next page...

C1A8E3 Input/Output File Contents, continued

the replaced with *Who is John Galt?*

```
1 |The number-sign or "stringizing" operator (#) converts macro
2 |parameters (after expansion) to string constants. It is used
3 |only with macros that take arguments. If it precedes a formal
4 |parameter in Who is John Galt? macro definition, Who is John Galt? actual argument passed
5 |by Who is John Galt? macro invocation is enclosed in quotation marks and
6 |treated as a string literal. The string literal Who is John Galt?n replaces
7 |each occurrence of a combination of Who is John Galt? stringizing operator
8 |and formal parameter within Who is John Galt? macro definition.
9 |
10 |White space preceding Who is John Galt? first token of Who is John Galt? actual argument
11 |and following Who is John Galt? last token of Who is John Galt? actual argument is ignored.
12 |Any white space between Who is John Galt? tokens in Who is John Galt? actual argument is
13 |reduced to a single white space in Who is John Galt? resulting string literal.
14 |Thus, if a comment occurs between two tokens in Who is John Galt? actual
15 |argument, it is reduced to a single white space. The resulting
16 |string literal is automatically concatenated with any adjacent
17 |string literals from which it is separated only by white space.
18 |
19 |
```

string literal replaced with *TESTING*

```
1 |The number-sign or "stringizing" operator (#) converts macro
2 |parameters (after expansion) to string constants. It is used
3 |only with macros that take arguments. If it precedes a formal
4 |parameter in the macro definition, the actual argument passed
5 |by the macro invocation is enclosed in quotation marks and
6 |treated as a TESTING. The TESTING then replaces
7 |each occurrence of a combination of the stringizing operator
8 |and formal parameter within the macro definition.
9 |
10 |White space preceding the first token of the actual argument
11 |and following the last token of the actual argument is ignored.
12 |Any white space between the tokens in the actual argument is
13 |reduced to a single white space in the resulting TESTING.
14 |Thus, if a comment occurs between two tokens in the actual
15 |argument, it is reduced to a single white space. The resulting
16 |TESTING is automatically concatenated with any adjacent
17 |TESTINGs from which it is separated only by white space.
18 |
19 |
```