

Assignment 8

C/C++ Programming I

C1A8 General Information

Where Does a Program Look for Files When Attempting to Open Them? Where Does a Program Create New Files? Where Should You Put Instructor-Supplied Data Files?

What is a "Working Directory"?

A program's "Working Directory" is the directory it uses for any files it opens or creates if their names are specified without a path, and you must place any instructor-supplied data file(s) (.txt or .bin extensions) your program needs in that directory. Its default location differs between IDEs and operating systems and it is important to know where it is and how to change it. For further information please refer to the **Determining/Changing the "Working Directory"** topic in the version of the course document titled "Using the Compiler's IDE..." that is applicable to the IDE you are using.

Opening Files – Testing for Failure/Success

Always check the success/failure status of opening a file before using it or opening another file.

Supplying Information to a Program via its "Command Line"

It is often more appropriate to supply information to a program via "command line arguments" than by user prompts. Such arguments can be provided regardless of how a program is being run, whether it be from within an IDE, a system command window, a GUI icon, or a batch file. For this course I strongly recommend using an IDE for running all programs.

If you are not familiar with using command line arguments first review note 8.3 for information on how to process them within any program, then review the appropriate version of the course document titled "Using the Compiler's IDE...", which illustrates implementing an arbitrary command line in several ways including implementing command arguments containing spaces.

It is important to note that command line redirection information (note 4.2), if any, is only visible to the operating system and will not be among the command line arguments available to the program being run.

Get a Consolidated Assignment 8 Report (optional)

If you would like to receive a consolidated report containing the results of the most recent version of each exercise submitted for this assignment:

Send an empty-body email to the assignment checker with the subject line **C1A8_162461_U09339367** and no attachments.

Inspect the report carefully since it is what I will be grading. You may resubmit exercises and report requests as many times as you wish before the assignment deadline.

C1A8E0 (6 points total - 1 point per question – No program required)

Assume language standards compliance and any necessary standard library support unless stated otherwise. These are not trick questions and there is only one correct answer, but basing an answer on runtime results is risky. Place your answers in a plain text "quiz file" named **C1A8E0_Quiz.txt** formatted as:

a "Non-Code" Title Block, an empty line, then the answers:

- 1. A
- 2. C
- etc.

1. *stdin*, *stdout*, and *stderr*:
(Note 10.1)
 - A. may cause a compiler error in C if *using namespace std*; is not declared first.
 - B. are the names of three standard files.
 - C. are all of data type *FILE*
 - D. may not all be of data type *FILE* *
 - E. none of the above
2. What is the most serious problem?

```
struct Savings {  
    double principal, interest;  
    public:  
        double total()  
        {return(principal + interest);}  
};  
Savings savings;  
cout << "Total account value is "  
    << savings.total() << "\n"  
    "Interest = " << savings.interest;
```


(Note 5.11)
 - A. Garbage values will be output.
 - B. *total* is not an **inline** function.
 - C. *total* accesses private data members.
 - D. *total* is not called with a *Savings* object.
 - E. *savings.interest* is illegal.
3. If *ofstream* *oFile*("myFile"); succeeds, how many bytes does *oFile* << "/n/n/n" write into the file?
(Note 10.2)
 - A. 0
 - B. 3 or 4
 - C. 4 or 5
 - D. 5 or 6
 - E. 3 or 6
4. Noting that *scanf* calls *ungetc* if necessary, what gets printed if *xyz* is type **int** and 65FFz is entered from the keyboard?

```
scanf("%x", &xyz);  
ungetc('A', stdin);  
scanf("%x", &xyz);  
printf("%x", xyz);
```


(Note 10.5)
 - A. 65FF
 - B. a
 - C. FF
 - D. nothing is printed
 - E. Implementations may differ on this.
5. What is the most serious problem if the intent is to output only the characters in the file represented by *inFile*?

```
int ch;  
do  
    cout << (char)(ch = inFile.get());  
while (ch != EOF);
```


(Notes 4.3A & 4.3B)
 - A. EOF must be tested using *inFile.eof()*.
 - B. *get* is not capable of returning EOF.
 - C. **while** (ch != EOF) may not detect EOF at all or false EOF indications may occur.
 - D. EOF is checked in the wrong place.
 - E. *ch* is type **int** but *get* returns type **char**.
6. What is the most serious problem if the intent is to write to the file?

```
ofstream oFile("myname");  
oFile.write("ABCD", 4);
```


(Notes 7.4, 10.4A, & 10.4B)
 - A. The open's success/failure is not tested.
 - B. *oFile.write*("ABCD", 4); is not legal syntax.
 - C. The write operation will always fail.
 - D. More than 4 characters might be written.
 - E. The write should be *oFile* << "ABCD";

Submitting your solution

Send an empty-body email to the assignment checker with the subject line **C1A8E0_162461_U09339367** and with your quiz file attached.

See the course document titled "How to Prepare and Submit Assignments" for additional exercise formatting, submission, and assignment checker requirements.

C1A8E1 (6 points – C++ Program)

Exclude any existing source code files that may already be in your IDE project and add three new ones, naming them **C1A8E1_SavingsAccount.h**, **C1A8E1_SavingsAccount.cpp**, and **C1A8E1_main.cpp**. Do not use **#include** to include either of the two **.cpp** files in each other or in any other file. However, you may use it to include any appropriate header file(s) you need. You must include **C1A8E1_SavingsAccount.h** in any file that needs its contents.

This exercise is not an example of rigorous banking practices but is only to familiarize you with simple C++ **class** members. Assume the **class** below represents just the data members of a savings account:

```
class SavingsAccount    // Do not change the order or access level of this class's data members
{
    int type;
    string ownerName;    // You must handle the case where the name contains whitespace
    long IDnbr;
    double balance, closurePenaltyPercent;
};
```

Arrays and loops are neither necessary nor allowed anywhere in this exercise.

File **C1A8E1_SavingsAccount.h** must contain the following:

- the **class SavingsAccount** definition from above with the following added to its contents:
 - the prototype only for member function **GetInitialValues**
 - the prototype only for member function **DisplayValues**
 - the entire definition of member function **CalculatePenalty**
- the entire definition of function **DisplayValues** – placed outside **class SavingsAccount**

File **C1A8E1_SavingsAccount.cpp** must contain:

- the entire definition of member function **GetInitialValues**

File **C1A8E1_main.cpp** must contain function **main**, which must:

- declare a variable of type **SavingsAccount** (do not use dynamic allocation).
- call functions **GetInitialValues**, **DisplayValues**, and **CalculatePenalty** in that order. You may respond to any prompts with whatever values you deem sufficient to test your program.
- display the value returned from **CalculatePenalty** using the following format, where the question mark represents the actual value:
Account closure penalty: ?

GetInitialValues: a non-inline member function, which must:

- have a **void** parameter list and return **void**;
- prompt the user for 5 values and initialize the 5 **class** data members to those values. The user may enter any values that can be represented in the members' data types. Prompt for each value separately and in the same order as the members appear in the class.

DisplayValues: a const inline member function, which must:

- have a **void** parameter list and return **void**
- not prompt for anything
- use one **cout** to display the values of all data members in the same order as they appear in the **class**, using the following format, where the question marks represent the actual values:
Account type: ?
Owner name: ?
ID number: ?
Account balance: ?
Account closure penalty percent: ?

1 **CalculatePenalty**: a const inline member function, which must:

- 2 1. have a **void** parameter list and return an appropriate type
- 3 2. calculate the account closure penalty and return it to the caller. It will be the percent specified
- 4 by **closurePenaltyPercent** of the account balance specified by **balance**. Do not use
- 5 division.
- 6 3. not prompt or display anything

9 Manually re-run your program several times, testing with different values.

12 Submitting your solution

13 `Send an empty-body email to the assignment checker with the subject line **C1A8E1_162461_U09339367**

14 and with all three source code files attached.

15 *See the course document titled "How to Prepare and Submit Assignments" for additional exercise*

16 *formatting, submission, and assignment checker requirements.*

19 Hints:

- 20 1. If you attempt to read the account owner's name from user input using the >> operator, your
- 21 program will fail if the name contains whitespace. This is because that operator stops reading a
- 22 string when the first non-leading whitespace is encountered. The correct approach is to use the
- 23 **getline** function (note 7.7).
- 24
- 25 2. See note 4.1. Some operations that read newline-terminated input leave the newline character
- 26 in the input buffer, which then acts as leading whitespace for a subsequent read. If that
- 27 character is still there when the **getline** function is called, only it will be read and the line will be
- 28 considered empty. Use either **cin >> ws** or **cin.ignore()** before calling **getline** to remove
- 29 and discard the newline character. **ws** is not a variable but is instead what is known as a
- 30 "manipulator". It is in the **std** namespace and is available when you include standard C++
- 31 header file **iostream**. It is not necessary to include standard C++ header file **omanip** to use the
- 32 **ws** manipulator.
- 33
- 34 3. 1.3% of 0.2 is not 0.26
- 35
- 36 4. Be sure to use "include guards" (note D.2) in header file **C1A8E1_SavingsAccount.h**. Use
- 37 **#include** to include this file in files **C1A8E1_SavingsAccount.cpp** and **C1A8E1_main.cpp**.
- 38
- 39 5. If you get a "multiply defined" linker error regarding function **DisplayValues**, it is because it has
- 40 not been declared as "inline".

C1A8E2 (8 points – C++ Program)

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C1A8E2_main.cpp**. Write a program in that file to perform a "Search and Replace All" operation. This operation consists of performing a case-sensitive search for all occurrences of an arbitrary sequence of characters within a file and substituting another arbitrary sequence in place of them.

One of the biggest problems some students have with this exercise occurs simply because they do not read the command line information I have provided in the appropriate version of the course document titled "Using the Compiler's IDE...". Aside from this, I recommend (but do not require) that you implement the algorithm provided in the hints.

Your program:

- May assume, for simplicity:
 - It will only be used with text files.
 - The input and output file names will be different.
 - No character sequence will be split across multiple lines.
- Must support multiple occurrences of the "search" character sequence on a single line.
- Must support cases where the length of the "search" character sequence is different from the length of the "replacement" character sequence.
- Must get the following information from the command line in the order listed. Spaces must be allowed in all items:
 - The name of the file to search (the input file).
 - The name of the file to store the results in (the output file).
 - The sequence of characters to search for.
 - The sequence of characters to use as replacements.
- Must verify that exactly the correct number of command arguments exist before accessing them and must exit with an error message and error code if they do not.
- Must read one line at a time from the input file into a character buffer and process it completely before reading the next line.
- Must not copy anything from the character buffer to any destination other than the output file.
- Must not use data types **list**, **queue**, **stack**, **string**, or **vector**.
- Must not use functions **peek**, **seekp**, **seekg**, **tellp**, **tellg**, **fseek**, **ftell**, **fsetpos**, or **fgetpos**.
- Must not call the **strlen** function more than once for the same string, including inside a loop.
- Must not declare more than one array or use dynamic memory allocation.
- Must not test for an empty input file or one that does not contain the search sequence.

Manually re-run your program several times, testing at least the following 2 cases with instructor-supplied data file **TestFile1.txt**, which must be placed in the program's "working directory". Choose any name you wish for the output file as long as it is different from the input file name:

Test#	Input File	Output File	Search For	Replace Each With
1	<i>TestFile1.txt</i>	your choice	<i>the</i>	<i>John Galt?</i>
2	<i>TestFile1.txt</i>	your choice	<i>string literal</i>	<i>TESTING</i>

For example (Test #1), if the input file contained,

These are the answers to neither of their questions!

the output file would contain,

These are John Galt? answers to neiJohn Galt?r of John Galt?ir questions!

Note that there were three replacements on one line and that "the" was part of another word in two of those cases.

Submitting your solution

Send an empty-body email to the assignment checker with the subject line **C1A8E2_162461_U09339367** and with your source code file attached.

See the course document titled "How to Prepare and Submit Assignments" for additional exercise formatting, submission, and assignment checker requirements.

Hints:

1. The value of EOF cannot be correctly stored or detected using data type **char** or **unsigned char**; use type **int** instead (Notes 4.3A & 4.3B). Since the end of file condition only occurs when a read (or write) is attempted, testing for it before such an attempt is meaningless and inappropriate. Only test after such an attempt before the value obtained from reading is actually used (Note 4.3B).
2. Not testing files for a successful opening is bad programming (Notes 10.3 & 10.4B).
3. It is not necessary to determine the length of the replacement string.

You may use any algorithm you wish to complete this exercise as long as none of the requirements/restrictions are violated. The following describes the algorithm I used and recommend. Drawing a diagram of memory as the algorithm progresses always helps:

General Algorithm Description:

For each line read into a character buffer from the input file

```
{  
    Set a pointer to first character in line.  
    For each "search" sequence found in line using the strstr library function  
    {  
        Copy characters into the output file from the pointer to where the "search" sequence starts.  
        Write the replacement string into the output file.  
        Move the pointer to the next character in the input line after the "search" sequence ends.  
    }  
    Copy remainder of line into the output file.  
}
```

Algorithm Step-by-Step Implementation Details:

1. Do all the standard things, including declaring needed variables and opening the input and output files.
2. Implement a loop statement that contains the following 3 statements (**a**, **b**, and **c**) in order:
 - a. a statement that gets the next line from the input file and stores it into a character buffer. (The newline character must be discarded.) If the end of file condition occurs, terminate the loop. Otherwise, proceed to step **b** below.
 - b. a "for" statement that does everything in steps **i** and **ii** below (in order):
 - i. does the following in its "control" section. The "control" section is the portion of the "for" statement that is in parentheses just after the keyword **for**:
 - 1) The "initial expression" (Note 3.5) initializes a character pointer (I'll call it **cp1**) to point to the beginning of the character buffer you are reading each line into.
 - 2) The "controlling expression" (Note 3.5) assigns the return value of the **strstr** function to a different character pointer (I'll call it **cp2**). The first argument of **strstr** will be **cp1** and the second argument will be a pointer to the first character of the string you are looking for in the file.
 - 3) The "loop expression" (Note 3.5) is empty.
 - ii. does the following in the loop "body":
 - 1) Uses the **write** function to write to the output file. The first argument of **write** will be **cp1** and the second will be **cp2-cp1**.
 - 2) Uses the **<<** operator to write the replacement string to the output file.
 - 3) Updates **cp1** by assigning to it the sum of **cp2** and the length of the string you are searching for in the file.
 - c. a statement that writes the string in **cp1** and a newline character to the output file.