Assignment 3
C/C++ Programming II

C2A3 General Information

Assignment 3 consists of FOUR (4) exercises:

C2A3E1 C2A3E2 C2A3E3 C2A3E4

All requirements are in this document.

Get a Consolidated Assignment 3 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 **C2A3_177752_U09845800** 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.

Redirection of Input and/or Output

Some exercises in this course require the use of input and/or output redirection. Please see note 4.2 of the course book for an explanation of this concept. Redirection information must be placed on the program's "command line" (see the section below).

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 recommend using the IDE for this purpose. 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.

C2A3E1 *(2 points – Short answers only – No program required)*

Place your answers in a plain text file named **C2A3E1_Sentences.txt**.

Assume the two declarations and ten code snippets below:

```
double film[46];
double *x;

 1.  x = film
 2.  x = (double *)sizeof( film [1])
 3.  sizeof( film ) / printf("sizeof")
 4.  sizeof(32L + 50LLu + film )
 5.  5 + (int)x[cin.get() + 18] + & film
 6.  (double &) film
 7.  sizeof(& film [10])
 8.  & film [(int)x]
 9.  & film - sizeof(0[x])
10.  film - sizeof(unsigned short)
```

Create a numbered sentence for each snippet that describes the type that identifier `film` is treated as by the compiler in that snippet. The only two possibilities are the sentences produced by using the Right-Left Rule (note 6.1) and Decayed Right-Left Rule (note 7.13) on the original declaration. Every sentence must begin with the words "`film` is" or "`film` decays to", as appropriate. IMPORTANT: I am asking only for the type of `film` (the underlined part), not the type of the entire snippet. For example, the correct answer for the first snippet is:

1. film decays to a pointer to a double.

Some of the code snippets are impractical or might cause runtime errors, but they all are syntactically correct. For this exercise there is no reason to attempt to compile any of them.

- Use literal numbers (not the words for them) in your sentences if numeric values are needed.
- Do not place anything on the answer lines except the sentence number and the sentence.
- Do not place anything in between the answer lines.
- If you have questions or comments, place them after the entire group of answer lines.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C2A3E1_177752_U09845800** and with your text file attached.

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

Hints:

1. Refer to notes 6.1, 6.16, 7.13, and A.1.
2. Unnecessary whitespace between tokens means nothing to the compiler.

C2A3E2 *(4 points – C++ Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C2A3E2_TestDeclarations.cpp**. Also add instructor-supplied source code file **C2A3E2_main-Driver.cpp**. Do not write a `main` function! `main` already exists in the instructor-supplied file and it will use the code you write.

This exercise:

- Requires absolutely no knowledge of what the data types being described actually mean.
- Is trivial if you understand notes 13.2-13.3B but can be extremely difficult if you do not.

The Reverse Right-Left rule (notes 13.3A & 13.3B) is used to convert meaningful English sentences that describe standard C and C++ declarations and type casts into code. For example,

a sentence that reads	`test` is an **int**.
gets coded as	`int test;`
a sentence that reads	This is a C-style type cast to a pointer to a **double**.
gets coded as	`(double *)`
and can be used with the syntax	`(double *)test`

File **C2A3E2_TestDeclarations.cpp** must contain a function named `TestDeclarations`.

Just before your definition of function `TestDeclarations`, define an appropriately named and typed **const** variable and use it in your code to avoid magic numbers.

`TestDeclarations` syntax:

```
void TestDeclarations();
```

Parameters:

none

Synopsis:

Contains exactly five single statements, in order, that do the following. Only initialize if so stated:

1. Declares `aryP` to be "a pointer to an array of 6 pointers to **void**" and initializes it to `0` as part of the declaration.
2. Declares `fcnA` to be "a function returning a pointer to a **char**" that has a parameter named `p1` that's "a pointer to **void**".
3. Declares `ppa` to be "a pointer to a parameterless function returning an **int**".
4. Declares `rppa` to be "a reference to a pointer to a parameterless function returning an **int**" and initializes it to `ppa` as part of the declaration.
5. C-style casts `aryP` to "a pointer to a parameterless function returning an **int**" and assigns the result to `ppa`.

Return:

void

- Place the comment **// 1.** or **// 2.** or **// 3.** or **// 4.** or **// 5.** as appropriate on the same line as each statement, after that statement, to indicate which of the above it represents. Place no other comments in the code.
- Ignore any "unreferenced" symbol/variable warnings from your compiler, but not from the Assignment Checker.
- DO NOT use any unnecessary parentheses.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C2A3E2_177752_U09845800** and with both source code files attached.

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

Hints:

A C++ reference variable is merely an alias for another variable. As such, every non-parameter reference variable must be initialized to the variable for which it is an alias in the same statement that declares it or a compiler error will be generated.

C2A3E3 *(6 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add a new one, naming it **C2A3E3_RecordOpinions.c**. Also add instructor-supplied source code file **C2A3E3_main-Driver.c**. Do not write a `main` function! `main` already exists in the instructor-supplied file and it will use the code you write.

File **C2A3E3_RecordOpinions.c** must contain a function named `RecordOpinions` that has no parameters and a `void` return. It must implement a survey of how shoppers like a product by prompting them to enter a decimal integer rating value within an allowed range of `BEST` to `WORST`. If any shopper enters a specified termination value the function must display a table of survey results and then return.

Define the following four macros to have any integer values within the stated constraints and use them appropriately in your code, but your code must also work with any other integer values within those constraints. Although defining additional macros is unnecessary, you may do so as long as it is not necessary to explicitly change their values if the values of any of the required macros are changed:

`BEST` – highest allowed rating value (any value >= 0)

`WORST` – lowest allowed rating value (always equal to `-BEST`)
The value of `WORST` must be derived from `BEST` and not specified as a literal number.

`CHOICES` – the number of integer values from `BEST` down to `WORST`
For example, if `BEST` is `3`, the value of `CHOICES` will be `7`. The value of `CHOICES` must be derived from `BEST` and not specified as a literal number.

`TERMINATE` – any value outside the range of allowed rating values.

`RecordOpinions` must:

1. Use a one-dimensional automatic type `int` array having exactly `CHOICES` elements. No other arrays are permitted. Each element represents an allowed rating and will contain a count of how many of that rating have been entered so far (similar to note 6.3).
2. Initially output a message that indicates the allowed rating range and the `TERMINATE` value. Then loop to individually prompt shoppers to enter their ratings.
   - If "end of file" is encountered or the rating represented by `TERMINATE` is entered, notify the shopper and go to step 3 below.
   - If an out-of-range rating other than that represented by `TERMINATE` is entered, notify and reprompt the shopper.
   - If an in-range rating is entered, increment the array element that represents that rating and prompt the next shopper. All array accesses must use the syntax ***p[i]***, where ***p*** represents a variable that points to the array's middle element and ***i*** represents the rating the shopper entered.
3. Return from `RecordOpinions` after first displaying a table like that below. The table must list all allowed rating values in the "Rating" column and the number of each that shoppers entered in the "Quantity" column. Entries must be in worst to best rating order and the least significant digits of all values must be aligned. A `BEST` value of 2 was used in this example:

```
Rating    Quantity
------    --------
    -2          25
    -1          50
     0         100
     1           3
     2           0
```

- If you believe you need more than one array or an array having other than `CHOICES` elements, you have not understood the requirements and should reread them as well as note 6.3.
- Do not test if the values of the required macros are valid.
- Do not use a loop to initialize your array to 0s or call a function to do it.

- Do not produce the absolute value of anything.

Manually re-run your program several times using various `BEST` and `TERMINATE` values and input ratings. Then test by specifying `BEST` and `TERMINATE` values of 5 and 999, respectively, and redirecting input from instructor-supplied data file **TestFile6.txt**. That file must be placed in the program's "working directory". To test with different macro values you will need to recompile after each change.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C2A3E3_177752_U09845800** and with both source code files attached.

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

Hints:

1. See note 6.3 for a similar exercise.

2. Whenever appropriate the best way to initialize an automatic array to all zeros is by using the standard syntax:

```
int ratings[CHOICES] = {0};
```

3. During runtime testing the assignment checker will change the values your code uses for the `BEST` and `TERMINATE` macros as well as the shopper ratings responses. Make sure all other code remains valid regardless of the values (within the stated constraints) the assignment checker uses.

4. Be sure your array contains the correct number of elements. For example, if `BEST` is `4`, the number of elements required (and the value of `CHOICES`) will be `9`.

5. Be sure your algorithm can support negative rating values and rating ranges that do not start or end with zero.

6. Look up the `scanf` function online or in any C textbook to learn about its return value.

C2A3E4 *(8 points – C Program)*

Exclude any existing source code files that may already be in your IDE project and add two new ones, naming them **C2A3E4_OpenFile.c** and **C2A3E4_ParseStringFields.c**. Also add instructor-supplied source code file **C2A3E4_main-Driver.c**. Do not write a `main` function! `main` already exists in the instructor-supplied file and it will use the code you write.

Often programs must read text files containing data fields of arbitrary length separated by arbitrary delimiters, such as commas, semicolons, etc. For example: **Hello, John Doe, Susan J. Smith**

File **C2A3E4_OpenFile.c** must contain a function named `OpenFile`.
`OpenFile` syntax:

```
FILE *OpenFile(const char *fileName);
```

Parameters:
`fileName` – a pointer to the name of the file to be opened
Synopsis:
Opens the file named in `fileName` in the read-only text mode. If the open fails an error message is output to `stderr` and the program is terminated with an error exit code. The error message must mention the name of the failing file.
Return:
a pointer to the open file if the open succeeds; otherwise, the function does not return.

File **C2A3E4_ParseStringFields.c** must contain a function named `ParseStringFields`.
`ParseStringFields` syntax:

```
void ParseStringFields(FILE *fp);
```

Parameters:
`fp` – a pointer to a file open in the read-only text mode
Synopsis:
Reads input from the text file in `fp` one line at a time and uses the `strtok` function to find each delimited field and display it on a separate output line. The characters "`AEIOUaeiou\t\n`" are treated as delimiters and any/all whitespace at the beginning of any field is skipped, with the `isspace` function being used to detect such whitespace. Certain character sequences will result in blank output lines or lines ending with one or more whitespaces.
Return:
void

- You may assume that lines will always contain less than 255 characters.
- Test your program using instructor-supplied data files **TestFile10.txt** and **TestFile12.txt**, which must be placed in the program's "working directory".

Example:
A properly written program would produce the display shown on the next page if the following two lines were read:

```
John Jones,2345 Bo Inlet St.,      Isle Ohau,           USA
      Mary        Lu,876-1/2 Back Road Dr.,BC, Mexico
```

Observe that the display contains no leading whitespace and there are 19 total lines, with two of them being blank. Do not show line numbers or anything else that is not actually in the file.

Sample Display

```
J
hn J
n
s,2345 B

nl
t St.,
sl

h
,
S
M
ry        L
,876-1/2 B
ck R
d Dr.,BC, M
x
c
```

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C2A3E4_177752_U09845800** and with all three source code files attached.

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

Hints:

Read one line at a time from the file and parse it with `strtok`. Be sure to skip leading whitespace. Whitespace is not just the *space* character itself but every character defined by the `isspace` function.