General Information

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.

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

Given the declaration

    float values[25];

and the following 10 expressions

    1.   x = <u>values</u>
    2.   **sizeof(** <u>values</u> **)**
    3.   **sizeof(** <u>values</u> + (**char**)65 **)**
    4.   **sizeof(** <u>values</u> [6] **)**
    5.   <u>values</u> [3]
    6.   22 [<u>values</u>]
    7.   & <u>values</u>
    8.   &(<u>values</u> + printf("Hi"))
    9.   & <u>values</u> [6]
    10. (**char ***) <u>values</u>

use the Right-Left rule to create a numbered sentence for each expression that describes the data type that identifier **values** is treated as. IMPORTANT: I am asking for the data type of **values** only (the <u>underlined part</u>), not the data type of the entire expression. Every sentence must begin with the words "**values** is". For example, in the first expression the correct answer is:

    1.   **values** is a pointer to a **float**.

Submitting your solution

Send your text file to the Assignment Checker with the subject line **C2A3E1_ID**, where **ID** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

Hint:
I consider this exercise to be a 2-point gift so don't make it harder than it actually is. There are only two possible sentences to choose from for all cases in the table (note 6.16). Although expression #8 is semantically illegal and will not compile, consider how array designator conversion would apply to it.

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**. <u>Do not write a **main** function!</u> **main** already exists in the instructor-supplied file and it will use the code you write.

The Reverse Right-Left rule is used to convert meaningful English sentences that describe standard C and C++ declarations and type casts into the syntactically correct declarations and type casts themselves. For example,

a sentence that reads	**test** is an **int**.
gets coded as	**int test;**
a sentence that reads	Type cast to a pointer to a **double**.
gets coded as	**(double *)**
and can be used with the syntax	**(double *)test**

The following 5 sentences describe 4 declarations and 1 type cast, all of which can be implemented as syntactically-legal code:

1. **vp** is a pointer to **void**.
2. **fcnA** is a function returning an **int** and having one **int** parameter named **val**.
3. **ppa** is a pointer to a pointer to an array of 9 **float**s.
4. **rppa** is a reference to a pointer to a pointer to an array of 9 **float**s.
5. Type cast to a pointer to a pointer to an array of 9 **float**s

File **C2A3E2_TestDeclarations.cpp** must contain a function named **TestDeclarations**.
TestDeclarations syntax:
```
void TestDeclarations();
```
Parameters:
    none
Synopsis:
    Contains <u>only</u> the following 5 lines of code:
1. the declaration of **vp** as described in sentence 1 above; initialize **vp** to **0** as part of the declaration;
2. the declaration of **fcnA** as described in sentence 2 above;
3. the declaration of **ppa** as described in sentence 3 above;
4. the declaration of **rppa** as described in sentence 4 above; initialize **rppa** to **ppa** as part of the declaration;
5. a statement that typecasts **vp** using the typecast described in sentence 5 above and assigns the result to **ppa**.

Return:
    void

- Place a comment by each of the 5 lines of code indicating the sentence number it represents. It's not necessary to repeat the actual sentences but if you do they must not contain magic numbers.
- Ignore any compiler-generated "unreferenced" symbol/variable warnings from your compiler (but not from the Assignment Checker).
- If declaration 2 causes a compiler error when placed inside the function, place it on the line just before the function instead.

Submitting your solution

Send both source code files to the Assignment Checker with the subject line **C2A3E2_ID**, where **ID** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your 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.

1

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

6
7   File **C2A3E3_RecordOpinions.c** must contain a function named **RecordOpinions**.
8   **RecordOpinions** syntax:
9   **void** RecordOpinions(**void**);
10 Parameters:
11   none
12 Synopsis:
13   Prompts users for their ratings of a product, counts the quantity of each rating, and displays a table
14   of the total quantity of each rating.
15 Return:
16   **void**

17
18
19 Additional **RecordOpinions** requirements/details:

20
21   1. Define macro **ENDPOINT** to be an arbitrary positive integer value and use it to determine the
22   range of legal user response values, which must be from **-ENDPOINT** (worst) to **+ENDPOINT** (best).

23
24   2. Define macro **TERMINATE** to be an integral value outside the range of legal response values.

25
26   3. Although you are free to define additional macros if you wish, it must not be necessary to
27   explicitly change any of their values if the values associated with either **ENDPOINT** or **TERMINATE**
28   are changed.

29
30   4. Keep count of the quantity of each response value in the appropriate element of a type **int**
31   1-dimensional array having exactly **(2 * ENDPOINT + 1)** elements. <u>Do not use more than 1</u>
32   <u>array or a multidimensional array for any purpose.</u>

33
34   5. The actual user response value must be used directly as the complete index value (negative or
35   positive) into the array without going out of bounds. For example, if a user inputs a value of -3
36   that is within the acceptable input range you must use that value as the complete index value.

37
38   6. If an out of range value other than the value represented by **TERMINATE** is entered, reject it and
39   re-prompt the user. If the value represented by **TERMINATE** is entered end the survey and
40   display a table like the one below that indicates the total number of responses for each possible
41   rating (an **ENDPOINT** value of 2 was used in this example). The least significant digits of all values
42   must be aligned for readability:

43   Rating   Responses
44   ------   ---------
45   -2   25
46   -1   50
47   0   100
48   1   3
49   2   0

50
51 Manually re-rerun your program several times using various **ENDPOINT**, **TERMINATE**, and response values.
52 Also test by redirecting input from instructor-supplied data file **TestFile6.txt**, which must be placed in the
53 program's "working directory". Use **ENDPOINT** and **TERMINATE** values of 5 and 999, respectively, when
54 using this file.
55

Submitting your solution

Send both source code files to the Assignment Checker with the subject line **C2A3E3_ID**, where **ID** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your Assignments" for additional exercise formatting, submission, and Assignment Checker requirements.

Hints:

Look up the **scanf** function online or in any C text book to learn about its return value.

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**. <u>Do not write a `main` function!</u> `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 file **TestFile10.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.

```
 1        Sample Display
 2        J
 3        hn J
 4        n
 5        s,2345 B
 6
 7        nl
 8        t St.,
 9        sl
10
11        h
12        ,
13        S
14        M
15        ry          L
16        ,876-1/2 B
17        ck R
18        d Dr.,BC, M
19        x
20        c
21
22
```

Submitting your solution

Send all three source code files to the Assignment Checker with the subject line **C2A3E4_ID**, where **ID** is your 9-character UCSD student ID.

See the course document titled "Preparing and Submitting Your 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.

Get a Consolidated Assignment 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 email to the assignment checker with the subject line **C2A3_ID**, where **ID** is your 9-character UCSD student ID. 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.