Assignment 3
C/C++ Programming II

C2A3 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.

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 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.

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

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

Given the declaration

```
double values[25];
```

and the following 10 expressions

1. `x = values`
2. `& values`
3. `(double **) values`
4. `sizeof( values [25] )`
5. `sizeof( & values + 0x20 )`
6. `values [-1]`
7. `75 [ values ]`
8. `&( putchar('A') + values )`
9. `sizeof( int ) * sizeof( values )`
10. `& values [6][7]`

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 only `values` (the underlined part), not the data type of the entire expression. Every sentence must begin with the words "`values` is" or "`values` decays to", as appropriate. For example, in the first expression the correct answer is:

1. `values` decays to a pointer to a **double**.

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:
Refer to notes 6.16 and A.1 to do this exercise. There are only two possible sentences to choose from for all 10 expressions. Also remember that 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 and can be extremely difficult if you don't.

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`.
`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 `cHelp` to be "a function returning a reference to a **char**" that has a parameter named `help` that's "a reference to a **float**".
2. Declares `abc` to be "a pointer to an array of 9 pointers to **short**s".
3. Declares `def` to be "a reference to a pointer to an array of 9 pointers to **short**s".
4. Declares `dpA` to be "an array of 9 pointers to **double**s".
5. Type casts (C-style) `dpA` to "a pointer to an array of 9 pointers to **short**s" and assigns the result to `abc`.

Return:
void

- Place the comment ***// 1.*** or ***// 2.*** or ***// 3.*** or ***// 4.*** or ***// 5.*** as appropriate on the same line as each 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 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.

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`.
`RecordOpinions` syntax:

```
void RecordOpinions(void);
```

Parameters:
none
Synopsis:
Prompts users for their ratings of a product, counts the quantity of each rating, and displays a table of the total quantity of each rating.
Return:
void

Additional `RecordOpinions` requirements/details:

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

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

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

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

5. Any in-range user response value must be used directly as the complete index value (negative or positive) into the array without going out of bounds. For example, if a user inputs an in-range value of -3 the value of the entire expression used within the square brackets to access the desired array element must also be -3.

6. Each user prompt must indicate the legal range of input values and the value needed to terminate the survey.

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

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

Manually re-rerun your program several times using various `ENDPOINT`, `TERMINATE`, and response values. Also test by redirecting input from instructor-supplied data file **TestFile6.txt**, which must be placed in the

program's "working directory". Use `ENDPOINT` and `TERMINATE` values of 5 and 999, respectively, when using this file.

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.

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 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.

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 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.