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.

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

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

3

4 Assume the two declarations and ten code snippets below:

5

```
6         double fish[57];
7         double *x;
8
9      1.  x = fish
10     2.  fish + 5 == (double *)0x2A65
11     3.  fputc((char)0[ fish ], stdout)
12     4.  sizeof((int *)'90' - (int *)& fish )
13     5.  0xABC + & fish [22]
14     6.  putchar((int)(void *)& fish )
15     7.  sizeof( fish ) + scanf("%lf", &x)
16     8.  scanf("abc%lf", & fish [18])
17     9.  while (cout.put((char)& fish [7]))
18    10.  putchar(* fish )
```

19

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

26

27   1. fish decays to a pointer to a double.

28

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

31

32 **Submitting your solution**

33 Send your text file to the assignment checker with the subject line **C2A3E1_*ID***, where **ID** is your
34 9-character UCSD student ID.

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

37

38

39 **Hints:**
40   1. Refer to notes 6.1, 6.16, 7.13, and A.1.
41   2. Unnecessary whitespace between tokens means nothing to the compiler.

1 **C2A3E2** *(4 points – C++ Program)*

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

6
7 This exercise:
8 • Requires absolutely no knowledge of what the data types being described actually mean;
9 • Is trivial if you understand notes 13.2-13.3B and can be extremely difficult if you don't.

10
11 The Reverse Right-Left rule (notes 13.3A & 13.3B) is used to convert meaningful English sentences that
12 describe standard C and C++ declarations and type casts into code. For example,

13
14 a sentence that reads   **test** is an **int**.
15 gets coded as   **int test;**

16
17 a sentence that reads   This is a C-style type cast to a pointer to a **double**.
18 gets coded as   **(double *)**
19 and can be used with the syntax   **(double *)test**

20
21
22 File **C2A3E2_TestDeclarations.cpp** must contain a function named **TestDeclarations**.
23 **TestDeclarations** syntax:
24   **void TestDeclarations();**
25 Parameters:
26   none
27 Synopsis:
28   Contains exactly five single statements, in order, that do the following. Only initialize if so stated:
29   1. Declares **afe** to be "a pointer to an array of 6 pointers to pointers to **long**s".
30   2. Declares **pv** to be "a pointer to a function returning a **float**" that has a parameter named **pa**
31   that's "a pointer to an array 6 **int**s" and initializes **pv** to **0** as part of the declaration.
32   3. Type casts (C-style) **pv** to "a pointer to an array of 6 pointers to pointers to **long**s" and assigns
33   the result to **afe**.
34   4. Declares **rF1** to be "a function returning a reference to an **int**" that has a parameter named
35   **precision** that's "a pointer to a **double**".
36   5. Declares **rF3** to be "a function returning a pointer to an **int**" that has a parameter named
37   **precision** that's "a reference to a **double**".
38 Return:
39   **void**

40
41 • Place the comment **// 1.** or **// 2.** or **// 3.** or **// 4.** or **// 5.** as appropriate on the same line as each
42   statement to indicate which of the above it represents. Place no other comments in the code.
43 • Ignore any "unreferenced" symbol/variable warnings from your compiler, but not from the assignment
44   checker.
45 • DO NOT use any unnecessary parentheses.

46
47 **Submitting your solution**

48 Send both source code files to the assignment checker with the subject line **C2A3E2_ID**, where **ID** is your
49 9-character UCSD student ID.

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

52
53

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

1 **C2A3E3** *(6 points – C Program)*

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 Additional **RecordOpinions** requirements/details:
19
20   1. Define macro **ENDPOINT** to be an arbitrary positive integer value and use it to determine the
21   range of legal user response values, which must be from **-ENDPOINT** (worst) to **+ENDPOINT** (best).
22
23   2. Define macro **TERMINATE** to be an integral value outside the range of legal response values.
24
25   3. Although you are free to define additional macros if you wish, it must not be necessary to
26   explicitly change any of their values if the values associated with either **ENDPOINT** or **TERMINATE**
27   are changed.
28
29   4. Keep count of the quantity of each response value in the appropriate element of a type **int**
30   1-dimensional array having exactly **(2 * ENDPOINT + 1)** elements. <u>Do not use more than 1</u>
31   <u>array or a multidimensional array for any purpose.</u>
32
33   5. Any in-range user response value must be used directly as the complete index value (negative
34   or positive) into the array without going out of bounds. For example, if a user inputs an in-range
35   value of -3 the value of the <u>entire expression</u> used within the square brackets to access the
36   desired array element must also be -3.
37
38   6. Each user prompt must indicate the legal range of input values and the value needed to
39   terminate the survey.
40
41   7. If an out of range value other than the value represented by **TERMINATE** is entered, reject it and
42   re-prompt the user. If the value represented by **TERMINATE** is entered end the survey and
43   display a table like the one below that indicates the total number of responses for each possible
44   rating (an **ENDPOINT** value of 2 was used in this example). The least significant digits of all values
45   must be aligned for readability:

```
46      Rating  Responses
47      ------  ---------
48        -2       25
49        -1       50
50         0      100
51         1        3
52         2        0
```

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

---

1 program's "working directory". Use **ENDPOINT** and **TERMINATE** values of 5 and 999, respectively, when
2 using this file.
3
4
5 ## Submitting your solution
6 Send both source code files to the assignment checker with the subject line **C2A3E3_ID**, where **ID** is your
7 9-character UCSD student ID.

8 *See the course document titled "How to Prepare and Submit Assignments" for additional exercise*
9 *formatting, submission, and assignment checker requirements.*
10

11
12 **Hints:**
13 Look up the **scanf** function online or in any C text book to learn about its return value.

1 **C2A3E4** *(8 points – C Program)*

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

6

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

9

10 File **C2A3E4_OpenFile.c** must contain a function named `OpenFile`.
11 `OpenFile` syntax:
12   FILE *OpenFile(**const char** *fileName);
13 Parameters:
14   `fileName` – a pointer to the name of the file to be opened
15 Synopsis:
16   Opens the file named in `fileName` in the read-only text mode. If the open fails an error message is
17   output to `stderr` and the program is terminated with an error exit code. The error message must
18   mention the name of the failing file.
19 Return:
20   a pointer to the open file if the open succeeds; otherwise, the function does not return.

21

22 File **C2A3E4_ParseStringFields.c** must contain a function named `ParseStringFields`.
23 `ParseStringFields` syntax:
24   **void** ParseStringFields(FILE *fp);
25 Parameters:
26   `fp` – a pointer to a file open in the read-only text mode
27 Synopsis:
28   Reads input from the text file in `fp` one line at a time and uses the `strtok` function to find each
29   delimited field and display it on a separate output line. The characters "**AEIOUaeiou\t\n**" are
30   treated as delimiters and any/all whitespace at the beginning of any field is skipped, with the
31   `isspace` function being used to detect such whitespace. Certain character sequences will result in
32   blank output lines or lines ending with one or more whitespaces.
33 Return:
34   **void**

35

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

39

40

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

44

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

47

48 Observe that the display contains no leading whitespace and there are 19 total lines, with two of them
49 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
```

23 **Submitting your solution**

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

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

28
29

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