Assignment 6
C/C++ Programming II

C2A6 General Information

Assignment 6 consists of FOUR (4) exercises:

C2A6E1   C2A6E2   C2A6E3   C2A6E4

All requirements are in this document.

1 **C2A6 General Information, continued**

2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 --- No General Information for This Assignment---
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44 **Get a Consolidated Assignment 6 Report (optional)**
45 If you would like to receive a consolidated report containing the results of the most recent version of
46 each exercise submitted for this assignment:
47 `Send an empty-body email to the assignment checker with the subject line **C2A6_164440_U09339367**
48 and no attachments.
49 Inspect the report carefully since it is what I will be grading. You may resubmit exercises and report
50 requests as many times as you wish before the assignment deadline.

1 **C2A6E1** *(2 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 **C2A6E1_GetPointers.c**. Also add instructor-supplied source code file **C2A6E1_main-Driver.c**.
4 <u>Do not write a `main` function!</u> `main` already exists in the instructor-supplied file and it will use the code
5 you write.
6
7 File **C2A6E1_GetPointers.c** must contain functions named `GetPrintfPointer` and `GetPutsPointer`.
8
9 `GetPrintfPointer` syntax:
10   **int (*GetPrintfPointer(void))(const char *format, ...);**
11 Parameters:
12   none
13 Synopsis:
14   Declares a pointer named `pPrintf` of appropriate type to point to the standard library `printf`
15   function and initializes it to point to that function. <u>It does all of this in one single statement.</u>
16 Return:
17   the initialized pointer named `pPrintf`, which points to the standard library `printf` function
18
19 `GetPutsPointer` syntax:
20   **int (*GetPutsPointer(void))(const char *str);**
21 Parameters:
22   none
23 Synopsis:
24   Declares a pointer named `pPuts` of appropriate type to point to the standard library `puts` function
25   and initializes it to point to that function. <u>It does all of this in one single statement.</u>
26 Return:
27   the initialized pointer named `pPuts`, which points to the standard library `puts` function
28
29
30 • Never explicitly write a prototype for a library function. Instead, use **#include** to include the
31   appropriate standard library header file. It will already contain the needed prototype.
32 • The `GetPrintfPointer` and `GetPutsPointer` functions must each contain two and only two
33   statements:
34   1. The statement that declares and initializes the pointer.
35   2. The statement that returns the pointer.
36
37
38 **Submitting your solution**

39 `Send an empty-body email to the assignment checker with the subject line **C2A6E1_164440_U09339367**
40 and with both source code files <u>attached</u>.

41 *See the course document titled "How to Prepare and Submit Assignments" for additional exercise*
42 *formatting, submission, and assignment checker requirements.*
43

44
45 **Hints:**
46 Look up the standard library `printf` and `puts` functions in your IDE's built-in help, any good C
47 programming textbook, or online, and examine their prototypes.

1 **C2A6E2** *(4 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 **C2A6E2_GetValues.cpp** and **C2A6E2_SortValues.cpp**. Also add instructor-supplied source
4 code file **C2A6E2_main-Driver.cpp**. <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 File **C2A6E2_GetValues.cpp** must contain a function named `GetValues`.
8 `GetValues` syntax:
9   `float *GetValues(float *first, size_t elements);`
10 Parameters:
11   `first` – a pointer to the first element of an array of **float**s
12   `elements` – the number of elements in that array
13 Synopsis:
14   Prompts the user to input `elements` whitespace-separated floating point values, which it then reads
15   with `cin` and stores into the successive elements of the array in `first` starting with element 0.
16 Return:
17   a pointer to the first element of the array

18

19

20 File **C2A6E2_SortValues.cpp** must contain a function named `SortValues`.
21 `SortValues` syntax:
22   `float *SortValues(float *first, size_t elements);`
23 Parameters:
24   `first` – a pointer to the first element of an array of **float**s
25   `elements` – the number of elements in that array
26 Synopsis:
27   Sorts the array in `first` <u>in descending order</u> using the "bubble sort" algorithm
28 Return:
29   a pointer to the first element of the sorted array

30

31

32 • `SortValues` must not call any functions.
33 • `GetValues` must not call any functions other than `operator>>` and `operator<<`.
34 • Use no global variables or global information about the array in either function.
35 • Do <u>not</u> use the syntax `pointer[offset]` or `*(pointer + offset)` to access array elements in
36   either function. Use `*pointer` or `*pointer++` instead.
37 • Use the following test values. Copying/pasting them from this document to the user prompt is an
38   easy way to avoid retyping them each time:
39   *1st prompt:*   1.2   3.4   5   6   7.7   8e4   22.6e-4   11.22   .00   0.4
40   *2nd prompt:*   -20   4   +16.8   -.0003   32.79   76   -6e6
41   *3rd prompt:*   1 2 3 4 5

42

43

44 **Submitting your solution**

45 `Send an empty-body email to the assignment checker with the subject line **C2A6E2_164440_U09339367**
46 and with all three source code files <u>attached</u>.

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

49

50

51 **Hints:**
52 Do not declare any arrays or create anything dynamically.

1 **C2A6E3** *(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 **C2A6E3_DisplayClassStatus.c**. Also add instructor-supplied source code file
4 **C2A6E3_main-Driver.c**. Do not write a `main` function! `main` already exists in the instructor-supplied file
5 and it will use the code you write.

6

7 A certain school keeps two sets of student names for every class taught. One set is for individuals who
8 have registered (registrants) and the other is for individuals (registered or not) who have attended the
9 first class meeting (attendees). Each set is kept in an appropriately-named ragged array as follows:

10

11   `const char *names[] = { "Al", "Ned Nasty", "Sweet L. Sally", etc. };`

12

13

14 File **C2A6E3_DisplayClassStatus.c** must contain functions named `Compare`, `SortStudents`, and
15 `DisplayClassStatus`.

16

17 `Compare` syntax:
18   `int Compare(const void *elemA, const void *elemB);`
19 Parameters:
20   `elemA` – a pointer to an element of a *names* array
21   `elemB` – a pointer to an element of a *names* array
22 Synopsis:
23   Compares the names represented by `elemA` and `elemB` using the standard library function `strcmp`.
24 Return:
25   <0 if the name represented by `elemA` is less than the name represented by `elemB`;
26   0 if the name represented by `elemA` is equal to the name represented by `elemB`;
27   >0 if the name represented by `elemA` is greater than the name represented by `elemB`.
28
29

30 `SortStudents` syntax:
31   `void SortStudents(const char *studentList[], size_t studentCount);`
32 Parameters:
33   `studentList` – A pointer to the first element of a *names* array
34   `studentCount` – The number of elements in the array
35 Synopsis:
36   Uses the standard library `qsort` function and the `Compare` function above to sort the array in
37   `studentList` into alphabetical order. No variables other than the two parameters may be
38   declared.
39 Return:
40   **void**

41

42

43 `DisplayClassStatus` syntax:
44   `void DisplayClassStatus( const char *registrants[], size_t registrantCount,`
45   `const char *attendees[],  size_t attendeeCount);`
46 Parameters:
47   `registrants` – pointer to the first element of a registrants *names* array
48   `registrantCount` – the number of elements in the registrants *names* array
49   `attendees` – pointer to the first element of an attendees *names* array
50   `attendeeCount` – the number of elements in the attendees *names* array
51 Synopsis:
52   1. Determines and displays which of the registrants did not attend the first meeting by repeatedly
53   calling the standard library `bsearch` function to search the attendees array for each name in the
54   registrants array.

Page 1 of 2 of C2A6E3

2. Determines and displays which of the attendees were not registered by repeatedly calling `bsearch` to search the registrants array for each name in the attendees array.
3. <u>Do not sort any arrays. Simply search them "as is".</u>
4. Results must be displayed in the following format, using the phrases "Not present:" and "Not registered:" as shown to differentiate the two groupings.

```
    Not present:
        Orphan Annie
        Toto The Dog
        Madonna
    Not registered:
        Little Mary
        Big John
        Tiny Tim
```

Return:
  void

The same comparison function must be used for both `qsort` and `bsearch`.

IMPORTANT: One purpose of this exercise is to illustrate the erroneous results that are usually obtained when `bsearch` is used on an unsorted array. My driver code will accomplish this by calling your `DisplayClassStatus` function both before and after calling your `SortStudents` function. For this reason your `DisplayClassStatus` function must do no sorting.

Submitting your solution

`Send an empty-body email to the assignment checker with the subject line **C2A6E3_164440_U09339367** and with both source code files <u>attached</u>.

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

Hints:
The first argument of **bsearch** must always be <u>the address of</u> (a pointer to) the object to be searched for, <u>not the value of</u> that object.

1   **C2A6E4** *(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 **C2A6E4_OpenFile.c** and **C2A6E4_List.c**. Also add instructor-supplied source code files
4   **C2A6E4_List-Driver.h** and **C2A6E4_main-Driver.c**. <u>Do not write a `main` function!</u> `main` already exists in
5   the instructor-supplied implementation file and it will use the code you write.

6

7   <u>Regarding data type `List`, which is used in this exercise...</u>
8   　　`List` is a typedef'd data type that is defined in instructor-supplied header file
9   　　　　**C2A6E4_List-Driver.h**
10   　　Any file that uses this data type must include this header file using `#include`.

11

12

13   File **C2A6E4_OpenFile.c** must contain a function named `OpenFile`.
14   `OpenFile` syntax:
15   　　`FILE *OpenFile(`**`const char`** `*fileName);`
16   Parameters:
17   　　`fileName` – a pointer to the name of the file to be opened
18   Synopsis:
19   　　Opens the file named in `fileName` in the read-only text mode. If the open fails an error message is
20   　　output to `stderr` and the program is terminated with an error exit code. The error message must
21   　　mention the name of the failing file.
22   Return:
23   　　a pointer to the open file if the open succeeds; otherwise, the function does not return.

24

25

26   File **C2A6E4_List.c** must contain functions named `CreateList`, `PrintList`, `FreeList`.

27

28   `CreateList` syntax:
29   　　`List *CreateList(FILE *fp);`
30   Parameter:
31   　　`fp` – a pointer to an open text file containing zero or more whitespace-separated words (strings)
32   Synopsis:
33   　　Creates a singly-linked list from strings it reads from the text file represented by parameter `fp`. Each
34   　　list node represents a unique case-dependent string and the number of times it occurred in the file.
35   　　This is the simplest algorithm and is recommended:
36   　　　　**1.** *Attempt to read a string from the file. If successful:*
37   　　　　　　**A.** *Search the list for that string.*
38   　　　　　　　　***i.*** *If found:*
39   　　　　　　　　　　**a.** *Increment the node's string count.*
40   　　　　　　　　***ii.*** *else:*
41   　　　　　　　　　　**a.** *Allocate a new node, and then*
42   　　　　　　　　　　**b.** *allocate memory for the string (including its \0), point the node's char*
43   　　　　　　　　　　　　*pointer to that allocation, and copy the string into it.*
44   　　　　　　　　　　**c.** *Set the node's string count to 1.*
45   　　　　　　　　　　**d.** <u>*Push*</u> *the node onto the list.*
46   　　　　　　**B.** *Repeat from step **1**.*
47   　　　　**2.** *Else, return the list's "head" pointer.*

48   Return:
49   　　the list's head pointer.

50

51   Examples – Number of nodes created if file contains:
52   　　`Fly fly!` (2 nodes)　　　`Fly Fly!` (2 nodes)　　　`Fly fly !` (3 nodes)　　　`Fly Fly !` (2 nodes)

---

1   `PrintList` syntax:
2   `List *PrintList(const List *head);`
3   Parameter:
4   `head` – the head pointer to the previously-described list
5   Synopsis:
6   Displays a non-sorted table of the data attributes from the list whose head pointer is passed to it,
7   starting at the head of the list. The display must be in the format illustrated below, in which the first
8   character in each string is aligned and the least significant digits of the occurrence counts are
9   aligned. There are no blank lines between entries. For example:

10   `the       107 ea`
11   `White      25 ea`
12   `White?      4 ea`
13   `if         16 ea`
14   etc...
15   Return:
16   `head`

17
18
19   `FreeList` syntax:
20   `void FreeList(List *head);`
21   Parameter:
22   `head` – the head pointer to the previously-described list
23   Synopsis:
24   Frees all dynamic allocations in the list.
25   Return:
26   **void**
27   Restrictions:
28   The `FreeList` function must call no functions or macros other than the standard library *free*
29   function, which it may call as needed.

30
31   General Exercise Requirements:
32   • Do not dynamically allocate space for a new node or string until you have first read a new string
33   from the text file, searched the existing list for it, and found it was not there.
34   • Do not use dynamic allocations other than those necessary for each node and its string.
35   • Do allocate space for a node and its string separately, **allocating for the node before the string**.
36   • Do allocate **exactly** the right amount of memory needed for each string, including its \0.
37   • Do not sort the list.
38   • Do not write code that requires a "pointer to a pointer" type or uses the syntax **(*p)->xyz**. Doing so
39   is unnecessary, cluttering, and inappropriate for this exercise.
40   • Do test the program on instructor-supplied data file **TestFile1.txt**, which must be placed in the
41   program's "working directory".
42   • Do not attempt to read the entire data file into your program at once.

43
44   **Submitting your solution**

45   `Send an empty-body email to the assignment checker with the subject line **C2A6E4_164440_U09339367**
46   and with all four source code files underlined.

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

49
50
51   **Hints:**
52   Include each string's null terminator when allocating memory and copying. When deleting a node
53   **always** free its string before freeing the node itself. Freeing the node first results in a memory leak.