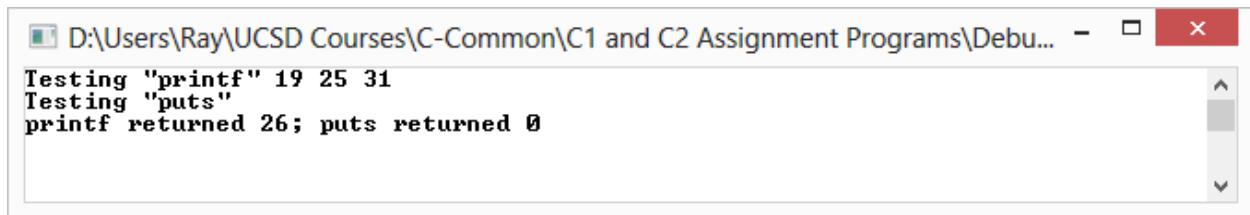


Exercise 1 (2 points – C Program)

```
1  /*
2
3  /*
4  * ...the usual title block Student/Course/Assignment/Compiler information goes here...
5  *
6  * This file contains functions:
7  *   GetPrintfPointer: Creates and returns a pointer to the printf function;
8  *   GetPutsPointer: Creates and returns a pointer to the puts function;
9  */
10
11 #include <stdio.h>
12
13 /*
14 * This function returns a pointer to the standard library
15 * printf function and does nothing else.
16 */
17 int (*GetPrintfPointer(void))(const char *format, ...)
18 {
19     int (*pPrintf)(const char *format, ...) = printf;
20     return pPrintf;
21 }
22
23 /*
24 * This function returns a pointer to the standard library
25 * puts function and does nothing else.
26 */
27 int (*GetPutsPointer(void))(const char *str)
28 {
29     int (*pPuts)(const char *str) = puts;
30     return pPuts;
31 }
```

C2A6E1 Screen Shot



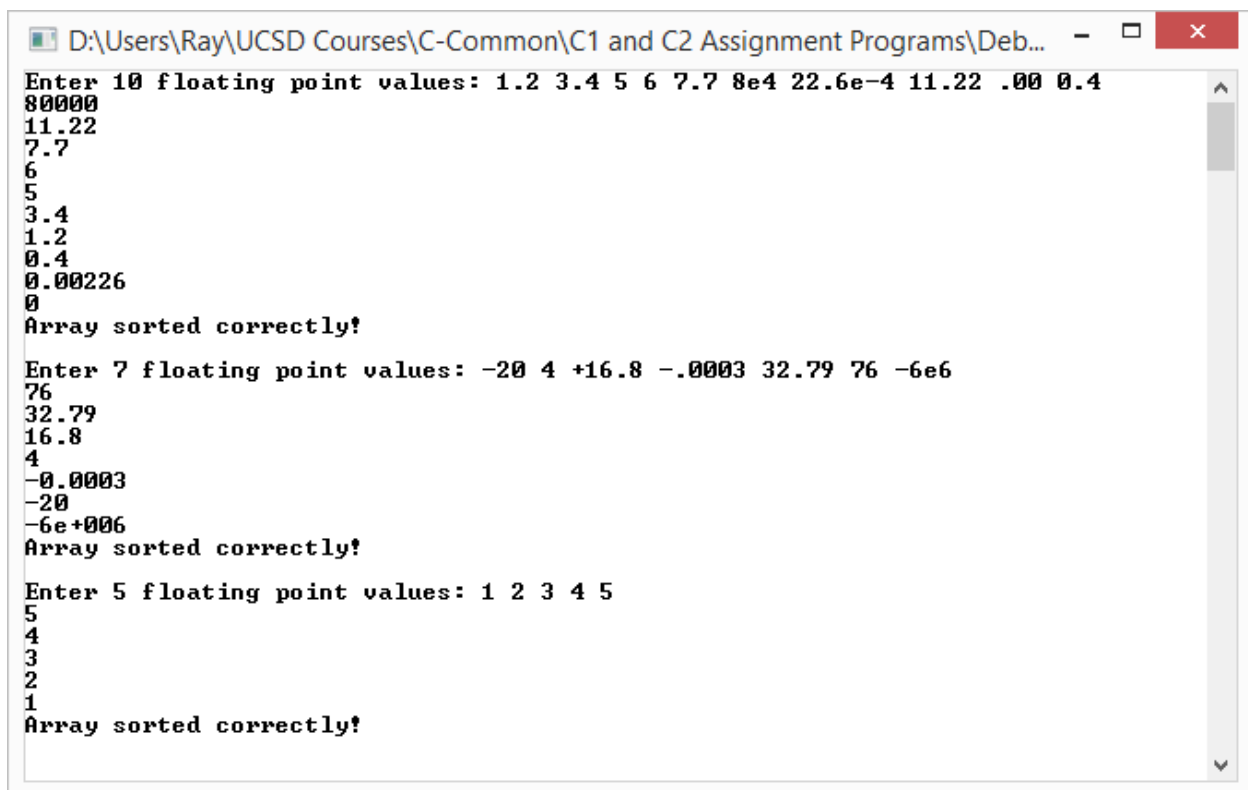
```
D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Debu...
Testing "printf" 19 25 31
Testing "puts"
printf returned 26; puts returned 0
```

Exercise 2 (4 points – C++ Program)

```
1 ***** FILE C2A6E2_GetValues.cpp *****
2
3
4 //
5 // ...the usual title block Student/Course/Assignment/Compiler information goes here...
6 //
7 // This file contains function GetValues, which prompts the user to input
8 // values used to populate the array represented by its first parameter.
9 //
10
11 #include <iostream>
12
13 //
14 // Prompt the user to enter the number of floating point values specified
15 // by <elements> and store them in the array in <first>. <first> is
16 // returned.
17 //
18 //
19 float *GetValues(float *first, size_t elements)
20 {
21     std::cout << "Enter " << int(elements) << " floating point values: ";
22     // Get one user entry per iteration and store in successive array elements.
23     for (float *end = first + elements; first < end; ++first)
24         std::cin >> *first;
25     return first - elements;
26 }
27
28 ***** FILE C2A6E2_SortValues.cpp *****
29
30 //
31 // ...the usual title block Student/Course/Assignment/Compiler information goes here...
32 //
33 // This file contains function SortValues, which sorts the elements of
34 // the array represented by its first parameter.
35 //
36
37 #include <cstddef>
38
39 //
40 // Sort the <elements> in the array in <first> in descending numerical
41 // order using the Bubble Sort algorithm. <first> is returned.
42 //
43 float *SortValues(float *first, size_t elements)
44 {
45     float *last = &first[elements - 1];
46     bool swapped;
47
48     // Loop until no swap occurs during a complete pass through the array.
49     do
50     {
51         swapped = false;
52         //
53         // One complete set of loop iterations represents one complete pass
54         // through the array.
55         //
56         for (float *ptr = first, *next = first + 1; ptr < last; ++ptr, ++next)
57         {
```

```
1     if (*ptr < *next)           // need to exchange
2     {
3         float temp = *ptr;      // do the...
4         *ptr = *next;           // ...3 step...
5         *next = temp;           // ...swap
6         swapped = true;         // indicate swap occurred
7     }
8 }
9 --last;
10 } while (swapped);
11
12 return first;
13 }
```

C2A6E2 Screen Shot



```
D:\Users\Ray\UCSD Courses\C-Common\C1 and C2 Assignment Programs\Deb...
Enter 10 floating point values: 1.2 3.4 5 6 7.7 8e4 22.6e-4 11.22 .00 0.4
80000
11.22
7.7
6
5
3.4
1.2
0.4
0.00226
0
Array sorted correctly!
Enter 7 floating point values: -20 4 +16.8 -.0003 32.79 76 -6e6
76
32.79
16.8
4
-0.0003
-20
-6e+006
Array sorted correctly!
Enter 5 floating point values: 1 2 3 4 5
5
4
3
2
1
Array sorted correctly!
```

Exercise 3 (6 points – C Program)

```
1  /*
2
3  /*
4  * ...the usual title block Student/Course/Assignment/Compiler information goes here...
5  *
6  * This file contains functions:
7  *   Compare: Compares the strings represented by its two parameters;
8  *   SortStudents: Sorts strings in the array represented its 1st parameter;
9  *   DisplayClassStatus: Displays specific strings from the arrays represented
10 *     by its two pointer parameters.
11 */
12
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <string.h>
16
17 /*
18 * Compare the C-strings in <elemA> and <elemB> using the standard library
19 * function strcmp and return the result directly from strcmp.
20 */
21 int Compare(const void *elemA, const void *elemB)
22 {
23     return(strcmp(*(char **)elemA, *(char **)elemB));
24 }
25
26 /*
27 * Call the standard library function qsort to sort <studentCount> C-strings
28 * in the array in <studentList>.
29 */
30 void SortStudents(const char *studentList[], size_t studentCount)
31 {
32     qsort((void *)studentList, studentCount, sizeof(*studentList), Compare);
33 }
34
35 /*
36 * Use the standard library bsearch function to search the array of C-strings
37 * in <attendees> for each C-string in the array in <registrants>. Then use
38 * the standard library bsearch function to search the array of C-strings in
39 * <registrants> for each C-string in the array in <attendees>. In each
40 * case display the C-strings NOT found.
41 */
42 void DisplayClassStatus(const char *registrants[], size_t registrantCount,
43                        const char *attendees[], size_t attendeeCount)
44 {
45     const char **cpp, **end;
46
47     /*
48      * For each name searched for bsearch returns a pointer to the appropriate
49      * element if the name is found and a null pointer if not. Each element
50      * is a pointer to a char and, therefore, the first bsearch argument must
51      * be a pointer to a pointer to a char (type casted to a void pointer).
52      */
53
54     /* See how many registrants are listed in the attendees array. */
55     printf("    Not present:\n");
56     for (cpp = registrants, end = cpp + registrantCount; cpp < end; ++cpp)
57         /* If the name was not found in the array, display it. */
```

```

1     if (!bsearch((void *)cpp, (void *)attendees, attendeeCount,
2                 sizeof(*attendees), Compare))
3         printf("          %s\n", *cpp);
4
5     /* See how many attendees are listed in the registrants array. */
6     printf("          Not registered:\n");
7     for (cpp = attendees, end = cpp + attendeeCount; cpp < end; ++cpp)
8         /* If the name was not found in the array, display it. */
9         if (!bsearch((void *)cpp, (void *)registrants, registrantCount,
10                     sizeof(*registrants), Compare))
11             printf("          %s\n", *cpp);
12 }

```

C2A6E3 Screen Shot

```

Class #0 before sorting:
  Not present:
  Not registered:
    The Flash
    The Green Lantern
    Xenon Man

Class #0 after sorting:
  Not present:
  Not registered:

Class #1 before sorting:
  Not present:
    Tough Guy
    Mae East
    Avenger
    Jo
    Mary
    Agitation King
    Zabo The Great
    Slim Jim
    Stinky The Clown
    Carl Crumb
    What's On Second
    Aces Wild
  Not registered:
    Al
    Tough Guy
    Mae East
    Bill
    Jo
    Mary
    Petty Patty
    Elmer Fudd
    Slim Jim
    Carl Crumb
    Ned Nasty
    Who's On First
    Aces Wild
    Night Flyer
    Carl Clean

Class #1 after sorting:
  Not present:
    Agitation King
    Avenger
    What's On Second
    Zabo The Great
  Not registered:
    Carl Clean
    Elmer Fudd
    Ned Nasty
    Petty Patty
    Who's On First

End Of Class Reports

```

Exercise 4 (8 points – C Program)

```
1  ***** FILE C2A6E4_OpenFile.c *****
2
3  /*
4   * ...the usual title block Student/Course/Assignment/Compiler information goes here...
5   *
6   * This file contains function OpenFile, which opens the file specified by its
7   * parameter in the read-only mode.
8   */
9
10 #include <stdio.h>
11 #include <stdlib.h>
12
13 /*
14  * Open the file named in <fileName> in the "read only" mode and return its
15  * FILE pointer if the open succeeds. If it fails display an error message
16  * and terminate the program with an error code.
17  */
18
19 FILE *OpenFile(const char *fileName)
20 {
21     FILE *fp;
22
23     /* Open the file named in <fileName> in the read-only mode. */
24     if ((fp = fopen(fileName, "r")) == NULL)
25     {
26         /* Print an error message and terminate with an error code. */
27         fprintf(stderr, "File \"%s\" didn't open.\n", fileName);
28         exit(EXIT_FAILURE);
29     }
30     return fp;
31 }
32
33 ***** FILE C2A6E4_List.c *****
34
35 /*
36  * ...the usual title block Student/Course/Assignment/Compiler information goes here...
37  *
38  * This file contains functions:
39  *   SafeMalloc: Dynamically allocates memory; built-in failure testing;
40  *   CreateList: Creates a singly-linked list from text file strings;
41  *   PrintList: Prints contents of singly-linked list of text file strings;
42  *   FreeList: Frees the singly-linked list of text file strings;
43  */
44
45 #include <stdio.h>
46 #include <stdlib.h>
47 #include <string.h>
48 #include "C2A6E4_List-Driver.h"
49
50 #define BUFSIZE 256                /* size of input buffer */
51 #define BUFFMT "%255s"            /* scanf field for buffer */
52
53 /*
54  * The syntax and functionality of SafeMalloc is identical to that of malloc
55  * with the following exception: If SafeMalloc fails to obtain the requested
56  * memory it prints an error message to stderr and terminates the program
57  * with an error code.
58  */
```

```
1 static void *SafeMalloc(size_t size)
2 {
3     void *vp;
4
5     /*
6      * Request <size> bytes of dynamically-allocated memory and terminate the
7      * program with an error message and code if the allocation fails.
8      */
9     if ((vp = malloc(size)) == NULL)
10    {
11        fputs("Out of memory\n", stderr);
12        exit(EXIT_FAILURE);
13    }
14    return(vp);
15 }
16
17 /*
18  * Create a singly-linked list where each node represents a unique
19  * whitespace-separated string from the text file in <fp>. A new node is
20  * created and pushed at the head of the list if a string not already in
21  * the list is read from the file. If a node for that string already
22  * exists its occurrence count is merely incremented.
23  */
24 List *CreateList(FILE *fp)
25 {
26     List *head;                /* pointer into list */
27     char buf[BUFSIZE];         /* for string input */
28
29     /* loop to get strings for insertion at top of list */
30     for (head = NULL; fscanf(fp, BUFFMT "%s", buf) != EOF;)
31     {
32         List *p;               /* pointer into list */
33         /* loop to find duplicates; order of logical && operands is critical */
34         for (p = head; p != NULL && strcmp(p->str, buf); p = p->next)
35             ;
36         if (p != NULL)         /* found same string */
37             ++p->count;        /* incr. string count */
38         else                   /* add new string at top */
39         {
40             size_t length = strlen(buf) + 1;    /* string len + the '\0' */;
41
42             p = (List *)SafeMalloc(sizeof(List)); /* allocate new node */
43             p->str = (char *)SafeMalloc(length); /* alloc string storage */
44             memcpy(p->str, buf, length);        /* copy string */
45             p->count = 1;                        /* init. string count */
46             /* Push the node onto the list (2 steps). */
47             p->next = head;                     /* next = head pointer */
48             head = p;                          /* point head to node */
49         }
50     }
51     return head;
52 }
53
54 /*
55  * Print the string and the number of occurrences of it represented by each
56  * node in the list in head-to-tail order.
57  */
```

```
1 List *PrintList(const List *head)
2 {
3     const List *p;                /* pointer into list */
4
5     for (p = head; p != NULL; p = p->next)    /* printing loop */
6         printf("%-15s%4d ea\n", p->str, p->count); /* string & count */
7
8     return (List *)head;
9 }
10
11
12 /*
13  * Free all dynamically-allocated memory in the list in head-to-tail order.
14  */
15 void FreeList(List *head)
16 {
17     while (head)                /* loop to free dynamic storage */
18     {
19         List *pTmp = head;      /* save pointer to current node */
20
21         head = head->next;        /* get pointer to next node */
22         free(pTmp->str);          /* free current node's str storage */
23         free(pTmp);              /* free current node's storage */
24     }
25 }
```

C2A6E4 Screen Shots are on the next page...

C2A6E4 Screen Shots

separated	1	ea
which	1	ea
from	1	ea
literals	1	ea
adjacent	1	ea
any	1	ea
concatenated	1	ea
automatically	1	ea
space.	2	ea
argument,	1	ea
two	1	ea
occurs	1	ea
comment	1	ea
if	1	ea
Thus,	1	ea
resulting	2	ea
single	2	ea
reduced	2	ea
tokens	2	ea
between	2	ea
white	4	ea
Any	1	ea
ignored.	1	ea
last	1	ea
following	1	ea
token	2	ea
first	1	ea
preceding	1	ea
space	3	ea
White	1	ea
definition.	1	ea
within	1	ea
stringizing	1	ea
combination	1	ea
of	4	ea
occurrence	1	ea
each	1	ea
replaces	1	ea
then	1	ea
literal	2	ea
literal.	2	ea
as	1	ea
treated	1	ea
and	3	ea
marks	1	ea

quotation	1	ea
enclosed	1	ea
invocation	1	ea
by	2	ea
passed	1	ea
argument	4	ea
actual	5	ea
definition,	1	ea
the	13	ea
in	5	ea
parameter	2	ea
formal	2	ea
a	6	ea
precedes	1	ea
it	3	ea
If	1	ea
arguments.	1	ea
take	1	ea
that	1	ea
macros	1	ea
with	2	ea
only	2	ea
used	1	ea
is	7	ea
It	1	ea
constants.	1	ea
string	6	ea
to	3	ea
expansion)	1	ea
<after	1	ea
parameters	1	ea
macro	4	ea
converts	1	ea
<#>	1	ea
operator	2	ea
"stringizing"	1	ea
or	1	ea
number-sign	1	ea
The	3	ea