CptS 121 — Program Design and Development

Date: 2020/07/12
Name: Devin Moore
ID#: 011 044 739
Class: CptS121 Andrew O'Fallon Section 9
TA: Muthuu Svs

Due Date: All parts are due by Wednesday, December 16, midnight PST!

Part I: Lab/Programming Final Exam
Worth 10% of your course grade

Exam Rules

- You must work **individually** on this exam.
- You may use your book, notes, and online resources if necessary.
- Using Blackboard, please submit your solution to the **"Written Final Exam"** folder. Yes, there's also a "Lab Final Exam" folder in Blackboard, but we are NOT going to use that!!!
- Your lab final project must contain one header file (a .h file), two C source files (which must be .c files), all resource files, and project workspace.
- Your project must build properly. The most points an assignment can receive if it does not build properly is 65 out of 100.
- Late exam solutions will **not** be accepted!

Programming Problem Overview:

Write a console application that computes and displays the total charges for each customer of a telecommunications company. Each customer's profile is represented by a record in a file called *customers.txt*. A record consists of the following:

- Customer name (last, first)
- Customer plan (A or B)
- Talk used in minutes
- Data used in MB

You must read in all of the records from the file. You may assume the file does not consist of more than 100 records. You will need to write algorithms that sort customer plans based on their names (in reverse dictionary ordering 'z' – 'a') and that traverses through the records and determines charges for each customer based on the following:

Plan A Customers:

- Up to 1000 minutes of talk for $35; $0.50 per minute thereafter (overage)
- Up to 10 GB of data for $25; $0.25 per MB thereafter (overage)
- Monthly charge: $60, excluding extra charges for overages

Plan B Customers:

- Up to 2000 minutes of talk for $55; $0.40 per minute thereafter (overage)
- Up to 20 GB of data for $30; $0.15 per MB thereafter (overage)
- Monthly charge: $85, excluding extra charges for overages

You will need to write the following information to another file called *charges.txt*:

- Total charges (sum of all customers' charges)
- Average charges per customer
- Max charges
- Min charges

Design Requirements:

For this problem you must define the following `struct` type:

```
typedef struct profile
{
	char name[100];		// customer's name - last, first
	char plan;		// plan 'A' or 'B'
	int talk_minutes;	// number of minutes used for talking
	int data_MB;		// amount of data used in MB
	double charges;		// total charges for this customer – you will compute!
```

```
} Profile;
```

You must also define an array of `Profile`s that contains a maximum of 100 customer profiles. For example,

```
Profile customers[100]; // the input file may not exceed 100 customer profiles
```

You must also complete the following:

- Open *customers.txt* for mode read
- Read all records in *customers.txt* and store them into the array of `Profile`s
- *Sort* the array based on customers names 'z' – 'a'
- Compute the charges for each customer based on the plan for the customer, and the talk and data usages – store the charges back into the `Profile` in the array; recall 1 GB = 1000 MB
- Compute the total or sum of charges for all customers
- Compute the average charges per customer
- Compute the maximum charges
- Compute the minimum charges
- Open *charges.txt* for mode write
- Write the total, average, maximum, and minimum charges to *charges.txt*
- Close *customers.txt* and *charges.txt*

Sample customers.txt File:

```
Trump, Donald
B
1770
19000
Sanders, Bernie
A
995
900
Musk, Elon
A
1100
10005
Obama, Barack
B
2050
21000
```

Sample charges.txt File:

```
Total: $511.25
Average: $127.81
Max: $255.00
Min: $60.00
```

Notes and Hints

- You should perform error checking in your program.
- You are not required to document your code as you would with a programming assignment! However, a sound top-down design, coupled with good documentation, will increase the chances that you receive partial credit in the event that your solution is incorrect.

Grading

The lab portion of the exam is worth 100 points. To grade your solution, your lab TA will run your program using the four records listed above. You will receive points as follows:

- **5 pts:** Program correctly defines the `Profile` struct
- **5 pts:** Program correctly **o**pens *customers.txt* for mode read
- **10 pts:** Program correctly **r**eads all records in *customers.txt* and stores them into the array of `Profile`s
- **15 pts:** Program sorts the array of `Profile`s based on customer name 'z' – 'a'
- **15 pts:** Program correctly **c**omputes the charges for each customer based on the plan for the customer, and the talk and data usages – store the charges back into the `Profile` in the array
- **10 pts:** Program correctly **c**omputes the total or sum of charges for all customers
- **5 pts:** Program correctly **c**omputes the average charges per customer
- **10 pts:** Program correctly **c**omputes the maximum charges for a customer
- **10 pts:** Program correctly **c**omputes the minimum charges for a customer
- **5 pts:** Program correctly **o**pens *charges.txt* for mode write
- **5 pts:** Program correctly **w**rites the total, average, maximum, and minimum charges to *charges.txt*
- **5 pts:** Program correctly **c**loses *customers.txt* and *charges.txt*

Note: The most points you can receive for a solution that does not properly build is 65 out of 100.

Part II: Written Final Exam
Worth 10% of your course grade

TOTAL POINTS POSSIBLE: 150

NOTE: Complete the answers to the questions, and add your file to the project that you created for Part I. Please add your document to the "Resource Files" folder in Visual Studio.

Part I: Conceptual Questions (90 pts) – Multiple Choice and True/False – 2 pts each

1. True or False: Algorithms should be specified in pseudocode *before* they are implemented in C.

2. True or False: In C, *false* is defined as any result that is zero.

3. True or False: An *advantage* to defining functions in C is that they may be *reused* in other projects.

4. True or False: Suppose a function main calls a function f. As a parameter to f, main passes a pointer p to a struct. Further suppose that f's body modifies p by pointing it to a different struct. When f returns, the pointer passed by main will also be pointing to a different struct.

5. True or False: An identifier/variable name, in C, may start with a digit.

6. True or False: Preprocessor directives are handled *during compilation*.

7. True or False: The control expression of a switch-case statement in C can be of type *enum*.

8.
9. True or False: Assume that a character pointer cptr points to a character array carray of size 10. The following expressions are equivalent: *carray[2] and *(cptr + 2).

10. True or False: In ***C***, you can assign an array of character pointers to point to specific addresses, i.e.

    ```
    char *array[10], var;
    array[5] = &var;
    ```

11. True or False: We use the debugger to find *logic* errors.
12. True or False: In ***C***, arrays and pointers are essentially interchangeable—you can use *always* use pointer notation for arrays and *always* use array notation for pointers.
13. True or False: In ***C***, when an array is passed to a function, only the address of the *zeroth* element is passed to the function.
14. True or False: An actual argument is the argument that we pass into a function when it is called, i.e. my_function ('a'), where 'a' is the actual argument.
15. True or False: The following is an example of a *sentinel-controlled* loop:

    ```
    while (my_array[i] != '\0') …
    ```

16. True or False: The condition in the following `if` statement is a *syntax* error:

    ```
    int number = -50;
    ```

...

if (number)

...

17. True or False: The standard ***C*** function `strlen()` will return 4 when invoked upon the variable name, declared below:

    char name[] = {'c', 'p', 't', 's', '\0', '1', '2', '1', '\0'};

18. True or False: A *structure* is a collection of related *variables* under one name.
19. True or False: In ***C***, output parameters allow for a function to return more than one value *indirectly*.
20. True or False: A variable of a specific struct type can be copied to another variable of the same struct type through `strcpy()`.
21. True or False: An *array* of *structs* may be used to replace *parallel* arrays, if the appropriate attributes are defined in the struct.
22. True or False: Function `malloc()` is used to dynamically allocate memory at program *runtime*.
23. True or False: Shifting a number *right* by *two* bit positions, using >>, is the same as multiplying the number by *four*.
24. True or False: Character arrays that are passed to functions in the C character library (ctype.h) need to always have a null character as the last terminating element in order to achieve the correct result.
25. True or False: An *iterative* construct (i.e. a while, for, do-while loop) may *always* be used instead of *recursion*.
26. True or False: The bitwise AND (&) operator may be used to clear individual bits.
27. True or False: More bytes of memory are required for a *double precision* variable than a *character* variable.
28. True or False: Given the following snipet of code:

```
char *my_ptr = "CptS 121";
printf ("%c", *(my_ptr++));
```

    The output of the printf is 'p'.

29. True or False: The functions in <string.h> use the *length* of the string to determine the end of the string.
30. True or False: The strcmp function in <string.h> will return 0 when comparing “CptS121” to “cpts121”.

31. True or False: Provided an integer is 4 bytes, the array `number` begins at address 1111, and the following:

```
int *num_ptr = NULL, number[3] = {42, 1111, 56};
num_ptr = number;
```

The result of num_ptr += 2 is *1113*.

32. True or False: All recursive solutions must have a *base* case.

33. What will be the value of i after the C statements at the right have been executed?

a. 11
b. 33
c. 34
d. 10
e. 46

```
int i = 10;
int j = 33;
    while ((3 * ++i) < j)
        i = i + j++;
```

34. What is displayed by the C statements that follow if the input value is 3?

```
scanf("%d", &ctl);
switch (ctl) {
case 0:
case 1:
    printf("red ");
case 2:
    printf("blue ");
    break;
case 3:
    printf("orange ");
    break;
case 4:
    printf("yellow");
}
printf("\n");
```

a. red
b. blue
c. red blue
d. yellow
e. orange yellow
f. none of the above

Questions **34-37** refer to the following program *segment*. Assume that all variables are of type int.

```
.....
        z = 0;
        g = 0;
        s = 0;
        i = 50;
        while (i > 0) {
                scanf("%d", &t);
                s = s + t;
                if (t >= 0)
                        g = g + 1;
                else
                        z = z + 1;
                --i;
        }
```

35. How many times is the loop body of the while statement executed?
    a. once
    b. forever
    c. 49 times
    d. 50 times
    e. until a number 50 or larger is entered

36. The value stored in variable s at the end of the execution of the loop could best be described as:
    a. the average of the numbers scanned
    b. the sum of the numbers scanned
    c. the largest of the numbers scanned
    d. how many numbers were scanned
    e. the sentinel value

37. The value stored in variable z at the end of the execution of the loop could best be described as:
    a. how many positive items were scanned
    b. the sum of all positive items scanned
    c. how many negative items were scanned
    d. the sum of all negative items scanned
    e. the sentinel value

38. The loop can best be categorized as a:
    a. counter-controlled loop
    b. sentinel-controlled loop
    c. loop that computes a product
    d. general conditional loop
    e. none of the above

39. What is the output of the following program?

```
#include <stdio.h>
        void do_something(int *thisp, int that)
        {
                int the_other;

                the_other = 5;
                that = 2 + the_other;
                *thisp = the_other * that;
        }

        int main(void)
        {
                int first, second;

                first  = 3;
                second = 2;
                do_something(&second, first);
                printf("%-d  %3d\n", first, second);

                return (0);
        }
```

a. 35 7
b. 3 35
c. 35 3
d. 1 2
e. 0 7
f. none of the above

Consider the following declarations as you determine whether each of the assignment statements in questions 39-45 is valid or invalid (it either won't compile or could crash):

```
#define SIZE1 65
#define SIZE2 34

char a_list[SIZE1] = "CptS 121", b_list[SIZE1], a_char = 'v';
int  nums[SIZE2], vals[SIZE2], i = 1, *ptr = vals;
```

40. Valid or Invalid: *nums = i;
41. Valid or Invalid: *ptr = vals + 1; // this is valid but I think it just increments the pointer value
42. Valid or Invalid: a_list[SIZE1] = 'd';
43. Valid or Invalid: b_list[30] = 0.37 * vals[1];
44. Valid or Invalid: a_list = b_list;
45. Valid or Invalid: nums[i + 7] = (int) a_char;
46. Valid or Invalid:

```
for  (i = 1;  i <= SIZE1;  ++i)
        nums[SIZE2 - i] = i;
```

Part II: Fill-In-The-Blank, Short Answer, and Some Code (60 pts)

47. **(3 pts)** Fill in the blanks in the following ***C*** statement such that it allocates memory for an array of 50 character values:

    char *name = (___char *___) malloc(___50 * sizeof(char)_____);

48. **(3 pts)** Write a declaration for an array a of 10 strings, each of which has at most 80 characters (including the null character):

    _____char stringArray[10][80];_____

49. **(6 pts)** Circle or underline all syntax, logic, and runtime errors found in the following C fragment. Be sure to circle omitted punctuation.

```
char *name_ptr == NULL, name[50] = “Barack Obama”, // get rid of extra ‘=’
      *name2 = “Joe Biden”;

name_ptr = name[0]; // get rid of “[0]”

do
{
        printf (“Enter a new name: ”)__ // need a ‘;’ here
        scanf (“%c”, name_ptr); // use %s or gets
__while (name == name2); // need a close curly brace ‘}’ at the start of while
```

50. **(2 pts)** What are two *advantages* of using recursion in a C program?

    1. Allows you to condense potentially complicated nested conditionals or case statements into more readable code.
    2. ____Can make repetitive tasks potentially more efficient to run_____

51. **(8 pts)** Complete the following statements based on this declaration. Assume that the operations are cumulative:

    char taste[11] = "bitter", touch[5] = "soft";

    The value of strlen(taste) is ____6____. The contents of touch[4] is ____‘\0’___.

The value of strcmp(“soft”, taste) is negative/positive/zero (**circle one**). The contents of touch after the call strncpy(touch, taste, 3) is ___“bitt”_____. The contents of taste[2] after the call strncat(taste, &touch[3], 1) is_____‘t’_____.

52. (**4 pts**) What is the output of the following code segment?

```
for  (k = 4;  k > 0;  k = k - 1)
{
        for  (i = 1; i <= 5 - k;  i = i + 1)
           printf(".");
        for  (j = 1;  j <= 2 * k - 1;  j = j + 1)
           printf("A");
        printf("\n");
}
```

```
.AAAAAAA
..AAAAA
...AAA
....A
```

53. **(5 pts)** Define a struct called Vehicle (be sure to rename it) that contains a `make` field of length 20, a `model` field of length 25, and a `year`:

```
typedef struct
{
        char make[20];
        char model[25];
        int year;
} Automobile;
```

53. **(5 pts)** Rewrite the if statement below using only the relational operator < in all conditions. Assume that the value of score is between 0 and 100 inclusive.

Original if statement:	Your rewrite using only <:

if (score >= 90)
  printf("A\n");
else if (score >= 80)
  printf("B\n");
else
  printf("C\n");
	if (score < 80)
	{ printf("C\n"); }
	else if (score < 90)
	{ printf("B\n"); }
	else
	{ Printf("A\n"); }

54. (**4 pts)** Rewrite the following code segment as an equivalent segment that uses a *do-while* statement.

Original while loop:	Your rewrite as a do-while loop:

int sum = 0, next = 1,
  bound = 10;
while (next < bound) {
  sum += (++next);
 }
	int sum = 0, next = 1,
	  bound = 10;
	do {
	  sum += (++next);
	} while (next < bound);

55. **(2 pts)** Assuming the following declaration:

char my_array[9] = {'C', 'p', 't', 'S', ' ', '1', '2', '1'};

What is the *value* of *(my_array + 4)? ' '

56.**(3 pts)** What will be the values of k[1], k[2], and k[3] *after* execution of the code segment below assuming the input data shown?

Code segment:	

int k[6] = {0, 0, 0, 0, 0, 0};
int i, n;
for (i = 2; i < 6; ++i) {
  scanf("%d", &n);
  k[i] = n;
 }
Input data: 42 6 -56 7
	Value of k[1]: 0
	Value of k[2]: 42
	Value of k[3]: 6

57. **(3 pts)** What is dynamic memory?

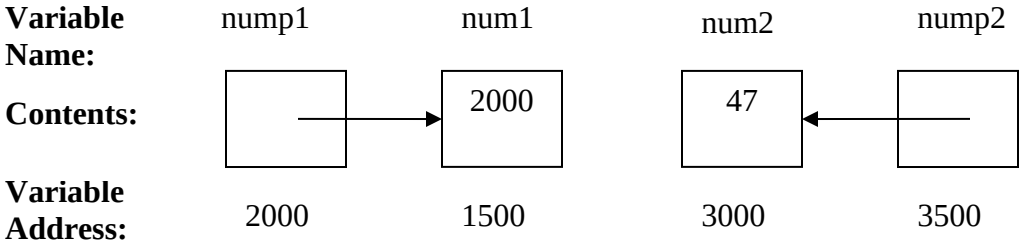
Memory that is allocated during runtime by the programmer (malloc())0

Memory created this way is allocated to the heap, and is only freed when the programmer frees it in the code free()

58. **(2 pts)** Provide the base case(s) for the recursive solution that solves the Factorial (n) problem. Recall n! = n * (n-1) * (n-2) * … * 1.

n = 0; when n = 0 return 1; // 0! = 1

59. **(10 pts)** Given the following information, fill in the blanks in the statements or outputs. For the output, just provide the number, not the entire printf string.



Statement	Output
printf ("The contents of nump2 is: %d\n", nump2);	3000
printf ("The indirect value of nump1 is: %d\n", *nump1);	2000
printf ("The address of nump2 is: %d\n", &nump2);	3500
printf ("The direct value of num2 is: %d\n", num2);	47
printf ("The address of num1 is: %d\n", &num1);	1500