

## COP 3223 Sec 3: Spring'24 C Programming Test 2 Practice (50 points)

1. (10 points) Write down what the printed output of this program is. Assume input is aaBcc\*cC7

```
#include <stdio.h>
#include <ctype.h>
int main() {
    int index, freq[26], c, stars, maxfreq;
    for (index=0; index<26; index++)
        freq[index] = 0;
    while ( (c = getchar()) != '7') {
        if (isalpha(c))
            freq[tolower(c)-'a']++;
    }
    maxfreq = freq[25];
    for (index = 24; index >= 0; index--) {
        if (freq[index] > maxfreq)
            maxfreq = freq[index];
    }
    printf("a b c d e f\n");
    for (index=0; index<5; index++) {
        for (stars=0; stars< (maxfreq - freq[index]); stars++)
            printf(" ");
        for (stars=0; stars< (freq[index]); stars++)
            printf("*");
        printf("%c \n", ('A' + index) );
        printf(" \n");
    }
    return 0;
}
```

THIS OUTPUT WILL BE GRADED \*\*\* MARK YOUR BLANK SPACES \*\*\*

		-----
Out Line 1		
Out Line 2		
Out Line 3		
Out Line 4		
Out Line 5		
Out Line 6		

2. (9 points) Write down what the printed output of this program is:

```
1.  #include <stdio.h>
2.
3.  int main()
4.  {
5.      int a[6] = {0, 3, 5, 7, 4, -1};
6.      int b[6] = {1, 10, 20, 40, 100, 2000};
7.      int i,x,temp;
8.
9.      for(i = 0; i < 6; i++)
10.         printf("%d ", a[i]);
11.     printf("\n");
12.
13.     for(i = 1; i < 6; i++)
14.     {
15.         a[i] += a[i-1];
16.     }
17.
18.     for(i = 0; i < 6; i++)
19.         printf("%d ", a[i]);
20.     printf("\n");
21.
22.     printf("Result is: %f\n", a[5] / 6.);
23.
24.     x = 5;
25.
26.     for(i = 0; i < x; i++)
27.     {
28.         temp = b[x];
29.         b[x] = b[i];
30.         b[i] = temp;
31.         x--;
32.     }
33.
34.     for(i = 0; i < 6; i++)
35.         printf("%d ", b[i]);
36.     printf("\n");
37.
38.     return 0;
39. }
```

3. (8 points) Write a C program that will read in integers until it encounters the integer -99. Note that -99 could be the first integer. For each other (earlier) integer, if present, you need to compute its squared value, and then print out the integer followed by its squared value, one pair per line with two blanks between the pair. Assume the other integers (besides the -99) are positive, and do not print a line for -99.
4. (12 points) Write a complete C program that will first read in an arbitrary number of integers (which it should not do anything with) until it encounters the integer -99. Assume it is guaranteed to encounter

the integer -99. Following this, it will definitely encounter (and hence, it will read in) two integers, a width  $W$ , and a height  $H$  (you do NOT need to prompt for these). The program is expected to then draw a parallelogram, but draw it upside down. So, given width  $W$  and height  $H$ , the program should print  $W$  stars on each of  $H$  lines, in such a manner that the first line should have  $H - 1$  blanks before the stars, and on following lines, the number of blanks should decrease by one. Assume all input is correct. Remember: it may help you to assume that you do NOT need to prompt for anything, and you should make the program ignore the integers (if any) before the integer -99.

Sample input:

```
2 3 7 91 103 4 -99 3 5
```

Sample output:

```
***
***
***
***
***
```

5. (10 points) Write a program to prompt for and read in an odd positive integer and produce a diamond shape on the page made up of asterisks; the number is the height, and also the breadth.
6. (11 points) Write a program that will read in and decide if a given integer is prime (a prime number is divisible by only 1 and itself).
7. (11 points) Write a C program that asks the user to enter one float at a time. As long as the user continues to enter descending floats (the newest float is less than the previous one entered) the program should report the product of the current and previous number (except for the first float entered). If the user enters a number that is not smaller than their last response, the program should tell them so and exit. You can assume that for each run of the program, the user will enter at least two floats, and will not try to trick the program; so, you do not need to check for errors in what the user types in.
8. (11 points) (This is from the previous textbook) Write a program that prompts the user to enter an integer  $n$ , then prints all even squares between 1 and  $n$ . For example, if the user enters 100, the program should print: 4 16 36 64 100
9. (14 points) (This is from a previous textbook) Write a program that prints a one-month calendar. The user specifies the number of days in the month and the day of the week on which the month begins:

```
Enter number of days in month: 31
```

```
Enter starting day of the week (1=Sun, 7=Sat): 3
```

```
The month calendar is below:
```

```
    1  2  3  4  5
  6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

*Hint:* The most important part of the solution is a *for* statement that uses a variable  $i$  to count from 1 to  $n$ , where  $n$  is the number of days in the month, printing each value of  $i$ . Inside the loop, an *if* statement tests whether  $i$  is the last day in a week; if so, it prints a new line character.