COMP1511 18s2

Sample Final Exam Questions

[Last updated 12:30pm Friday 19/Oct]

The exam environment for your final exam will be similar to your two lab practical exams. Your exam supervisor will inform you how to login and start using your exam environment. There will be 7+ questions in the exam, and you need to answer all the questions. The required files will be available in the directories named "q1", "q2", "q3", "q4", "q5", etc. in your (exam) home directory. You need to submit the required files as specified in each exercise.

Important:

- This sample exam contains *sample* questions only. The questions in your final exam may be derived from the topics not covered in this sample exam.
- Hurdle Requirements:
  - You need to answer any **one** of the two **array** questions (marked below).
  - You need to answer any **one** of the two **linked list** questions (marked below).
- In order to receive marks for an exercise, your program must compile successfully without any errors and pass "new" auto-tests. You will be awarded marks only if your solution is correct and properly solves the problem (no hard-coded solutions!). If your algorithm/solution is incorrect, even if you pass auto-tests, you will NOT be awarded marks.

More: "Practice Questions for the Final Exam"

In addition to the following questions, you should also try **Practice Questions for the Final Exam** .

Part-1

Part-1: Question 1 (1 mark)

The file **question.c** contains this C Code:

```
#include <stdio.h>

int main(void) {
    int x = 11;
    int y = 3;
    printf("%d\n", x / y);
    return 0;
}
```

question.c is compiled with dcc on a CSE machine like this:

```
$ dcc question.c -o question
$
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./question
```

What does this program print?

Enter as your answer the output the program produces. Do not enter any extra characters. Do not enter any explanation.

Enter exactly the output the program produces.

If the program prints an error message just write **ERROR**. Do not enter the exact error message.

Part-1: Question 2 (1 mark)

The file **q2.c** contains this C Code:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("%d\n", argc);
    return 0;
}
```

q2.c is compiled with dcc on a CSE machine like this:

```
$ dcc q2.c -o q2
$
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./q2 hello world
```

What does this program print?

Enter as your answer the output the program produces. Do not enter any extra characters. Do not enter any explanation.

Enter exactly the output the program produces.

If the program prints an error message just write **ERROR**. Do not enter the exact error message.

Part-1: Question 3 (1 mark)

The file **q3.c** contains this C Code:

```
#include <stdio.h>
#include <string.h>

int main(void) {
    char *s = "hello";
    char i = strlen(s);
    printf("%c%d\n", s[0], i);
    return 0;
}
```

q3.c is compiled with dcc on a CSE machine like this:

```
$ dcc q3.c -o q3
$
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./q3
```

What does this program print?

Enter as your answer the output the program produces. Do not enter any extra characters. Do not enter any explanation.

Enter exactly the output the program produces.

If the program prints an error message just write **ERROR**. Do not enter the exact error message.

Part-1: Question 4 (1 mark)

The file **question.c** contains this C Code:

```
#include <stdio.h>
```

```
int main(void) {
    int x = 11;
    int y = 3;
    printf("%d\n", x % y);
    return 0;
}
```

question.c is compiled with dcc on a CSE machine like this:

```
$ dcc question.c -o question
$
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./question
```

What does this program print?

Enter as your answer the output the program produces. Do not enter any extra characters. Do not enter any explanation.

Enter exactly the output the program produces.

If the program prints an error message just write **ERROR**. Do not enter the exact error message.

Part-1: Question 5 (1 mark)

The file **q2.c** contains this C Code:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("%s\n", argv[1]);
    return 0;
}
```

q2.c is compiled with dcc on a CSE machine like this:

```
$ dcc q2.c -o q2
$
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./q2 hello world
```

What does this program print?

Enter as your answer the output the program produces. Do not enter any extra characters. Do not enter any explanation.

Enter exactly the output the program produces.

If the program prints an error message just write **ERROR**. Do not enter the exact error message.

Part-1: Question 6 (1 mark)

The file **q3.c** contains this C Code:

```
#include <stdio.h>
#include <string.h>

int main(void) {
    char *s = "hello";
    printf("%d\n", strcmp(s, s));
    return 0;
}
```

q3.c is compiled with dcc on a CSE machine like this:

```
$ dcc q3.c -o q3
$
```

It compiles successfully. No errors or warnings are produced by dcc. The program is run like this:

```
$ ./q3
```

What does this program print?

Enter as your answer the output the program produces. Do not enter any extra characters. Do not enter any explanation.

Enter exactly the output the program produces.

If the program prints an error message just write **ERROR**. Do not enter the exact error message.

Part-1: Questions 7 to 15

Approx. 15 questions in Part-1.
Other examples include:

- Q2, Q4, Q5 from **Week-03 Tutoria**l Exercises
- Q2 and Q9 from **Week-05 Tutorial** Exercises
- Q1 and Q3 from **Week-06 Tutorial** Exercises
- Q2 from **Week-12 Tutorial** Exercises
- questions on a wide variety of the topics from the course

Also see: **Practice Questions for the Final Exam (includes many questions for part-1)**

Part-2

Part-2: Question 01

The following two files will be available in the directory named "q1":

- isLeapYear.c
- test_isLeapYear.c

You need to implement a C function `isLeapYear`, in the given file `isLeapYear.c`, that takes a year as an argument and calculates whether that year is a leap year or not. The function must return 1 if the year is a leap yer, 0 otherwise.

```
int isLeapYear(int year)
```

For example,

```
isLeapYear(2018)          returns   0
isLeapYear(2000)          returns   1
isLeapYear(1900)          returns   0
```

For this question, you need to use the provided file `test_isLeapYear.c` to test your function. Please open the file `test_isLeapYear.c` using an available text editor, try to understand how your function is tested.

You can use the following commands to compile and run few tests already provided in the file `test_isLeapYear.c`. Please make sure that you also test your function extensively by adding more test cases in `test_isLeapYear.c`. You will be submitting only one file `isLeapYear.c`. To run the simple tests available in `test_isLeapYear.c`, execute the following commands:

```
$ dcc -o test_isLeapYear    isLeapYear.c    test_isLeapYear.c
$ ./test_isLeapYear
```

Again, please make sure that you test your function extensively by adding more test cases in `test_isLeapYear.c`.

You will be submitting only one file `isLeapYear.c` for this question. Submit your file `isLeapYear.c` with the following command, from "q1" directory:

```
$ submit q1
```

Part-2: Question 02

Write a program called `outliers.c` that reads integer values from standard input until the end of input stream or first unsuccessful read. The program finds number of values that are outside the range of 5 to 25 (inclusive), and prints the number of such outliers.

Make your program match the examples below exactly.

```
$ ./outliers
Enter number: 12
Enter number: 4
Enter number: 25
Enter number: 15
Enter number: 27
Enter number: Ctrl + D
Outliers: 2


$ ./outliers
Enter number: 0
Enter number: 26
Enter number: 5
Enter number: -2
Enter number: Ctrl + D
Outliers: 3
```

When you think your program is working you can use `autotest` to run some simple automated tests. From the directory "q2", run the following command:

```
$ ./autotest
```

Please make sure that you test your program extensively by considering more test cases.

You will be submitting only one file `outliers.c` for this question. Submit your file `outliers.c` with the following command, from "q2" directory:

```
$ submit q2
```

For other examples: see **Practice Questions for the Final Exam**

Part-2: Question 03
(array hurdle question)

The following two files will be available in the directory named "q3":

- arrayPositiveMin.c

- test_arrayPositiveMin.c

You need to implement the following function that returns minimum positive value from a given array `a` (of type `int`). A value is a positive value if it is greater than zero. If there are no positive values in a given list, the function should return zero.

```
int arrayPositiveMin(int a[], int size)
```

For example,

```
arrayPositiveMin([4, 2, 9], 3)                      returns  2
arrayPositiveMin([5, -6, 3 , 20], 4)                returns  3
arrayPositiveMin([5, -6, -12 , 5, 0, -2 , 8], 7)    returns  5
arrayPositiveMin([-2, -4, -8, -2], 4)               returns  0
```

For this question, you need to use the provided file `test_arrayPositiveMin.c` to test your function. Please open the file `test_arrayPositiveMin.c` using your favourite text editor, try to understand how your function is tested. If you have any questions, please ask your tutor.

You can use the following commands to compile and run few tests already provided in the file `test_arrayPositiveMin.c`. Please make sure that you also test your function extensively by adding more test cases in `test_arrayPositiveMin.c`. To run the simple tests available in `test_arrayPositiveMin.c`, execute the following commands:

```
$ dcc -o test_arrayPositiveMin    arrayPositiveMin.c    test_arrayPosi
tiveMin.c
$ ./test_arrayPositiveMin
```

Again, please make sure that you test your function extensively by adding more test cases in `test_arrayPositiveMin.c`.

You will be submitting only one file `arrayPositiveMin.c` for this question. Submit your file `arrayPositiveMin.c` with the following command, from "q3" directory:

```
$ submit q3
```

For other examples: see **Practice Questions for the Final Exam**

Part-2: Question 04
(array hurdle question)

This question will be related to Lab07 exercises. In particular, Exercise-02 that requires you to implement the following function, read Lab07 for more explanation.

```
void species_count(char species[], int n_sightings, struct pod sightings[n_sightings], int *n_pods, int *n_whales) {
    // REPLACE THIS COMMENT WITH YOUR CODE
    // THIS FUNCTION SHOULD NOT CALL SCANF OR PRINTF
    // IT SHOULD JUST ASSIGN VALUES to N_PODS AND N_WHALES
    *n_pods = 24; // CHANGE ME
    *n_whales = 42; // CHANGE ME
}
```

By the way, the sample solutions for Lab07 are also available (see Labs).

For other examples: see **Practice Questions for the Final Exam**

Part-2: Question 05
(linked-list hurdle question)

The following two questions will be related to the examples discussed in the Lectures on **Linked List**, and the related lab and tutorial quesrions. You can either use an iterative approach or a recursive approach,

- Linked List ,
- Recursion, Linked List with Recursion,
- code: Linked lists, recursion (wk07.zip).

This question will be related to Lab08 exercises. In particular, Exercise-02 that requires you to implement the following function, read Lab08 for more explanation.

```
void species_count(char species[], struct pod *first_pod, int *n_pods, int *n_whales) {
    // REPLACE THIS COMMENT WITH YOUR CODE
    // THIS FUNCTION SHOULD NOT CALL SCANF OR PRINTF
    // IT SHOULD JUST ASSIGN VALUES to N_PODS AND N_WHALES
    *n_pods = 24; // CHANGE ME
    *n_whales = 42; // CHANGE ME
}
```

By the way, the sample solutions for Lab08 are also available (see Labs).

Other examples include:

- find the maximum item in a list
- find the index of the maximum item in a list
- calculate sum of a list
- find an item (say 25) in a list
- find last item in a list
- find n^{th} item in a list
- print a list in python style

For other examples: see **Practice Questions for the Final Exam**

Part-2: Question 06
(linked-list hurdle question)

For example, you may need to implement a function similar to one of the following functions (most of them discussed in the lectures/tutorials/labs):

- remove largest value from a list
- remove odd values from a list
- remove a given item (say 25) from a list
- add item at the end of the list
- add item at the start of the list
- add item to an ordered list
- add item after n^{th} item in a list
- remove first item from a list
- remove last item from a list
- [from week-12 tutorial, Q4] given two lists, append one List to another
- [from week-12 tutorial, Q4] given two lists, create a list by interleaving ('zip') nodes (values) from two lists

For other examples: see **Practice Questions for the Final Exam**

Part-2: Question 07+

Complex programming using any of the features covered in course.

-- end --