COMP1511 18s2

Practice Questions for the Final Exam

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

To prepare for your final exam, you should try to solve as many problems as possible on the page. I recommend you should try to solve all of them! Later we will release solutions for some of the problems, however, please note that we will not release solutions for all the questions. Best way to learn is to try to solve a problem before reading it's possible solution.

The following questions use this datatype from lectures:

```
struct node {
    int data;
    struct node *next;
};
```

See list_empty.c and list.h for some of the questions on linked list below, and testList.c for autotests. Alternatively download all three files at tut_list_files.zip.

1. [for Part-2] : Implement a function *ordered* which returns 1 if a linked list is in (strictly) increasing order; 0 otherwise. It should have this prototype:

   ```
   int ordered(struct node *head);
   ```

2. [for Part-2] : Implement a function *identical* that returns 1 if the contents of the two linked lists are identical (same length, same values in data fields) and otherwise returns 0. It should have this prototype:

   ```
   int identical(struct node *head1, struct node *head2);
   ```

3. [for Part-2] : Implement a function *copy* which returns a copy of a linked list. It should have this prototype.

   ```
   struct node *copy(struct node *head);
   ```

   It should call malloc to create a new linked list of the same length and which contains the same data.

4. [for Part-2] : Implement a function *set_intersection* which given two linked lists in strictly increasing order returns a new linked list containing a copy of the elements found in both lists.
   The new linked list should also be in strictly increasing order. It should include only elements found in both lists.

   set_intersection should call malloc to create the nodes of the new linked list.

   set_intersection should have this prototype:

   ```
   struct node *set_intersection(struct node *set1, struct node *set2);
   ```

5. [for Part-2] : Write a C program `thirteen_stdin.c` which reads 2 integers from standard input and then prints all integers divisible by 13 between those numbers.
   Your program should behave like this:

   ```
   $ ./a.out
   Enter start: 10
   Enter finish: 42
   13
   26
   39
   ```

6. [for Part-2] : Modify the previous C program so that it instead takes 2 integers as command line arguments
   Your program should behave like this:

   ```
   $ ./a.out 10 42
   13
   26
   39
   ```

7. [for Part-2] : Exam questions typically specify no error checking required.
   If error checking was required - what checking would you add to the programs from the previous 2 questions?

8. [for Part-2] : Write a program `median.c` which reads integers until end-of-input is reached. It should then print the median (middle) of the integers. If there are an even number of integer you can print either of the two middle integers.
   Assume the numbers of integer is > 0 and < 1000.

Assume the integer are entered in sorted (non-decreasing) order.

Your program should behave like this:

```
$ ./a.out
1
2
4
8
16
5 numbers read. Median was 4
```

9. [for Part-2] : Modify the program from the previous question to check that the numbers of integers supplied is > 0 and < 1000, and to check they are in sorted (non-decreasing) order.

10. [for Part-2] : Write a function that takes in a 2d array of ints and multiplies every value in the array by a given int.

11. [for Part-2] : Write a function, prototype below, that takes a string, and a character and removes the first occurrence of that character from the string. It should return 1 if the letter was found and removed, 0 otherwise. Write a main function that could test this function.

```
int remove_char(char str[], char c)
```

12. [for Part-2] : Write a function that takes 2 strings as arguments and returns the length of their common prefix.
For example "carpark" and "carpet" have a common prefix length of 4.

13. [for Part-1] : Write a function that takes an array of pointers to strings and prints out all the strings with more than a given number of characters. The prototype should be

```
// text - the array of strings
// array_size - the number of strings in the array
// num_chars - print out any strings in the array with more than this number
// of characters
void print_if_longer(int array_size, char text[array_size][MAX_LEN], int num_chars);
```

14. [for Part-1] : What would be the output of the following code?

```
int x = -9;
int *p1 = &x;
int *p2;

p2 = p1;
printf("%d\n", *p2);
*p2 = 10;
printf("%d\n",x);
```

15. [for Part-1] : What would be the output of the following code?

```
int x = -9;
int y = 0;

while (x != 0){
    y = y - 1;
    x = x + 1;
}

printf("%d\n", x);
printf("%d\n",y);
```

16. [for Part-1] : What would be the output of the following code?

```
int i = -7;
int j = 0;

while (i != 0){
    j = j - i;
    i = i + 1;
}

printf("%d\n", i);
printf("%d\n",j);
```

17. [for Part-1] : Given the following code fragment:

```
char goals[] = "All your goals belong to us.";
char *a, *b, *c;

a = goals + 5;
b = &goals[10];
c = goals + (b - goals) + (b - a);
```

The fragment is valid C. It executes without error. Indicate clearly and exactly what the following expressions evaluate to:

1. `a == goals`
2. `a > goals`
3. `goals > c`
4. `c - b`
5. `goals - a`
6. `a[0] != b[0]`
7. `*c`
8. `goals[a - goals] == *a`
9. `c[a - b]`

18. [for Part-1] : Given the following code fragment:

```
int i = 0;
int j = 0;
char *s = "ceded";

while (s[i] != '\0') {
  j = j + s[i] - 'a';
  i = i + 1;
}
printf("%d %d\n", i, j);
```

The fragment is valid C. It executes without error. Indicate clearly and exactly what output will be printed.
