# Problem Set 2

This problem set is due **at 10:00pm** on **Thursday, October 12, 2023**.

The idea of the problem set is to get you comfortable with coding in `C` and making submissions to github. We've tried to keep it light.

Please make note of the following instructions:

- This assignment, like later assignments, consists of *exercises* and *problems*. **Hand in solutions to the problems only.** However, we strongly advise that you work out the exercises for yourself, since they will help you learn the course material. You are responsible for the material they cover.
- Remember that the problem set must be submitted on Gradescope. You can join using the code: 6GZKVG.
- We require that the solution to the problems is submitted as a PDF file, **typeset on LaTeX**, using the template available in the course materials. Each submitted solution should start with your name, the course number, the problem number, the date, and the names of any students with whom you collaborated.
- Code submissions will be over GitHub Classroom.
- You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of your essay should provide the following:
    1. A description of the algorithm in English and, if helpful, pseudocode.
    2. A proof (or indication) of the correctness of the algorithm.
    3. An analysis of the asymptotic running time behavior of the algorithm.
    4. Optionally, you may find it useful to include a worked example or diagram to show more precisely how your algorithm works.
- Late submissions will be penalised at the rate of **5% per hour** of the assignment credit.

**Problem 2-1.** Pointer Arithmetic [50 points]

The idea of this problem is to get you familiar with pointers in C. Take a look at the following code snippets, run them and tell us the output (in a verbatim block), and clearly **justify** the output. *Hint:* Play around with the inputs to try and figure out what is actually happening.

**(a)** [5 points] On my machine, the following loop prints -1808407282 . Why does it stop?

```
int i;
int sum = 0;
for (i = 0; i -= 20; i++)
{
    sum += i;
}
printf("%d\n", sum);
```

**(b)** [5 points] What is the following code doing? What will be the output of this program if the user input abxyba?

```
char str[100];
int i;
scanf("%s", str);
for (i = 0; str[i] != NULL; i++)
{
    if (str[i] != 'x' || str[i] != 'y')
    {
        printf("%c", str[i]);
    }
}
```

**(c)** [5 points] What will be the output of the following C program and why?

```
void doSomething(int* blah)
{
    *blah = 420;
}

int main()
{
    int x = 42;
    int* notblah = &x;
    printf("out1: %d\n", &x);
    doSomething(notblah);
    printf("out2: %d\n", x);
    printf("out3: %d", &x);
    return 0;
}
```

**(d)** [5 points] What will be the output of the following this program and why?

```c
const int a = 15;
const int *ptr;
ptr = &a;
*ptr = 47;
printf("%d\n", a);
return 0;
```

**(e)** [5 points] What will be the output of the following C program and why?

```c
char x[25];
char *s = "Please learn basics of C!";
int length = strlen(s);
int i;
for (i = 0; i < length; i++)
{
    x[i] = s[length - i];
}
printf("%s", x);
```

**(f)** [5 points] What will be the output of the following C program and why?

```c
char a = '99';
printf("out1: %d\n", a);
int x = 'A';
printf("out2: %d", x);
```

**(g)** [5 points] What will be the output of the following C program and why?

```c
int i = 2984;
int* j;
j = &i;
i = i + *j;
printf("%d", i);
```

**(h)** [5 points] What will be the output of the following C program and why?

```c
struct learnC {
    int a,b,c;
};

int main()
{
char *cptr = (char *) 0x5000;
int *iptr  = (int *) 0x7000;
struct learnC *sptr = (struct learnC *) 0x9000;

printf("char   0x%02lx %p %p\n", sizeof(char), cptr, (cptr + 1));
printf("int    0x%02lx %p %p\n", sizeof(int), iptr, (iptr + 1));
printf("struct 0x%02lx %p %p\n", sizeof(struct learnC), sptr, (sptr
   + 1));
return 0;
}
```

**(i)** [5 points] What will be the output of the following C program and why?

```
char c[]="LATENCY";
char *p=c;
printf("%s",p+p[3]-p[1]);
```

**(j)** [5 points] What will be the output of the following C program and why?

```
int arr[5] = {7, 19, 29, 0, 5};
int *p = arr;
int **pp = &p;
printf("%d\n", *p + **pp);
```

**Problem 2-2.** Why BSTs? [20 points]

In the last assignment you saw how a binary search tree was more efficient at solving the runway problem because inserting elements into a BST also meant implicitly sorting them.

Prove that an in-order traversal of a binary search tree always produces the elements of the tree in sorted order.

**Problem 2-3.** Linked Lists [50 points]

In the first classes we discussed linked lists, one of the most rudimentary data structures. This question will test your ability to create and manipulate linked lists. As usual the code is to be submitted on GitHub Classroom, you can access the assignment by clicking here.

Like last time, to compile code in this assignment properly run `make` from the assignment's directory. This will output an executable `asmnt2` that requires an integer input parameter. For example, to execute this assignment you could run `./asmnt2 5`. To clear the compiled files you can run `make clean`, it might make sense to do this before pushing your answers. The `make clean` command might fail on windows – just ignore it.

The integer `5` acts as a seed to generate random numbers using the Mersenne Twister algorithm. Any given seed will always generate the same set of random numbers in the same order, which will allow you to test the code you write easily, and allow us to test your code with our own set of random numbers. Use the same 16 random numbers in the order in which they are stored in the array to add to your data structures.

**(a)** You have an array of random numbers. Create a linked list and fill these elements into the list. Once you are done adding them to the list print out the list using the `printList()` function on a new line in the format `Linked List: a b c d` where the letters represent elements of the list. [5 points]

**(b)** Unlike arrays, linked lists do not require you to store the length of the list, thus providing flexibility when adding elements. You are asked to print exactly half a linked by traversing the list exactly once, without knowing the length of the list before hand. Describe the algorithm you would use here and then code up the solution in the function `printHalfList()`. Print the output in the form `Half List: a b c d` as described earlier on a new line. [10 points]

**(c)** Another advantage of linked lists is that it is fairly easy to rearrange elements by reassigning pointers. Write a function using insertion sort to sort your linked list and print out the list again in the format `Sorted List: a b c d` on a new line. [10 points]

**(d)** `C` requires you to manage memory usage on your own. Look up the `free` function that `C` provides and explain what its purpose is. [5 points]

**(e)** Linked lists make it fairly easy to delete nodes in the middle of the list. Write a function that deletes the element at the $k$th position in the list and use it to delete the 5th element in your list. You will need to use the `free` function to do this properly. Print out the new list as `Updated: a b c d` on a new line. [10 points]

**(f)** Write a function that reverses your linked list in one traversal and without creating a new list. Print out the resulting reversed list as on a new line in the format `Reversed: a b c d`. [10 points]

**Problem 2-4.** Bonus Section [15 points]

Sort your original linked list using merge-sort and print the output as `Merge Sort: a b c d` on a new line.

Here is a sample output on a smaller list to help you:

```
Linked list: 2 8 5 4 3 9
Half list: 2 8 5
Sorted list: 2 3 4 5 8 9
Updated: 2 3 4 5 9
Reversed: 9 5 4 3 2
Merge Sort: 2 3 4 5 8 9
```