CMPUT 350 Lab 0 Exercise Problems

Rules:

- You can use all course material and man pages, but no other information such as web pages, books, or written notes. Using other information sources during the exercise constitues cheating.
- Your programs must compile without warning using

  ```
  gcc -g -Wall -Wextra -Wconversion -Wsign-conversion -O ...
  ```

  In case there are compiler warnings or errors, marks will be deducted.
- Test your programs with different values. For now, the speed of your program is irrelevant. So don't spend time on optimization
- You must check for the appropriate preconditions/postconditions. Your program shouldn't crash or have undefined behaviour (**hint**: use asserts)!
- Your programs must be well structured and documented. Use ctrl-x t in Emacs to pretty-print it. Marks are assigned to functionality, program appearance, and comments.
- In case your program hangs, use ctrl-c to terminate it.
- Remember that you need to include the appropriate header files. To find out which ones you need for specific functions such as printf, use then man command.

Submit your solution files `p1.c p2.c` on eClass under "Lab 0 / Submission".

Important: Submit often (the system will only accept solutions submitted before 16:50)

---

1. [11 marks] In file `p1.c` implement the following functions:

```c
// precond: n > 0
void simple_sort(int A[], int n) {
    // ...
}

// Test code
// check whether simple_sort works by setting up an array
// and printing its elements after sorting
void test() {
    // ...
}
```

`simple_sort` rearranges elements in array `A` in non-decreasing order. In function `main()` test `simple_sort` by calling it with an initialized array and printing the sorted result. SimpleSort works by iteratively determining the minimum element, swapping it with the first element, shortening the array by one element, and continuing the process until there is no work left to do.

We provide skeleton files, along with a `main` which calls your `test` function. To compile the files,

```
gcc -g -Wall -Wextra -Wconversion -Wsign-conversion -O p1.c p1_main.c
```

2. [14 marks] Files `p2.h` and `p2.c` contains an empty linked-list structure implementation. The end of a list is marked by a node with `next = 0`. In file `p2.c`, implement the functions using `malloc`/`free` (**NOT** `new`/`delete`!)

We provide skeleton files, along with a `main` which calls your `test` function. To compile the files,

```
gcc -g -Wall -Wextra -Wconversion -Wsign-conversion -O p2.c p2_main.c
```