Systems Programming: Practical 8
Function Pointers

If you haven't finished Practical 6 yet, do that before attempting this one.

A Array of functions

Write four functions, which take two integers as parameters and give the result after adding, subtracting, multiplying or dividing these numbers, respectively. Create an array of four pointers to functions (hint: you can use `int (*ope[4])(int, int);` to declare such an array) and assign the elements of the array to point to your four functions.

Create a program that asks the user to input two integers, and then choose a number `0-3` of which function to apply to these two numbers (e.g. `0` could be `add()`, `1` could be `sub()` etc.). Call the function by using the element of the array at the corresponding index and output the result.

B Optional: InsertionSort

The InsertionSort algorithm can be used to sort an array `A` on `n` elements. It loops through the indices `i` of the input array (from `1` to `n-1`), and in each round moves elements so that the elements `A[0]...A[i]` end up in sorted order. See `https://en.wikipedia.org/wiki/Insertion_sort` if your are unfamiliar with InsertionSort. Implement the InsertionSort algorithm as a function `isort()` that takes the same parameters as `qsort`, i.e. it should have the following function declaration:

```
void isort(void *base, size_t nmemb, size_t size, int (*compar)(const void *, const void *));
```

- `void *base` is a pointer to the array
- `size_t nmemb` is the number of elements in the array
- `size_t size` is the size of each element
- `int (*compar)(const void *, const void *)` is a function pointer composed of two arguments and returns `0` when the arguments have the same value, `<0` when `arg1` comes before `arg2`, and `>0` when `arg1` comes after `arg2`.

(Hint: You may want to use a `char *` pointer to access individual bytes. Alternatively, you could use `memcpy()`.) Write a program which uses your `isort()` function to sort an array of integers.