Practical class 9

General comments

The objectives of this ninth lab session are:
- (i) To practise declaring, initialising and using pointers.
- (ii) To understand simple pointer arithmetic, particularly the effect of adding one to a pointer.
- (iii) To learn how to pass pointers to functions and how this can be used to effectively pass by reference.
- (iv) To practise how to dynamically allocate and free memory.

Instructions

1. First, we will write a simple program to practise the basic use of pointers, including how to declare, initialise and print them. To do this:

   a) Create a `char` variable called `c` that is initialised to `'S'`. Then create a `char`-pointer called `pc` that is initialised to point to `c`. Now print `c`, `pc` and `*pc`.

   b) Now increment `c` by 1 and again print `c`, `pc` and `*pc`.

   c) Then increment `pc` by 1 and again print `c`, `pc` and `*pc`.

In each case check that the output is what you expect.

Now repeat the above with (i) an `int` variable `i` that is initialised to `91` and (ii) a `double` variable `d` that is initialised to `3.14159`. In each case, try to predict the result before running the code.

2. We will now write and test a function that converts single characters to uppercase. Define a function called `to_upper_case()` that takes a pointer-to-`char` as its only argument and does not return anything.

   a) The first thing the function should do is to return without doing anything if the character is not a lowercase letter. Hint: in ASCII the lowercase letters lie between 97 (`'a'`) and 122 (`'z'`).

   b) Next, for lowercase letters, the function should convert the character to uppercase. Hint: the uppercase letters have ASCII values 32 lower than their lowercase equivalents (e.g. 65 is `'A'` and 90 is `'Z'`).

   c) Test your function on single characters by calling `to_upper_case()` from `main()`.

   d) Finally, test your function on a string. Define the string in `main()` and loop over its characters, calling `to_upper_case()` in each case.

3. Finally, we will create a program to calculate the first N square numbers, where N is supplied by the user at runtime.

   a) Read N from the user, making sure to check that it is a positive integer.

   b) Use `malloc()` to allocate space for N `long` numbers. Don't forget to check whether the allocation fails by testing whether the returned pointer is `NULL`.

   c) Calculate and store the first N squares in the allocated memory: 1, 4, 9, …, N^2. Then loop over the allocated memory and print out the first N square numbers.

   d) Finally, release the memory using `free()`, not forgetting to then set the pointer to `NULL`.

   e) Check that your program works correctly for N=10^6. How much dynamic memory do you use in this case?