Programming for Engineers Portfolio : Quarter 2, Set 2

1. The file `e1_reverse_static.cpp` contains a program that reads a sequence of values into an array and prints them in reverse order. Create an input file `e1_overflow.txt` that contains the smallest sequence of values that will overflow the array. The size of the file does not matter, nor the specific values used - we are only concerned with the length of the sequence.

   Note: you should be able to work it out by inspection of the code, but you might find it useful to try compiling with `-fsanitize=undefined` to see if the error gets detected.

2. Create a new file `e2_reverse_dynamic.cpp` which uses the same logic as `e1_reverse_static.cpp`, but which uses a dynamically allocated buffer via `new[]` to handle sequences of any size.

   Note: You are not allowed to include `<vector>` for this task, you need to use explicit dynamic memory.

---

3. Write a program `e3_is_sorted.cpp` which reads a sequence of values, and exits with a success code if the sequence is sorted, and exits with the failure code 1 if the sequence is not sorted. A sequence of values $x_1, x_2, ..., x_n$ is sorted if $x_i \leq x_{i+1}$ for $1 \leq i < n$. An empty sequence is always considered sorted.

4. The program `e4_sort.cpp` reads a sequence of values and prints them back out in the same order. Use the function `sort` from `<algorithm>` to sort the data before it is printed. For our purposes you can treat the input type `RandomAccessIterator` as a pointer, so you need to give `sort` two pointers defining the half-open range $[first, last)$ to be sorted.

5. Write a script `e5_check_sort.sh` which uses `e3_is_sorted` to check whether `e4_sort` can correctly sort a sequence of values. The message "Success" should be printed via `echo` if (and only) if `e4_sort` passes the test. You are only required to include one test test-case, but it should be able to tell the difference between a working and non-working sort program.

The file `e6_circle.cpp` is intended to read a width and a height from `stdin`, and then output an image of concentric circles in the `pbm` image format, which is a textual representation of an image. At the moment the source file is missing some details and definitions for an `image` type, which represents images as an array of pixels. A pre-compiled reference version of the executable program is available as `e6_circle_ref`.

6. Complete the definition of the type `image` and the function `image_destroy`. Your definitions should be compatible with the existing definition of `image_create`. Once completed, your program should compile, but if executed it will print a white rectangle.

7. Add definitions of the functions `image_set_pixel` and `image_get_pixel`. A pixel at co-ordinate (x, y) in a width w and height h image should be stored in index $y \times w + x$ of the `pixels` array.

8. A `pbm` format image can be converted to a normal format like `jpg` through the use of imagemagick's convert function. Create a script `render_circle_as_jpg.sh` which first uses either `e6_circle` or `e6_circle_ref` to generate a 256x256 `pbm`, and then uses imagemagick's `convert` command to save it as a file `e6_256x256_circle.jpg`.

   You should assume imagemagick is already installed in the test environment. On your own machine you can install it with `sudo apt install imagemagick`.

   If needed, the script can create intermediate files with the prefix `e8_`.

---

9. The program `e9_filter.cpp` contains a function prototype for `find_by_age`. This function accepts a pointer to a vector of people, and should return a vector of person pointers to those elements with an age greater than the filter value. Provide a definition for the function.

10. Modify the `main` function of `e9_filter.cpp` to add test cases for `find_by_age`, such that if any errors are found in the function the program returns 1, and if no errors are found it returns 0.