

Lab 12: Function Pointers & Algorithm Complexity

Aim

This lab class gives you an opportunity to:

- complete your eVALUate survey(s) for the unit;
- write programs with function pointers;
- explore priority queues; and
- practise determining the complexity of algorithms.

Part 1 — eVALUate

Context

eVALUate is the online system for gathering and reporting student feedback on learning experiences. We value your feedback and urge you to provide it.

Task



<https://evaluate.utas.edu.au/>

Part 2 — Function Pointers

Context

Priority queues are queues in which items are stored in an order governed by their 'priority'. The highest priority items are at the front of the queue and the lowest priority items are at the back, i.e. items are stored in descending order of priority. It is the `add()` function that facilitates this; all other functions are unchanged from a queue. In order to know how to compare two 'things' and determine which is the most important, `add()` must be passed a function which does the comparison. Hence `add()`'s function header is:

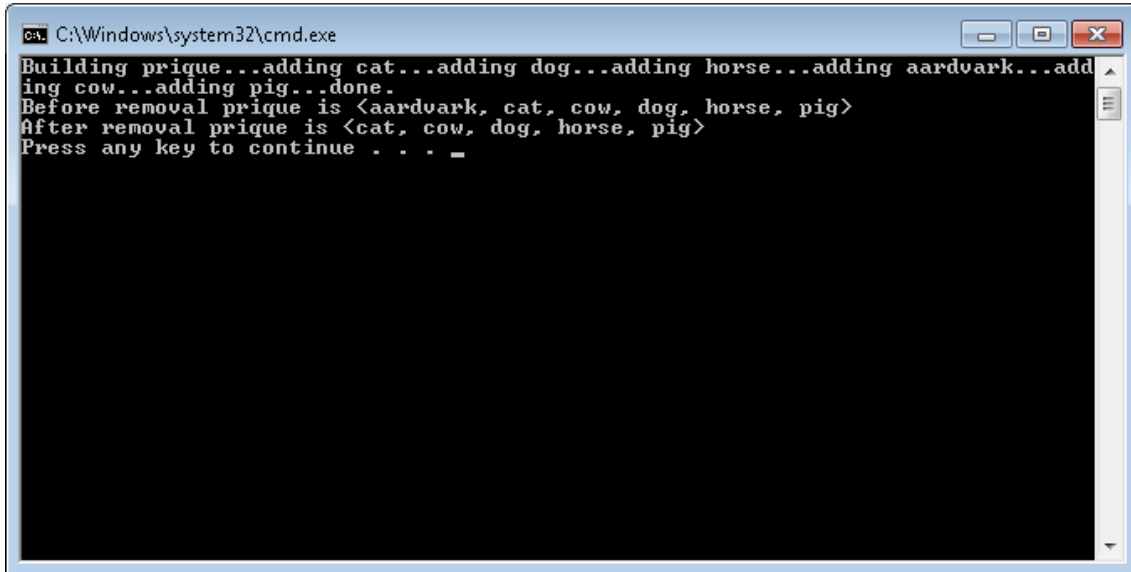
```
void add(priqueue p, void *o, bool (*greaterThan)(void *, void *));
```

Here `p` is the priority queue that the value `o` will be added to in an order determined by `greaterThan()` — which is a function accepting two 'things' and which returns `true` if the first is more important than the second and `false` otherwise.

The data structure being used to implement the ADT is a single-ended doubly-linked list. `priqueue` is a pointer to a struct containing a reference to the first node in the linked list. `dnode` is a pointer to a struct which contains a `data` field, a `next` field, and a `prev` field which refers to the node prior to the current one.

Tasks

1. Download the compressed project folder Lab12.zip from *MyLO* and after extracting all the files, open this project folder and open the project file (Lab12.sln).
2. Complete the implementation of the ADT by writing the add() function for the prique.c file.
3. Complete the driver application (Lab12.c) by writing the body of the alphabeticalOrder() function. *Hint*: alphabeticalOrder() can be implemented by calling strcmp().
4. Test that your implementation works. You should see the following output.



```
C:\Windows\system32\cmd.exe
Building prique...adding cat...adding dog...adding horse...adding aardvark...adding cow...adding pig...done.
Before removal prique is <aardvark, cat, cow, dog, horse, pig>
After removal prique is <cat, cow, dog, horse, pig>
Press any key to continue . . . _
```

Part 3 — Algorithm Complexity

Context

Algorithms can be examined to determine the ‘amount of work’ they perform so that they may be compared with other algorithms completing the same task. The amount of work can vary from a minimum (best case) to a maximum (worst case). These complexities can then be expressed in order (big-oh) notation.

Task

1. Consider the following sorting algorithm implemented in C.

Using “(a[i] > a[i+1])” as the fundamental operation, what are the best- and worst-case time complexities of the algorithm? What are these in big-oh notation? You might consider what happens when the list is already sorted and when the smallest item is at the rear of the list. Assume that the swap() function simply swaps the values at the given array locations.

```
void sort(int a[], int n)
{
    int i;
    bool exchange=true;

    while (exchange) {
        exchange=false;
        for (i=0; i<n-1; i++) {
            if (a[i] > a[i+1]) {
                swap(a,i,i+1);
                exchange=true;
            }
        }
    }
}
```