

CSCI 104L Lecture 6: Stacks and Queues

Stacks and Queues

We can define a stack recursively; a stack is either:

1. The empty stack, or
2. $S.\text{push}(\text{data})$, where S is a stack, and data is a data item.

The following “stack axioms” describe stack behavior:

1. For all stacks s , $s.\text{push}(\text{data}).\text{top}() = \text{data}$
2. For all stacks s , $s.\text{push}(\text{data}).\text{pop}() = s$

Question 1. Use the above 4 points to determine the result of the following operations:

- $s.\text{push}(5).\text{push}(4).\text{pop}().\text{top}()$
- $s.\text{push}(5).\text{top}().\text{pop}()$

An algorithm you would normally solve recursively can instead be solved with a stack.

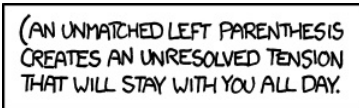
Question 2. Which strings are properly parenthesized?

1. $([ab])$
2. $ab\{\}$
3. $([ab\{c\}]de())$

You can solve the above problem using a stack; here is an outline:

1. If there are no more characters in the string, and the stack is empty, accept.
2. If there are no more characters in the string, and the stack is not empty, reject.
3. If the next character is an open parentheses, open brace, or open bracket, push it on the stack.
4. If the next character is a close parentheses, close brace, or close bracket, then pop the top item from the stack. If the top item from the stack is of a different type (or the stack was empty), reject.
5. Loop back to step 1

To actually calculate the value of the equation, push all characters (not just parentheses), and on step 4, pop back to the matching parentheses and evaluate the popped expression. Then push the result on the stack.



(AN UNMATCHED LEFT PARENTHESIS
CREATES AN UNRESOLVED TENSION
THAT WILL STAY WITH YOU ALL DAY.

Figure 1: XKCD # 859 “(“: Brains aside, I wonder how many poorly-written xkcd.com-parsing scripts will break on this title (or ;;”{<<[’ this mouseover text.

A queue can also be defined recursively; one is either:

1. The empty queue, or
2. `Q.enqueue(data)`, where `Q` is a queue, and `data` is a data item.

A **Deque** allows you to add and remove elements from both ends.

Question 3. How should a deque be implemented?

Question 4. Why stacks and queues, when Deques are more powerful?

STL's map class

```
#include<map>
#include "student.h"
int main() {
    map<string, Student> slist1;
    Student s1("Tommy", 86328);
    slist1["Tommy"] = s1; //associate the string Tommy with his student record.
    slist1.erase("Tommy");
    return 0;
}
```

STL's pair struct

```
std::pair<string, int> mypair("Tina", 1);
cout << mypair.first << " " << mypair.second << endl;
std::pair<char, double> p2('c', 2.3);
```

STL's iterator class

```
map<int, string> m;
map<int, string>::iterator it;
for (it = m.begin(); it != m.end(); ++it) {
    cout << it->second << endl;
}
it = m.find(42);
if (it != m.end()) cout << "meaning_of_life_found:" << it->second << endl;
```

- The data structure has two public functions: `begin()`, which returns an iterator at the start of the data, and `end()` which returns an iterator at the end of the data.
- Every iterator in the STL is implemented in the same manner, so that you can always use an iterator for a data structure, even though you may not understand how the data structure works.
- Think of it like a pointer (it is not a pointer, but it has overloaded operator* to act like one).
- Also, think of the `end()` function as returning one PAST the end of the data structure, so the above for-loop works properly.