Array-based stack implementation

Here we present the idea of stack implementation, based on arrays. We assume in current article, that stack's capacity is limited to a certain value and overfilling the stack will cause an error. Though, using the ideas from dynamic arrays implementation, this limitation can be easily avoided (see capacity management for dynamic arrays).

In spite of capacity limitation, array-based implementation is widely applied in practice. In a number of cases, required stack capacity is known in advance and allocated space exactly satisfies the requirements of a particular task. In other cases, stack's capacity is just intended to be "big enough". Striking example of the last concept is an application stack. It's capacity is quite large, but too deep recursion still may result in stack overflow.

Implementation

Implementation of array-based stack is very simple. It uses **top** variable to point to the topmost stack's element in the array.

1. Initialy **top = -1;**
2. **push** operation increases top by one and writes pushed element to **storage[top];**
3. **pop** operation checks that top is not equal to -1 and decreases top variable by 1;
4. **peek** operation checks that top is not equal to -1 and returns **storage[top];**
5. **isEmpty** returns boolean **(top == -1).**

Code snippets

Java implementation

```
public class Stack {
      private int top;
      private int[] storage;

      Stack(int capacity) {
            if (capacity <= 0)
                  throw new IllegalArgumentException(
                             "Stack's capacity must be positive");
            storage = new int[capacity];
            top = -1;
      }

      void push(int value) {
            if (top == storage.length)
                  throw new StackException("Stack's underlying storage is overflow");
            top++;
            storage[top] = value;
      }

      int peek() {
            if (top == -1)
                  throw new StackException("Stack is empty");
            return storage[top];
      }

      void pop() {
            if (top == -1)
                  throw new StackException("Stack is empty");
            top--;
      }
      boolean isEmpty() {
            return (top == -1);
      }

      public class StackException extends RuntimeException {
            public StackException(String message) {
                  super(message);
            }
      }
}
```

C++ implementation

```
#include <string>
using namespace std;

class Stack {
private:
      int top;
      int capacity;
      int *storage;
public:
      Stack(int capacity) {
            if (capacity <= 0)
                  throw string("Stack's capacity must be positive");
            storage = new int[capacity];
            this->capacity = capacity;
            top = -1;
      }

      void push(int value) {
            if (top == capacity)
                  throw string("Stack's underlying storage is overflow");
            top++;
            storage[top] = value;
      }

      int peek() {
            if (top == -1)
                  throw string("Stack is empty");
            return storage[top];
      }

      void pop() {
            if (top == -1)
                  throw string("Stack is empty");
            top--;
      }

      bool isEmpty() {
            return (top == -1);
      }

      ~Stack() {
            delete[] storage;
      }
};
```

Previous: Stack ADT Next: Linked list based stack implementation

Like 18 people like this.

Contribute to AlgoList

Liked this tutorial? Please, consider making a donation. Contribute to help us keep sharing free knowledge and write new tutorials.

Donate
PayPal

Every dollar helps!

Leave a reply

Your name (optional):	
Your e-mail (optional):	
Message:	

Send