Cheatography

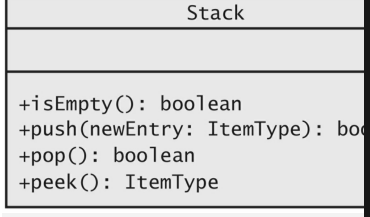
# C++ Data Structures & Algorithms Cheat Sheet

by mustang_balcony0a via cheatography.com/184329/cs/38436/

## Stack ADT Performance

| Time Complexity | Push | Peek | Pop |
|---|---|---|---|
| Worst Case | O(1) | O(1) | O(1) |

## Stack ADT UML




## Stack ADT Interface

```cpp
#ifndef __ISTACK_H__
#define __ISTACK_H__

template <class T>
class IStack {
public:
    virtual bool isEmpty() const = 0;

    virtual bool push(const T& newEntry) = 0;

    virtual bool pop() = 0;

    virtual T peek() const = 0;
};

#endif // _ISTACK_H__
```

## Array Stack ADT Interface

```cpp
#ifndef __ARRAY_STACK_H__
#define __ARRAY_STACK_H__

#include "IStack.h"

constexpr int MAX_STACK_SIZE =

template<class T>
class ArrayStack : public IStac
private:
    T items[MAX_STACK_SIZE];
    int top;

public:
    ArrayStack();
    bool isEmpty() const;
    bool push(const T& newEntry
    bool pop();
    T peek() const;
}
```

## Array Stack ADT Implementation

```cpp
template<class T>
ArrayStack<T>::ArrayStack() : top(-1)

template<class T>
bool ArrayStack<T>::isEmpty() const {
    return top < 0;
}

template<class T>
bool ArrayStack<T>::push(const T& newE
    bool success = false;
    if (top < MAX_STACK_SIZE - 1) {
        top++;
        items[top] = newEntry;
        success = true;
    }
    return success;
}

template<class T>
bool ArrayStack<T>::pop() {
    bool success = false;
    if (!isEmpty()) {
        top--;
        success = true;
    }
    return success;
}

template<class T>
T ArrayStack<T>::peek() const {
    return items[top];
}

#endif // __ARRAY_STACK_H__
```

## Linked-List Stack ADT

```cpp
#ifndef __LINKED_STACK_H__
#define __LINKED_STACK_H__

#include "IStack.h"
#include "Node.h"

template<class T>
class LinkedStack : public IStack<T> {
private:
    Node<T>* topPtr;

public:
    LinkedStack();
    LinkedStack(const LinkedStack<T>& oth
    virtual ~LinkedStack();

    bool isEmpty() const;
    bool push(const T& newItem);
    bool pop();
    T peek() const;

}

#include "LinkedStack.cpp"

#endif // __LINKED_STACK_H__
```

## Linked-List Array ADT Implementation I

```cpp
template<class T>
LinkedStack<T>::LinkedStack() : topPtr(nullptr) { }

template<class T>
LinkedStack<T>::LinkedStack(const LinkedStack<T>& otherPtr) {
    Node<T>* otherTopPtr = otherPtr -> topPtr;

    if (!otherTopPtr) {
        this -> topPtr = nullptr;
    } else {
        this -> topPtr = new Node<T>();
        this -> topPtr -> setItem(otherTopPtr -> getItem());

        Node<T>* newTopPtr = this -> topPtr;
        while (otherTopPtr) {
            otherTopPtr = otherTopPtr -> getNext();
            T nextItem = otherTopPtr -> getItem();
            Node<T>* newNodePtr = new Node<T>(nextItem);
            newTopPtr -> setNext(newNodePtr);
            newTopPtr = newTopPtr -> getNext();
        }
        newTopPtr -> setNext(nullptr);
    }
}

template<class T>
LinkedStack<T>::~LinkedStack() {
    while (!isEmpty()) {
        pop();
    }
}

template<class T>
bool LinkedStack<T>::isEmpty() const {
    return !topPtr
}
```

## Linked-List ADT Implementation II

```cpp
template<class T>
bool LinkedStack<T>::push(const T& newItem) {
    Node<T>* newNodePtr = new Node<T>(newItem, topPtr);
    topPtr = newNodePtr;
    newNodePtr = nullptr;
    return true;
}

template<class T>
bool LinkedStack<T>::pop() {
    bool success = false;
    if (!isEmpty()) {
        Node<T>* nodeToDeletePtr = topPtr;
        topPtr = topPtr -> getNext();

        nodeToDeletePtr -> setNext(nullptr);
        delete nodeToDeletePtr;
        nodeToDeletePtr = nullptr
        success = true;
    }
    return success;
}

template<class T>
T LinkedStack<T>::peek() const {

    if(isEmpty()) {
        throw PrecondViolatedExcep("peek() called with empty stack");
    }

    return topPtr -> getItem();
}
```