

# STACK C++ STL & TEMPLATES INTERVIEW PRACTICE

---

Problem Solving with Computers-II

C++

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

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```

# Announcements

- Pa02 released!
  - Its about implementing a BST with a movie data set, collecting and analyzing running time!
  - Part of the assignment involves writing a report, explaining the trends in your data
  - Start early!
- Midterm grades released!
  - Max: 100% (13 students)
  - Median: 87%
  - Mean: 85%

# Stack ADT

- Uses the Last In First Out (LIFO) principle
- Methods
  - i. push()
  - ii. pop()
  - iii. top()
  - iv. empty()

# C++STL

- The C++ Standard Template Library is a handy set of three built-in components:
  - Containers: Data structures
  - Iterators: Standard way to search containers
  - Algorithms: These are what we ultimately use to solve problems

# C++ STL container classes

```
array  
vector  
forward_list  
list  
set  
stack  
queue  
priority_queue  
multiset (non unique keys)  
deque  
unordered_set  
map  
unordered_map  
multimap  
bitset
```

# Finding the Maximum of Two Integers

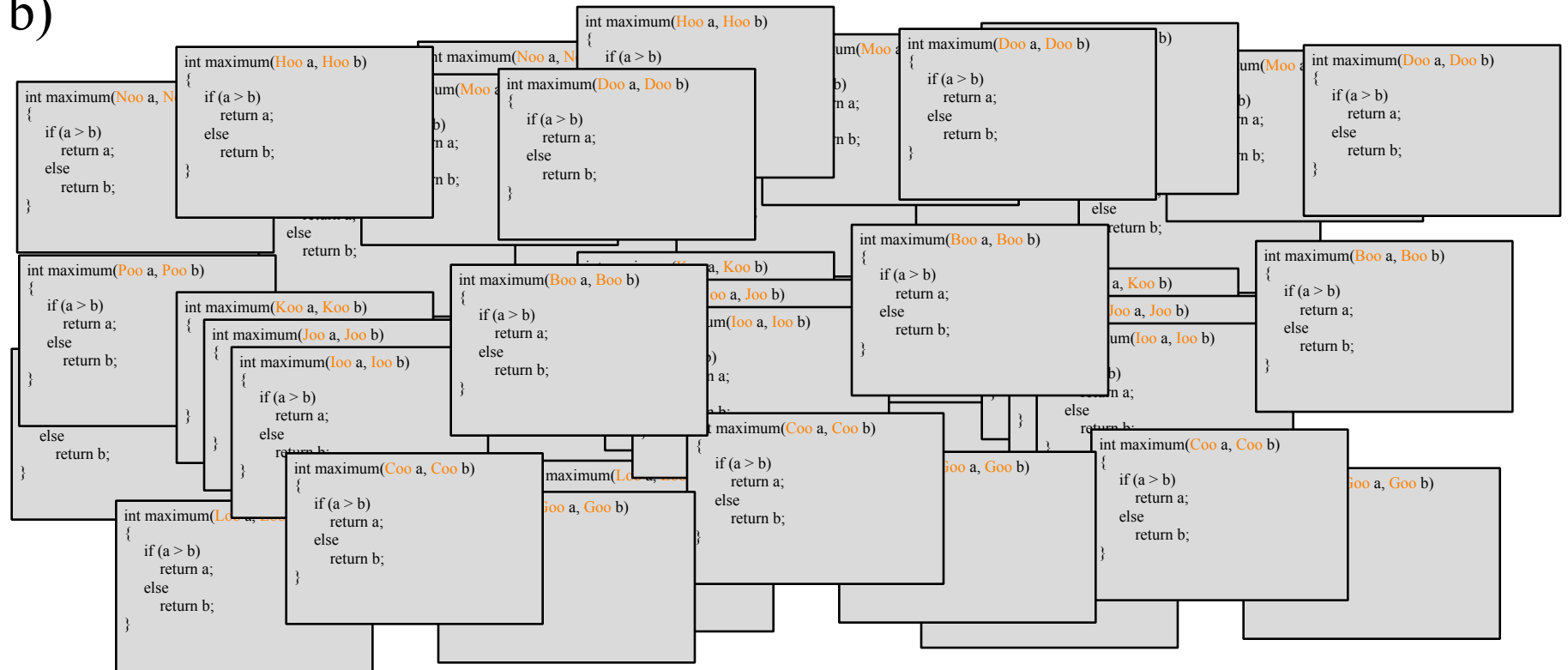
- Here's a small function that you might write to find the maximum of two integers.

```
int maximum(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

# One Hundred Million Functions...

Suppose your program uses 100,000,000 different data types, and you need a maximum function for each...

```
int maximum(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}
```



# A Template Function for Maximum

When you write a template function, you choose a data type for the function to depend upon...

```
template <class Item>
Item maximum(Item a, Item b)
{
    if (a > b)
        return a;
    else
        return b;
}
```



## Lab05 - BST with templates

BST, without templates:

```
class BSTNode {
public:
    BSTNode* left;
    BSTNode* right;
    BSTNode* parent;
    int const data;

    BSTNode( const int& d ) :
        data(d) {
            left = right
                = parent = nullptr;
        }

};
```

BST, with templates:

```
template<class Data>
class BSTNode {
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;

    BSTNode( const Data & d ) :
        data(d) {
            left = right
                = parent = nullptr;
        }

};
```

BST, with templates:

```
template<class Data>
class BSTNode {
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;
```

```
    BSTNode( const Data & d ) :
        data(d) {
        left = right = parent = nullptr ;
    }

};
```

How would you create a **BSTNode** object on the runtime stack?

- A. `BSTNode n(10);`
- B. `BSTNode<int> n;`
- C. `BSTNode<int> n(10);`
- D. `BSTNode<int> n = new BSTNode<int>(10);`
- E. More than one of these will work

{ } syntax OK too

BST, with templates:

```
template<class Data>
class BSTNode {
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;

    BSTNode( const Data & d ) :
        data(d) {
            left = right = parent = nullptr ;
        }

};
```

How would you create a **pointer** to BSTNode with integer data?

- A. BSTNode\* nodePtr;
- B. BSTNode<int> nodePtr;
- C. BSTNode<int>\* nodePtr;

BST, with templates:

```
template<class Data>
class BSTNode {
public:
    BSTNode<Data>* left;
    BSTNode<Data>* right;
    BSTNode<Data>* parent;
    Data const data;

    BSTNode( const Data & d ) :
        data(d) {
        left = right = parent = nullptr ;
    }

};
```

Write a line of code to create a new BSTNode object with int data on the heap and assign nodePtr to point to it.

```
template<typename Data>
class BST {

private:
    BSTNode<Data>* root;    //Pointer to the root of this BS

public:
    /** Default constructor. Initialize an empty BST. */
    BST() : root(nullptr){    }

    void insertAsLeftChild(BSTNode<Data>* parent, const Data& item){
        // Your code here

    }
```

# Working with a BST: Insert

```
//Assume this is inside the definition of the class
void insertAsLeftChild(BSTNode<Data>* parent, const Data& item)
{
    // Your code here
}
```

Which line of code correctly inserts the data item into the BST as the left child of the parent parameter.

- A. `parent.left = item;`
- B. `parent->left = item;`
- C. `parent->left = BSTNode(item) ;`
- D. `parent->left = new BSTNode<Data>(item) ;`
- E. `parent->left = new Data(item) ;`

# Working with a BST: Insert

```
void insertAsLeftChild(BSTNode<Data>* parent, const Data& item) {  
    parent->left = new BSTNode<Data>(item);  
  
}
```

Is this function complete? (i.e. does it do everything it needs to correctly insert the node?)

- A. Yes. The function correctly inserts the data
- B. No. There is something missing.

# What is difference between templates and typedefs?

```
template <class Item>
Item maximum(Item a, Item b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

```
typedef int item;
item maximum(item a, item b)
{
    if (a > b)
        return a;
    else
        return b;
}
```



# Template classes: Non-member functions

```
BST operator+(const BST& b1, const BST&b2);
```

```
template <class T>  
BST<T> operator+(const BST<T>& b1, const BST<T>&b2);
```

# Template classes: Member function definition

For the compiler a name used in a template declaration or definition and that is dependent on a template-parameter is assumed not to name a type *unless* its preceded by a typename

```
template<class T>
class BST{
    //Other code
    Node* getNodeFor(T value, Node* n) const;
};
```

# Template classes: Including the implementation

```
//In bst.h  
class BST{  
//code  
};
```

```
#include "bst.cpp"
```

## How to Convert a Container Class to a Template

1. The template prefix precedes each function prototype or implementation.
2. Outside the class definition, place the word `<Item>` with the class name, such as `bag<Item>`.
3. Use the name `Item` instead of `value_type`.
4. Outside of member functions and the class definition itself, add the keyword *typename* before any use of one of the class's type names. For example:

```
typename bag<Item>::size_type
```

5. The implementation file name now ends with `.template` (instead of `.cxx`), and it is included in the header by an include directive.
6. Eliminate any using directives in the implementation file. Therefore, we must then write `std::` in front of any Standard Library function such as `std::copy`.
7. Some compilers require any default argument to be in both the prototype and the function implementation.

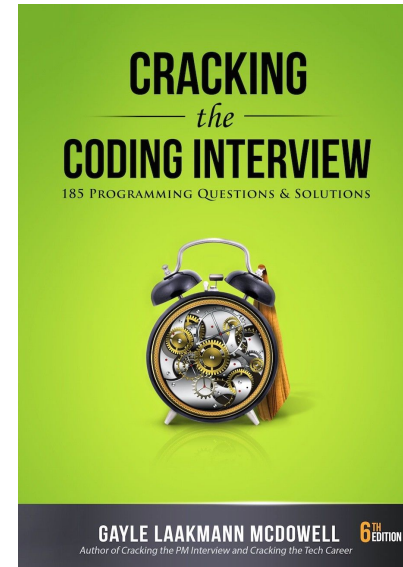
# Have you given a technical interview before?

A. Yes

B. No

# Tips for Technical Interviews

1. Listen carefully
2. Draw an example
3. State the brute force or a partially correct solution
  - then work to get at a better solution
4. Optimize:
  - Make time-space tradeoffs to optimize runtime
  - Precompute information: Reorganize the data e.g. by sorting
5. Solidify your understanding of your algo before diving into writing code.
6. Start coding!



## Small group exercise

Write a ADT called minStack of numbers that provides the following

- `push()` // inserts an element to the “top” of the minStack
- `pop()` // removes the last element that was pushed on the stack
- `top ()` // returns the last element that was pushed on the stack
- `min()` // returns the minimum value of the elements stored so far

