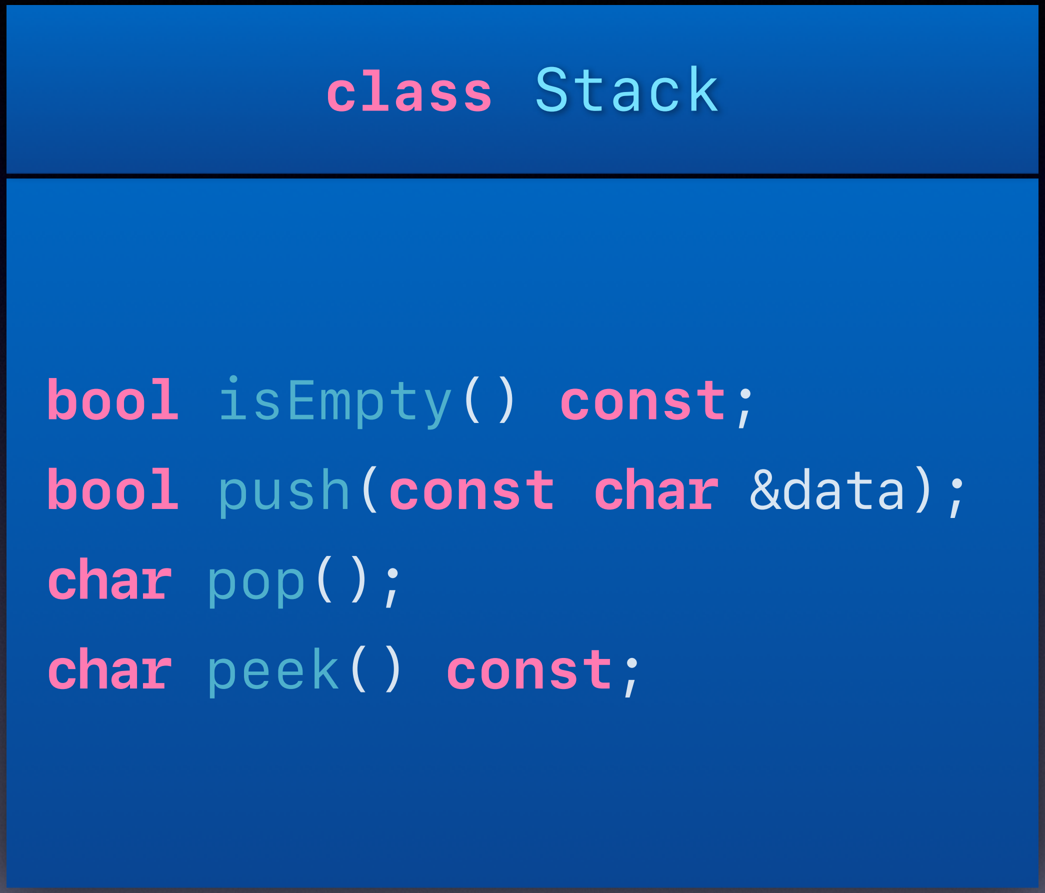
What we wrote last week:

```
bool isEmpty(Stack* stack);
bool push(Stack* stack, char data);
char pop(Stack* stack);
char peek(Stack* stack);
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International



```
class Stack
```

```
bool isEmpty() const;
bool push(const char &data);
char pop();
char peek() const;
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

```
Stack pringlesCan;
```

Alan speak: “method”

Using member function

```
bool Stack::isEmpty() const;


pringlesCan.isEmpty();
```

“function”

Using non-member function

```
bool isEmpty(const Stack &stack);


isEmpty(pringlesCan);
```

A non-member function appears outside of a class

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

```
c3 = add(c1, c2);
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

```
c3 = add(c1, c2);
```

Prefix notation

***pre**ceding the operands*

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

```
c3 = add(c1, c2);
```

Prefix notation
***pre**ceding the operands*

```
c3 = c1.add(c2);
c3 = c1 + c2;
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

```
c3 = add(c1, c2);
```

Prefix notation

***pre**ceding the operands*

```
c3 = c1.add(c2);
c3 = c1 + c2;
```

Infix notation

***in** between the operands*

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

Different languages will have different “styles”

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

Different languages will have different “styles”

C

```
strcmp(stringA, stringB) == 0
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

Different languages will have different “styles”

C

```
strcmp(stringA, stringB) == 0
```

Java

```
stringA.isEqual(stringB)
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

Different languages will have different “styles”

C

```
strcmp(stringA, stringB) == 0
```

Java

```
stringA.isEqual(stringB)
```

Swift / C++ (when using std::string)

```
stringA == stringB
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

Different languages will have different “styles”

C

```
strcmp(stringA, stringB) == 0
```

Java

```
stringA.isEqual(stringB)
```

Swift / C++ (when using std::string)

```
stringA == stringB
```

Javascript

```
stringA === stringB
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

Different languages will have different “styles”

C

```
strcmp(stringA, stringB) == 0
```

Java

```
stringA.isEqual(stringB)
```

Swift / C++ (when using std::string)

```
stringA == stringB
```

Javascript

```
stringA === stringB
```

C#

```
stringA == stringB
stringA.Equals(stringB)
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

What to do in destructor

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

What to do in destructor

For most of 122, you'll be using the destructor to deallocate stuff from the heap (malloc'd).

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

What to do in destructor

For most of 122, you'll be using the destructor to deallocate stuff from the heap (malloc'd).

```
class List {
private:

    int _capacity;

    int* _array;









};
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

What to do in destructor

For most of 122, you'll be using the destructor to deallocate stuff from the heap (malloc'd).

```
class List {
private:

    int _capacity;

    int* _array;

public:
    List(int capacity): _capacity(capacity) {
        _array = (int*)malloc(sizeof(int) * capacity);
    }

    List(const List &copy) { ... }

    ~List() {
        delete[] _array;
    }

};
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

What to do in destructor

For most of 122, you'll be using the destructor to deallocate stuff from the heap (malloc'd).

```
class List {
private:
    int _capacity;     on stack, no action needed
    int* _array;
public:
    List(int capacity): _capacity(capacity) {
        _array = (int*)malloc(sizeof(int) * capacity);
    }

    List(const List &copy) { ... }

    ~List() {
        delete[] _array;
    }

};
```

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

What to do in destructor

For most of 122, you'll be using the destructor to deallocate stuff from the heap (malloc'd).

```
class List {
private:
    int _capacity;
    int* _array;
public:
    List(int capacity): _capacity(capacity) {
        _array = (int*)malloc(sizeof(int) * capacity);
    }

    List(const List &copy) { ... }

    ~List() {
        delete[] _array;
    }

};
```

on stack, no action needed

on heap, need to deallocate manually

Alan Chu (ualch9@gmail.com) https://github.com/ualch9/cpts122-lab-presentations Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International

What to do in destructor

For most of 122, you'll be using the destructor to deallocate stuff from the heap (malloc'd).

```
class List {
private:
    int _capacity;
    int* _array;
public:
    List(int capacity): _capacity(capacity) {
        _array = (int*)malloc(sizeof(int) * capacity);
    }

    List(const List &copy) { ... }

    ~List() {
        delete[] _array;
    }

};
```

on stack, no action needed

on heap, need to deallocate manually