Data structures & algorithms

Tutorial 6

Lesson overview

- Recap of important topics from last week
- Two sum
- Implement a Stack with a Vector
- Stack Calculator

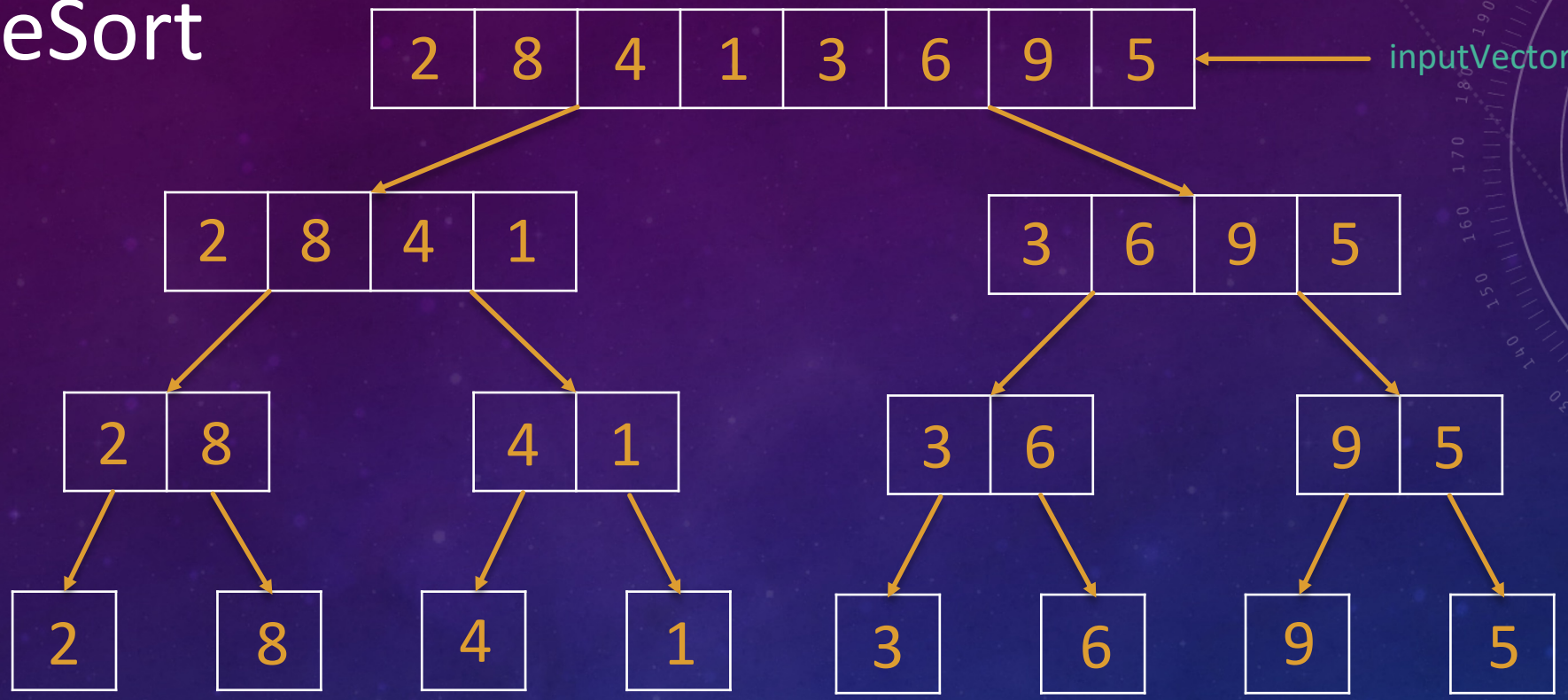
A quick recap of important things from last week

Egyptian multiplication

- In this exercise, we're going to work on two recursive functions, `multiply` and `power`
- If you recall with our factorial problem, the factorial function was defined as… this definition gave us the base case and a way to write the function recursively. As you can see below
- In this exercise, the definition is also given to us. This way is more than fine, but it is rather slow, there is a way to do this recursively and have it run in O(log y) time
- The faster approach would be to recursively call the function and half `y` on each call. Once the base case is reached it should exit the call and check if our `y` is odd or even. If it's odd we add `x` onto our variable and if its even we don't. After that operation is done, we return our variable
- Note: we want our recursive call to be on a variable and not in the return statement

```
int multiply(int x, int y) {
    if(y == 0) {
        return 0;
    }
    int variable =
multiply(???)
    if(odd) {
        // do something to
variable
    }
    return variable;
}
```

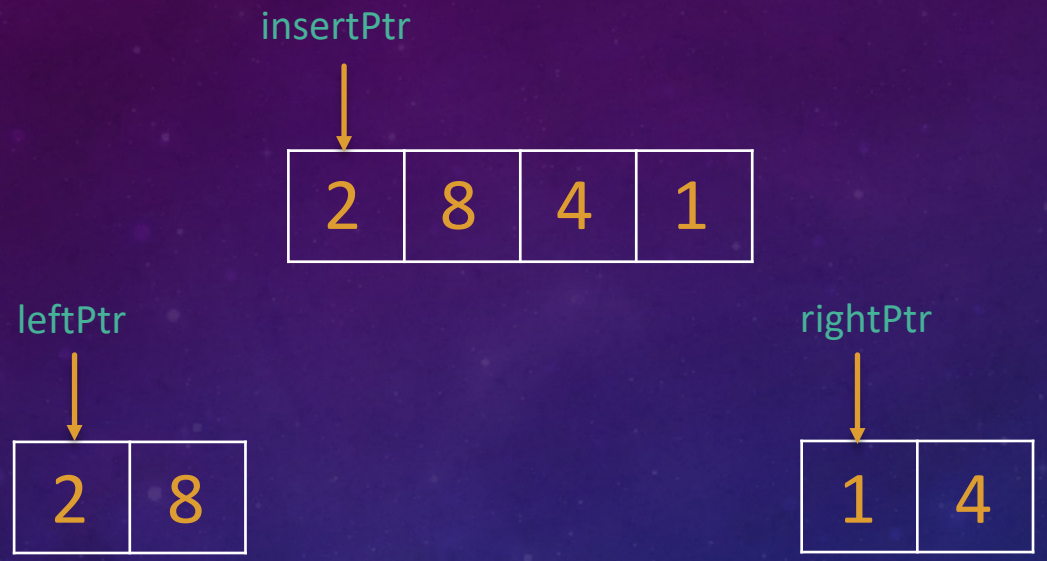
MergeSort



- MergeSort is a sorting algorithm that sorts an array in O(n * log n) time. It uses the divide-and-conquer paradigm in which it divides the array you pass to the function, into two sub-arrays
- The entry point to the algorithm is `mergesort(Iter low, Iter high)` which takes an array as parameter, or in our case the iterator to the beginning and end elements
- This `mergesort` function computes the midpoint of that array, and recursively calls on each `mergesort` sub-array until each one is of size 1

MergeSort

Is 2 smaller than 1?
Insert 1 and increment rightPtr and insertPtr



- Once the sub-arrays reach that size, we can call the `merge` function with three arguments, the beginning of the first sub-array, the end of the second sub-array, and the middle that separates the two.
- The function will merge each sub-array into an auxiliary array in sorted order. It does so with a pointer on each sub-array, and comparing the values to determine which element to insert and which pointer to increment

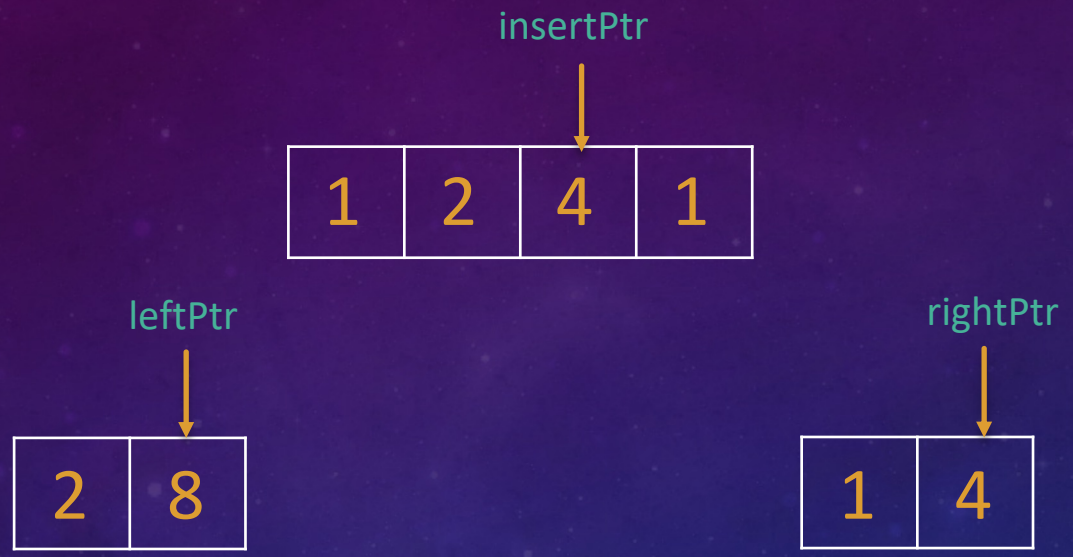
MergeSort

Is 2 smaller than 4?
Insert 2 and increment leftPtr and insertPtr

- Once the sub-arrays reach that size, we can call the `merge` function with three arguments, the beginning of the first sub-array, the end of the second sub-array, and the middle that separates the two.
- The function will merge each sub-array into an auxiliary array in sorted order. It does so with a pointer on each sub-array, and comparing the values to determine which element to insert and which pointer to increment

MergeSort

Is 8 smaller than 4?
Insert 4 and increment rightPtr and insertPtr



- Once the sub-arrays reach that size, we can call the `merge` function with three arguments, the beginning of the first sub-array, the end of the second sub-array, and the middle that separates the two.
- The function will merge each sub-array into an auxiliary array in sorted order. It does so with a pointer on each sub-array, and comparing the values to determine which element to insert and which pointer to increment

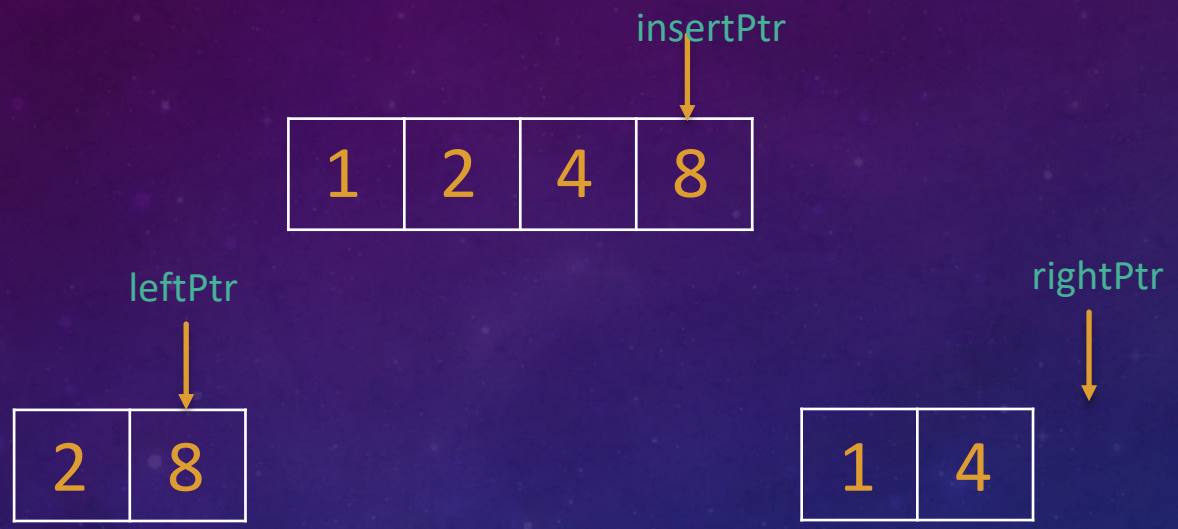
MergeSort

Since the right sub-array has been exhausted. We can insert everything left on the left sub-array

- Once the sub-arrays reach that size, we can call the `merge` function with three arguments, the beginning of the first sub-array, the end of the second sub-array, and the middle that separates the two.
- The function will merge each sub-array into an auxiliary array in sorted order. It does so with a pointer on each sub-array, and comparing the values to determine which element to insert and which pointer to increment

MergeSort

Since the right sub-array has been exhausted. We can insert everything left on the left sub-array



- Once the sub-arrays reach that size, we can call the `merge` function with three arguments, the beginning of the first sub-array, the end of the second sub-array, and the middle that separates the two.
- The function will merge each sub-array into an auxiliary array in sorted order. It does so with a pointer on each sub-array, and comparing the values to determine which element to insert and which pointer to increment

Lab time!

Two Sum

- Two Sum problem is one of the most popular easy Leetcode questions

Two Sum

- Two Sum problem is one of the most popular easy Leetcode questions
- In the problem, you are given an un-ordered vector and a target.

Two Sum

```
target = 4
```

- Two Sum problem is one of the most popular easy Leetcode questions
- In the problem, you are given an un-ordered vector and a target. The goal is to find two numbers in the vector that when added together equal the target.

Two Sum

`target = 4` `i != j`

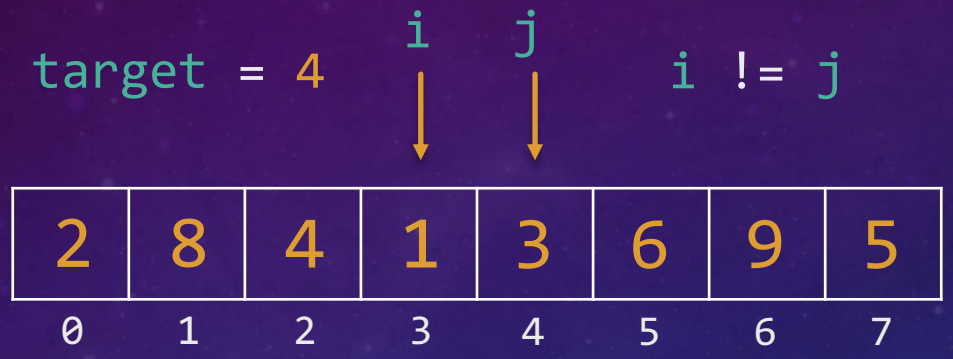
- Two Sum problem is one of the most popular easy Leetcode questions
- In the problem, you are given an un-ordered vector and a target. The goal is to find two numbers in the vector that when added together equal the target. These two numbers must also be in differing indexes within the array

Two Sum

`target = 4` `i != j`

- Two Sum problem is one of the most popular easy Leetcode questions
- In the problem, you are given an un-ordered vector and a target. The goal is to find two numbers in the vector that when added together equal the target. These two numbers must also be in differing indexes within the array
- Once two numbers have been found that add up to the target, output the pair of indices

Two Sum



- Two Sum problem is one of the most popular easy Leetcode questions
- In the problem, you are given an un-ordered vector and a target. The goal is to find two numbers in the vector that when added together equal the target. These two numbers must also be in differing indexes within the array
- Once two numbers have been found that add up to the target, output the pair of indices
- In the example above, the pair of indices {3, 4} add up to our target, so we return that

Two Sum

- There are a couple of ways to find the solution to this problem, you could try:

Two Sum

```
for(int i = 0; i < vec.size(); ++i){
        for(int j = 0; j < vec.size(); ++j){

        }
}
```

- There are a couple of ways to find the solution to this problem, you could try:
  - Brute force with a nested for-loop with a time complexity of $O(n^2)$

Two Sum

```
for(int i = 0; i < vec.size(); ++i){
        for(int j = 0; j < vec.size(); ++j){

        }
}
```

```
std::unordered_map<int, std::unordered_map>
```

- There are a couple of ways to find the solution to this problem, you could try:
  - Brute force with a nested for-loop with a time complexity of $O(n^2)$
  - Using a std::unordered_map and do it within a single loop, with a time complexity of O(n) and a space complexity of O(n)

Give “Two Sum” and try and implement a solution to the problem!

Implement a Stack with a Vector

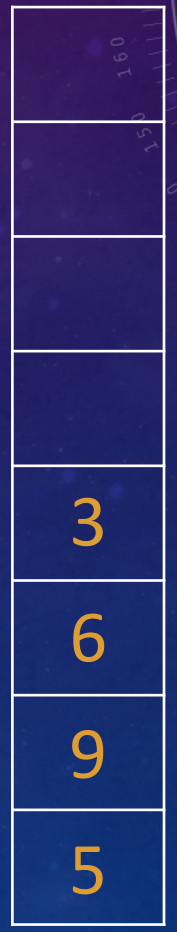
- A stack is essentially a vector with limited operations, it follows the last in, first out (LIFO) principle. Meaning that the last element put on the stack, will be the first element to be removed from it

Implement a Stack with a Vector

- A stack is essentially a vector with limited operations, it follows the last in, first out (LIFO) principle. Meaning that the last element put on the stack, will be the first element to be removed from it
- You can think the idea of a stack like a stack of pancakes or plates, where to add a pancake/plate it would be to the top of the stack… and to remove one it would also be at the top

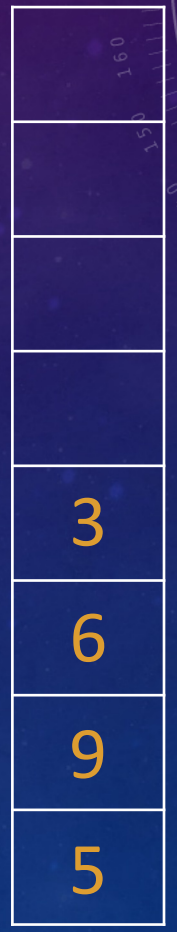
Implement a Stack with a Vector

- A stack is essentially a vector with limited operations, it follows the last in, first out (LIFO) principle. Meaning that the last element put on the stack, will be the first element to be removed from it
- You can think the idea of a stack like a stack of pancakes or plates, where to add a pancake/plate it would be to the top of the stack… and to remove one it would also be at the top
- So it's as if someone took a vector and flipped it to stand up vertically



Implement a Stack with a Vector

- A stack is essentially a vector with limited operations, it follows the last in, first out (LIFO) principle. Meaning that the last element put on the stack, will be the first element to be removed from it
- You can think the idea of a stack like a stack of pancakes or plates, where to add a pancake/plate it would be to the top of the stack… and to remove one it would also be at the top
- So it's as if someone took a vector and flipped it to stand up vertically
- For example, pushing the elements 4, 7 and 1 it would look like this…



Implement a Stack with a Vector

- A stack is essentially a vector with limited operations, it follows the last in, first out (LIFO) principle. Meaning that the last element put on the stack, will be the first element to be removed from it
- You can think the idea of a stack like a stack of pancakes or plates, where to add a pancake/plate it would be to the top of the stack… and to remove one it would also be at the top
- So it's as if someone took a vector and flipped it to stand up vertically
- For example, pushing the elements 4, 7 and 1 it would look like this…

Implement a Stack with a Vector

- A stack is essentially a vector with limited operations, it follows the last in, first out (LIFO) principle. Meaning that the last element put on the stack, will be the first element to be removed from it
- You can think the idea of a stack like a stack of pancakes or plates, where to add a pancake/plate it would be to the top of the stack… and to remove one it would also be at the top
- So it's as if someone took a vector and flipped it to stand up vertically
- For example, pushing the elements 4, 7 and 1 it would look like this…

Implement a Stack with a Vector

- A stack is essentially a vector with limited operations, it follows the last in, first out (LIFO) principle. Meaning that the last element put on the stack, will be the first element to be removed from it
- You can think the idea of a stack like a stack of pancakes or plates, where to add a pancake/plate it would be to the top of the stack… and to remove one it would also be at the top
- So it's as if someone took a vector and flipped it to stand up vertically
- For example, pushing the elements 4, 7 and 1 it would look like this… see how elements are pushed to the stop of the stack.

Implement a Stack with a Vector

- A stack is essentially a vector with limited operations, it follows the last in, first out (LIFO) principle. Meaning that the last element put on the stack, will be the first element to be removed from it
- You can think the idea of a stack like a stack of pancakes or plates, where to add a pancake/plate it would be to the top of the stack… and to remove one it would also be at the top
- So it's as if someone took a vector and flipped it to stand up vertically
- For example, pushing the elements 4, 7 and 1 it would look like this… see how elements are pushed to the stop of the stack. Whilst to remove two elements from the stack it would look like this

Implement a Stack with a Vector

- A stack is essentially a vector with limited operations, it follows the last in, first out (LIFO) principle. Meaning that the last element put on the stack, will be the first element to be removed from it
- You can think the idea of a stack like a stack of pancakes or plates, where to add a pancake/plate it would be to the top of the stack… and to remove one it would also be at the top
- So it's as if someone took a vector and flipped it to stand up vertically
- For example, pushing the elements 4, 7 and 1 it would look like this… see how elements are pushed to the stop of the stack. Whilst to remove two elements from the stack it would look like this

Implement a Stack with a Vector

- A stack is essentially a vector with limited operations, it follows the last in, first out (LIFO) principle. Meaning that the last element put on the stack, will be the first element to be removed from it
- You can think the idea of a stack like a stack of pancakes or plates, where to add a pancake/plate it would be to the top of the stack… and to remove one it would also be at the top
- So it's as if someone took a vector and flipped it to stand up vertically
- For example, pushing the elements 4, 7 and 1 it would look like this… see how elements are pushed to the stop of the stack. Whilst to remove two elements from the stack it would look like this

std::stack

Main operations for `std::stack`

```
std::stack<int> myStack{4, 5, 9, 10}; // Initialize with values
```

- `myStack.top()`: accesses element at the top of the stack
  - `myStack.top() // returns 10`
- `myStack.push(value)`: pushes `value` to the top of the stack
  - `myStack.push(13) // stack now contains {4, 5, 9, 10, 13}`
- `myStack.pop()`: removes element at the top of the stack
  - `myStack.pop() // vector now contains {4, 5, 9}`
- `myStack.size()`: returns the size of the stack
  - `myStack.size() // returns 4`
- `myStack.empty()`: empties the stack
  - `myStack.empty() // stack now contains {} (0 elements)`

- As you can see, it's like someone removed half of the operations from a vector.
- The goal of the exercise is to implement these operations of a stack using a vector

Try to “Implement a Stack with a Vector” and try and implement a solution to the problem!

Stack Calculator

- The goal for this exercise is to implement an algorithm to evaluate postfix notation… but what is post fix notation?

Stack Calculator

4 + 8 * 3

- The goal for this exercise is to implement an algorithm to evaluate postfix notation… but what is post fix notation?
- Well given this expression, it would be fairly difficult for a computer to calculate…

Stack Calculator

"4 + 8 * 3"

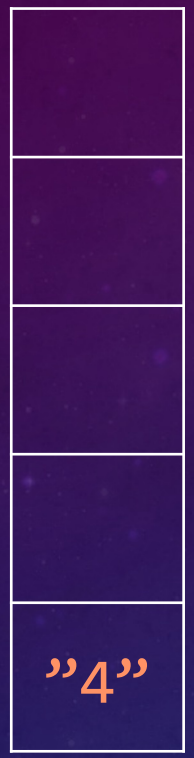
- The goal for this exercise is to implement an algorithm to evaluate postfix notation… but what is post fix notation?
- Well given this expression, it would be fairly difficult for a computer to calculate… especially if it was a string.

Stack Calculator

“4 + 8 * 3” “4 8 3 * +”

- The goal for this exercise is to implement an algorithm to evaluate postfix notation… but what is post fix notation?
- Well given this expression, it would be fairly difficult for a computer to calculate… especially if it was a string. The post fix notation would look like this.

Stack Calculator



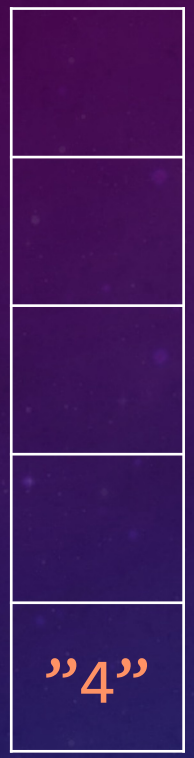
"4 + 8 * 3"

"4 8 3 * +"

- The goal for this exercise is to implement an algorithm to evaluate postfix notation… but what is post fix notation?
- Well given this expression, it would be fairly difficult for a computer to calculate… especially if it was a string. The post fix notation would look like this. The algorithm to create an expression in postfix follows a very simple idea. Basically you loop through the input and push all numbers to an output stack.

Stack Calculator

outputStack



"4 + 8 * 3"

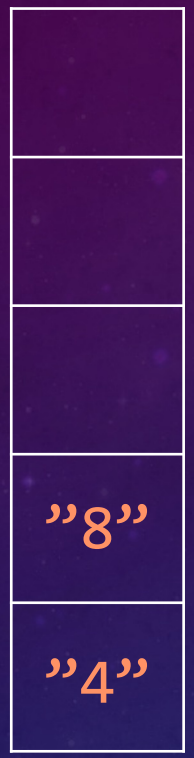
"4 8 3 * +"

operatorStack

- The goal for this exercise is to implement an algorithm to evaluate postfix notation… but what is post fix notation?
- Well given this expression, it would be fairly difficult for a computer to calculate… especially if it was a string. The post fix notation would look like this. The algorithm to create an expression in postfix follows a very simple idea. Basically you loop through the input and push all numbers to an output stack. You also have an operator stack, that will hold all operators until they are ready to be pushed onto the output based off its precedence

Stack Calculator

outputStack



"4 + 8 * 3"

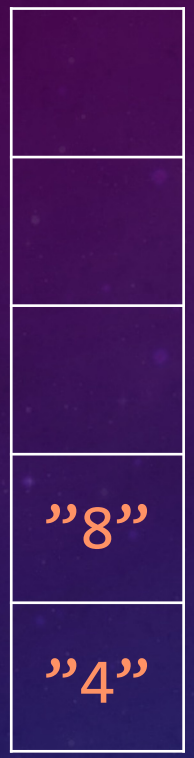
"4 8 3 * +"

operatorStack

- The goal for this exercise is to implement an algorithm to evaluate postfix notation… but what is post fix notation?
- Well given this expression, it would be fairly difficult for a computer to calculate… especially if it was a string. The post fix notation would look like this. The algorithm to create an expression in postfix follows a very simple idea. Basically you loop through the input and push all numbers to an output stack. You also have an operator stack, that will hold all operators until they are ready to be pushed onto the output based off its precedence

Stack Calculator

outputStack



"4 + 8 * 3"

"4 8 3 * +"

operatorStack

- The goal for this exercise is to implement an algorithm to evaluate postfix notation… but what is post fix notation?
- Well given this expression, it would be fairly difficult for a computer to calculate… especially if it was a string. The post fix notation would look like this. The algorithm to create an expression in postfix follows a very simple idea. Basically you loop through the input and push all numbers to an output stack. You also have an operator stack, that will hold all operators until they are ready to be pushed onto the output based off its precedence

Stack Calculator

outputStack

"4 + 8 * 3"

"4 8 3 * +"

operatorStack

- The goal for this exercise is to implement an algorithm to evaluate postfix notation… but what is post fix notation?
- Well given this expression, it would be fairly difficult for a computer to calculate… especially if it was a string. The post fix notation would look like this. The algorithm to create an expression in postfix follows a very simple idea. Basically you loop through the input and push all numbers to an output stack. You also have an operator stack, that will hold all operators until they are ready to be pushed onto the output based off its precedence. Once the input has been looped through, we can pop everything from the operator stack to the output stack

Stack Calculator

outputStack

"4 + 8 * 3"

"4 8 3 * +"

operatorStack

- The goal for this exercise is to implement an algorithm to evaluate postfix notation… but what is post fix notation?
- Well given this expression, it would be fairly difficult for a computer to calculate… especially if it was a string. The post fix notation would look like this. The algorithm to create an expression in postfix follows a very simple idea. Basically you loop through the input and push all numbers to an output stack. You also have an operator stack, that will hold all operators until they are ready to be pushed onto the output based off its precedence. Once the input has been looped through, we can pop everything from the operator stack to the output stack

Stack Calculator

outputStack

"4 + 8 * 3"

"4 8 3 * +"

operatorStack

- The goal for this exercise is to implement an algorithm to evaluate postfix notation… but what is post fix notation?
- Well given this expression, it would be fairly difficult for a computer to calculate… especially if it was a string. The post fix notation would look like this. The algorithm to create an expression in postfix follows a very simple idea. Basically you loop through the input and push all numbers to an output stack. You also have an operator stack, that will hold all operators until they are ready to be pushed onto the output based off its precedence. Once the input has been looped through, we can pop everything from the operator stack to the output stack

Stack Calculator

{‘4’, ‘8’, ‘3’, ‘*’, ‘+’}

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation

Stack Calculator

{‘4’, ‘8’, ‘3’, ‘*’, ‘+’}

sumStack

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation
- You will need to initialize a stack that contains ints (you can honestly just use a vector), this stack will store the results produced throughout the algorithm, and the result of the expression

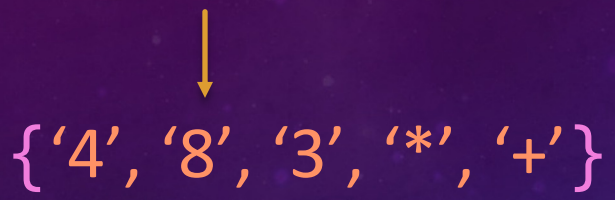
Stack Calculator

{‘4’, ‘8’, ‘3’, ‘*’, ‘+’}

sumStack

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation
- You will need to initialize a stack that contains ints (you can honestly just use a vector), this stack will store the results produced throughout the algorithm, and the result of the expression
- You will need to loop through the tokens and check whether it is a number, or not. If it is you will have to push it to the top of the stack (don’t forget to cast it to an int).

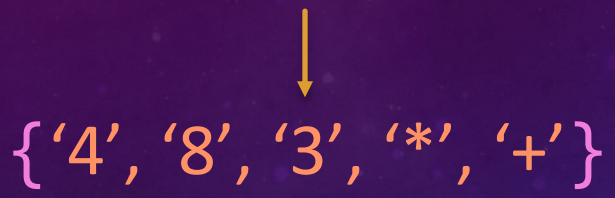
Stack Calculator



sumStack

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation
- You will need to initialize a stack that contains ints (you can honestly just use a vector), this stack will store the results produced throughout the algorithm, and the result of the expression
- You will need to loop through the tokens and check whether it is a number, or not. If it is you will have to push it to the top of the stack (don’t forget to cast it to an int).

Stack Calculator



sumStack

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation
- You will need to initialize a stack that contains ints (you can honestly just use a vector), this stack will store the results produced throughout the algorithm, and the result of the expression
- You will need to loop through the tokens and check whether it is a number, or not. If it is you will have to push it to the top of the stack (don’t forget to cast it to an int).

Stack Calculator


```
int value1 = sumStack.back();
int value2 = *(sumStack.end()-2);
```

sumStack

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation
- You will need to initialize a stack that contains ints (you can honestly just use a vector), this stack will store the results produced throughout the algorithm, and the result of the expression
- You will need to loop through the tokens and check whether it is a number, or not. If it is you will have to push it to the top of the stack (don’t forget to cast it to an int). If it is an operator rather than a number, you take the two last numbers

Stack Calculator


```
int value1 = sumStack.back();
int value2 = *(sumStack.end()-2);
int result = value1 * value2;
```

sumStack

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation
- You will need to initialize a stack that contains ints (you can honestly just use a vector), this stack will store the results produced throughout the algorithm, and the result of the expression
- You will need to loop through the tokens and check whether it is a number, or not. If it is you will have to push it to the top of the stack (don’t forget to cast it to an int). If it is an operator rather than a number, you take the two last numbers, and perform the necessary operation.

Stack Calculator


```
int value1 = sumStack.back();
int value2 = *(sumStack.end()-2);
int result = value1 * value2;
```

sumStack

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation
- You will need to initialize a stack that contains ints (you can honestly just use a vector), this stack will store the results produced throughout the algorithm, and the result of the expression
- You will need to loop through the tokens and check whether it is a number, or not. If it is you will have to push it to the top of the stack (don’t forget to cast it to an int). If it is an operator rather than a number, you take the two last numbers, and perform the necessary operation. Remove the last two elements from the sumStack…

Stack Calculator


```
int value1 = sumStack.back();
int value2 = *(sumStack.end()-2);
int result = value1 * value2;
```

sumStack

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation
- You will need to initialize a stack that contains ints (you can honestly just use a vector), this stack will store the results produced throughout the algorithm, and the result of the expression
- You will need to loop through the tokens and check whether it is a number, or not. If it is you will have to push it to the top of the stack (don’t forget to cast it to an int). If it is an operator rather than a number, you take the two last numbers, and perform the necessary operation. Remove the last two elements from the sumStack… and push the result on

Stack Calculator


```
int value1 = sumStack.back();
int value2 = *(sumStack.end()-2);
int result = value1 * value2;
```

sumStack

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation
- You will need to initialize a stack that contains ints (you can honestly just use a vector), this stack will store the results produced throughout the algorithm, and the result of the expression
- You will need to loop through the tokens and check whether it is a number, or not. If it is you will have to push it to the top of the stack (don’t forget to cast it to an int). If it is an operator rather than a number, you take the two last numbers, and perform the necessary operation. Remove the last two elements from the sumStack… and push the result on. Then you just repeat until you have looped through the whole expression…

Stack Calculator


```
int value1 = sumStack.back();
int value2 = *(sumStack.end()-2);
int result = value1 + value2;
```

sumStack

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation
- You will need to initialize a stack that contains ints (you can honestly just use a vector), this stack will store the results produced throughout the algorithm, and the result of the expression
- You will need to loop through the tokens and check whether it is a number, or not. If it is you will have to push it to the top of the stack (don’t forget to cast it to an int). If it is an operator rather than a number, you take the two last numbers, and perform the necessary operation. Remove the last two elements from the sumStack… and push the result on. Then you just repeat until you have looped through the whole expression…

Stack Calculator


```
int value1 = sumStack.back();
int value2 = *(sumStack.end()-2);
int result = value1 + value2;
```

sumStack

- In the evalRPN function, it is passed a vector containing list of tokens. This vector is our expression in postfix notation
- You will need to initialize a stack that contains ints (you can honestly just use a vector), this stack will store the results produced throughout the algorithm, and the result of the expression
- You will need to loop through the tokens and check whether it is a number, or not. If it is you will have to push it to the top of the stack (don’t forget to cast it to an int). If it is an operator rather than a number, you take the two last numbers, and perform the necessary operation. Remove the last two elements from the sumStack… and push the result on. Then you just repeat until you have looped through the whole expression… the final result is now at the top of the stack

Give “Stack Calculator” a go to try and implement your solution!

Access to google drive

- I will upload slides to the Google Drive after every class
- https://drive.google.com/drive/folders/1H5psebndM_YVyoJE-BJ_ODNJOfgq9-uI

Contact: Thomas.golding@uts.edu.au