A calculator

Kevin Wang

Spring Fall August 2022

Introduction

In this task i will describe how a stack works and can be used to calculate mathematical expressions described using reverse Polish notation. What is a stack A stack is a data structure where we can store and use the data. there are two operations to it we name it pop and push. We can simply imagine how this data structure works by sketching and calculate mathematical expressions in reverse Polish notation. Reverse polish notation means that instead of doing operations like this normal way we instead put the operand in the last like this 45+. The first thing i will do in this assignment is to try these stacks and calculating with reverse polish notation in Java. And also create class of objects that i will use for stack exercises. The second part of this assignment i will implement two different types of stack, a static stack and a dynamic stack and discuss these two. I will then do some benchmark of these stacks to see what the difference are in respect to time.

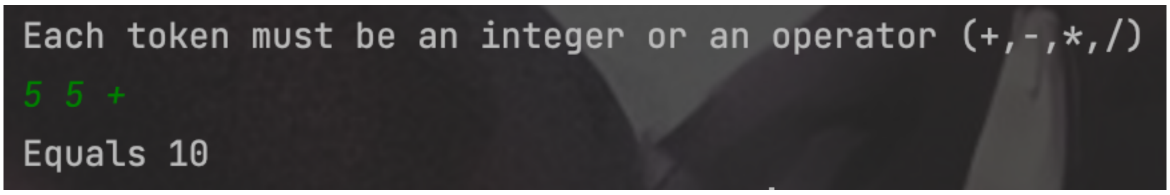
1 An expression

In this section i am suppose to create a class Item, inside this class there is an object that is a array consisting of items where each item has a type and a value. For this Item there is a ItemType class and what it does is that it determines if the item is an operation or a value.

2 The calculator

The calculator is pretty straightforward where i just implement simple operations. And then some simple pop and push logic.

In "step" method i implemented several operands where i followed the template code ADD and i just added sub(-), division(/), multiplication(*) and modulo division in the method. Just stack.pop in the variables and then stack.push the operation. This method is not returning any value. Next step is to implement stacks, where i can do my operations and store my values.



```
Each token must be an integer or an operator (+,-,*,/)
5 5 +
Equals 10
```

Figure 1: An image of the calculator

3 A static stack

A static stack means that its a fixed sized of storage for example my stack would be of size four. This stack is going to allocate an array of four items and keep track of a stack pointer (an index) which also are the push and pop method. And this stack would only hold integers. Push and pop will be implemented.

I can implement a static stack by using array. I create a array and set the max value to 5 for example.

$$classStack$$

static final int MAX = 5;
int top; //pointer
int a[] = new int[MAX];

the pointer point to the top of stack and not not above it. If the stack has no element is known as an empty stack. When the stack is empty the value of the top variable is -1.

A stack overflow is a logical run-time error and not a syntax error. Overflow should not be possible in a fixed stack. So what i can do is to know how large my stack storage is and try to not extend that in a static stack. And i can keep track of the stack by checking where the pointer points at. For example when pointer has value 0 it means that the stack only has one element.

When its overflow the computer either only calculate the first digits that the stack have space for, for example if i have 3 4 + 5 6 + ... but the program stop working if i have 40000 40000 +

There is a precondition of calling pop and that is the stack is non-empty otherwise the program will have a undefined outcome.

4 A dynamic stack

Since static stack means that the storage is in fixed then dynamic means that it grows as you add more items to it.

A stack contains a top pointer, which is “head” of the stack where pushing and popping items, as we know from the top. The first node will have ”null” in the stack storage and second node link have first node address in this stack and last node address in “top” pointer. When The stack is full the stack increase its size by double then elements are copied from previous to new storage.

Here is how i implemented a dynamic stack using a growing array. Instead of throwing a StackOverFlowError as we did for static stack we call a method called ExtendStack. This method is to double the size of the array, making room for the new elements that are about to be pushed.

```
    private static void extendStack() {
ExSize = ExSize * 2;
int[] newArray = new int[ExSize];

    for (int i = 0; i ¡ stack.length; i++) {
newArray[i] = stack[i];
}
stack = newArray;
}
```

The maximum array size will increase by double, whilst copying the old content into the new array. In terms of shrinking the stack, when pop is called, the method checks if the stack pointer is equal to or less than one forth of the current size, if so then it will call method reduceStack, that will shrink half of the size of the array. we do this in order to reduce memory waste from to having a unnecessary spaces in the stack.

```
    private static void reduceStack() {

    ExSize = ExSize / 2;
int[] tempArray = new int[ExSize];

    for (int i = 0; i ¡ tempArray.length; i++) {
tempArray[i] = stack[i];
}
stack = tempArray;

    }
```

5 calculate your last digit

My first 9 digits in my personal number is 199908110. Multiplying every digit with 2.1.2.1... and so on. Then taking sum of that to do modulo divison of modulo 10 (mod10). Then subtract the result with 10. I get:

$$10-((y12+y2+m12+m2...)mod10) = 10-((12+91....)mod10) = 10-(49mod10)$$

$$10-(49mod10) = 10-9 = 1$$

The result is thus 1. I did my calculation first y hand and then i took my result of 9 first digit and put it in static stack program and it worked. But if i had to do all my calculation from beginning then i should do it in a dynamic stack program.

So how does this expression looks like in reverse polish notation? it can be hard to know how to convert it in the beginning but once you doing it with stacks it would be easy. In reverse polish notation is going to look like this: (10 49 10 mod10 -).

Operation	Pop and push	Stack
10	10	4
49	49 10	3
10	10 49 10	2
mod10	10 9	1
-	1	0

The last digit of my personal number is 0 but somehow i got 1. Maybe its because of a slightly small calculation error i made. It would be correct if i got 50 mod 10.

6 Benchmark

In static stack the content of the data structure can be changed without changing the size or memory. This makes that the computer allocates the memory before program execution and there will be no memory reuse. The average time complexity for stack is thus O(1).

In dynamic stack the average for time complexity would be O(N). Depending on how large we need to increase our storage for example When we increase the size of array by 1 or double its size, then elements are copied from previous to new array, therefore the average time complexity becomes O(n)

The challenging part here is that since I'm only working with integers variable so my return value is of course also integer. But method System.nanoseconds requires the variable is declared as long.

Operation	Static	Dynamic
1 2 +	2.3μs	4.4μs
10 10 +	5.5μs	5.2μs
100 100 100 100+	Error	6.1μs
1000 1000 2000 + -	Error	7.0μs

For the third and fourth Static stack operations i had a time but the result is just what its capable of calculate.

7 Conclusion

The conclusion that can be drawn is that Stack is a very useful and flexible data structure when using of storing numbers and when doing calculations. And Both static and dynamic stacks have their own pos and cons. For example static is less efficient than dynamic data structure but is faster and better for memory, no overflow occurs and Easy to allocate. While Dynamic is more efficient but takes more memory and the memory is allocated at run-time, it takes more time. And of course more difficult to allocate.