SCC.121: Fundamentals of Computer Science
Stacks

Amit Chopra

amit.chopra@lancaster.ac.uk

What is a Stack?

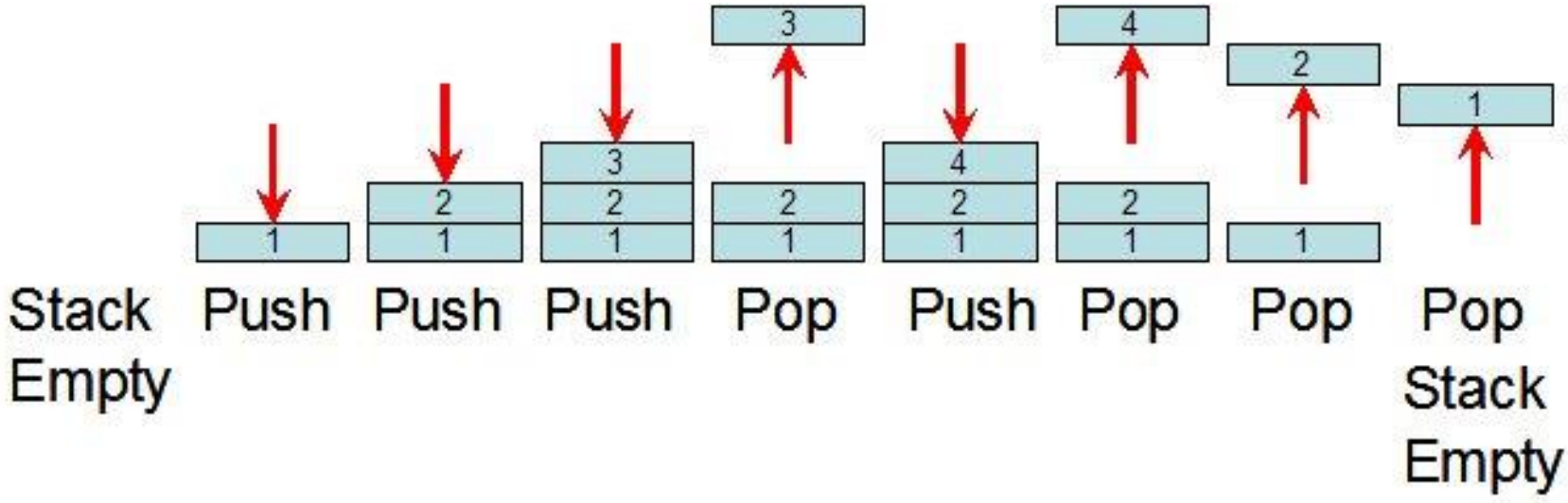
- A stack is an Abstract Data Type, where the collection of items are ordered by when they were added.
- Items can be inserted and removed only at one end (e.g. the top).
  - Last In, First Out (LIFO)
  - First in, Last Out (FILO)

Stack functions

- `push`
  - Place an item on the stack

- `pop`
  - Look at the item on top of the stack and remove it

Pushing and Popping



Thanks to Mark Brandyberry for the graphic

Stacks – what can go wrong?

- What happens if we try to pop an item off the stack when the stack is empty?
  - This is called stack underflow.
  - The pop method needs some way of telling us that this has happened.
- In bounded stacks, what happens if we try to push an item onto the stack when the stack is full?
  - This is called stack overflow.
  - The push method needs some way of telling us that this has happened.

Stack ADT (Assume stack of integers)

- Stack Stack()
- Push(S,int x)
- int Pop(S)
- bool Empty(S)

Unbounded Stack Implementation

```
Element {
    int data;
    Element prev;
}
```

```
Stack(){
    top = nil;
}
```

```
bool Empty(S){
      if(S.top is nil)
            return true;
      return false;
}
```

Unbounded Stack Implementation

```
Push(S,x) {
    el = new Element
    el.data = x
    el.prev = nil
    if(Empty(S))
        S.top = el
    else
        el.prev = S.top
        S.top = el
}
```

```
int Pop(S) {
    if(Empty(S))
        “Underflow”
    else
        tmp = S.top
        S.top = S.top.prev
        return tmp.data
}
```

Bounded Stack Using A[MAX_SIZE]

```
Stack(){
    top = -1;
}
```

```
bool Empty(S){
      if(S.top = -1)
            return true;
      return false;
}
```

```
bool Full(S){
      if(S.top = MAX_SIZE-1)
            return true;
      return false;
}
```

Bounded Stack Using A[MAX_SIZE]

```
Push(S,x) {
    if(Full(S))
        “Overflow!”
    else
      S.top++
      S.A[S.top] = x;
}
```

```
int Pop(S) {
    if(Empty(S))
        “Underflow”
    else
        data = S.A[top]
        S.top--
        return data
}
```

Applications

- Program execution via call stack
- Syntax and semantics of programming languages
  - Evaluating expressions
  - Parentheses (Braces) Matching
- Searching (Depth First)

Function call stacks

```
float stdDev(float values[]) {
    float avg = average(values)
    float total = 0.0
    for (int i = 0; i < values.length; i++)
       total += square_dist(values[i], avg)
    avg = total / values.arrayLength
    return sqrt(avg)
    }
```

```
float average(float values[]) {
    float total
    for (int i = 0; i < values.arrayLength;
i++)
       total += values[i]
    return total / values.arrayLength
    }
```

```
float square_dist(float a, float b) {
    if (a > b) return square(a – b)
       else return square(b – a)
    }
```

```
float square(float a) {
    return a * a
    }
```

Function call stacks

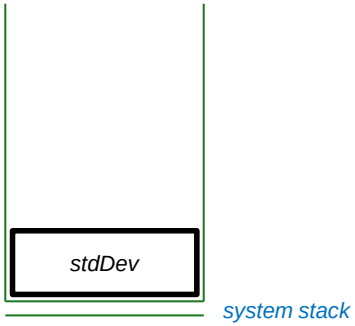
```
float stdDev(float values[]) {
    float avg = average(values)
    float total = 0.0
    for (int i = 0; i < values.length; i++)
       total += square_dist(values[i], avg)
    avg = total / values.arrayLength
    return sqrt(avg)
    }
```

24 bytes of local variables

```
float average(float values[]) {
    float total
    for (int i = 0; i < values.arrayLength;
i++)
       total += values[i]
    return total / values.arrayLength
    }
```

```
float square_dist(float a, float b) {
    if (a > b) return square(a – b)
       else return square(b – a)
    }
```

```
float square(float a) {
    return a * a
    }
```



Function call stacks

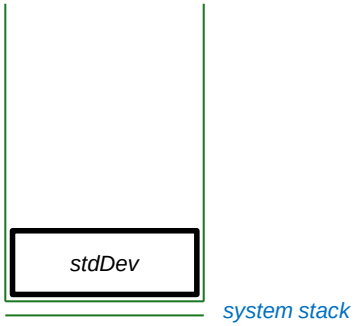
```
float stdDev(float values[]) {
     float avg = average(values)
     float total = 0.0
     for (int i = 0; i < values.length; i++)
        total += square_dist(values[i], avg)
     avg = total / values.arrayLength
     return sqrt(avg)
     }
```

24 bytes of local variables

```
float average(float values[]) {
     float total
     for (int i = 0; i < values.arrayLength;
i++)
        total += values[i]
     return total / values.arrayLength
     }
```

```
float square_dist(float a, float b) {
     if (a > b) return square(a – b)
        else return square(b – a)
     }
```

```
float square(float a) {
     return a * a
     }
```



Call a function: *push a new stack frame*

Function call stacks

```
float stdDev(float values[]) {
     float avg = average(values)
     float total = 0.0
     for (int i = 0; i < values.length; i++)
        total += square_dist(values[i], avg)
     avg = total / values.arrayLength
     return sqrt(avg)
     }
```

24 bytes of local variables

```
float average(float values[]) {
     float total
     for (int i = 0; i < values.arrayLength;
i++)
        total += values[i]
     return total / values.arrayLength
     }
```

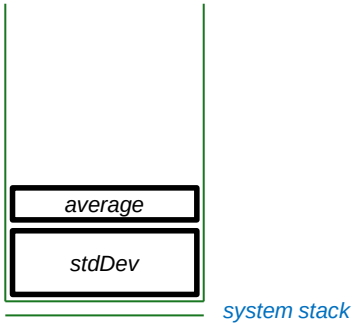
8 bytes

```
float square_dist(float a, float b) {
     if (a > b) return square(a – b)
        else return square(b – a)
     }
```

0 bytes

```
float square(float a) {
     return a * a
     }
```

0 bytes



Call a function: *push a new stack frame*

Function call stacks

```
float stdDev(float values[]) {
    float avg = average(values)
    float total = 0.0
    for (int i = 0; i < values.length; i++)
       total += square_dist(values[i], avg)
    avg = total / values.arrayLength
    return sqrt(avg)
    }
```

24 bytes of local variables

```
float average(float values[]) {
    float total
    for (int i = 0; i < values.arrayLength;
i++)
       total += values[i]
    return total / values.arrayLength
    }
```

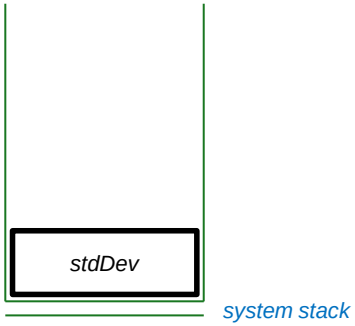
8 bytes

```
float square_dist(float a, float b) {
    if (a > b) return square(a – b)
       else return square(b – a)
    }
```

0 bytes

```
float square(float a) {
    return a * a
    }
```

0 bytes



Return from a function: *pop a stack frame*
(and restore the CPU state)

Function call stacks

```
float stdDev(float values[]) {
    float avg = average(values)
    float total = 0.0
    for (int i = 0; i < values.length; i++)
       total += square_dist(values[i], avg)
    avg = total / values.arrayLength
    return sqrt(avg)
    }
```

24 bytes of local variables

```
float average(float values[]) {
    float total
    for (int i = 0; i < values.arrayLength;
i++)
       total += values[i]
    return total / values.arrayLength
    }
```

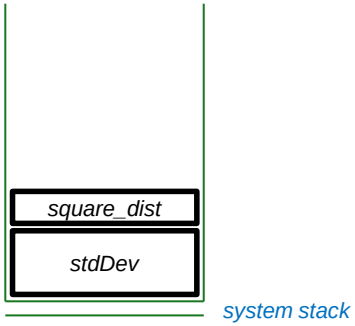
8 bytes

```
float square_dist(float a, float b) {
    if (a > b) return square(a – b)
       else return square(b – a)
    }
```

0 bytes

```
float square(float a) {
    return a * a
    }
```

0 bytes

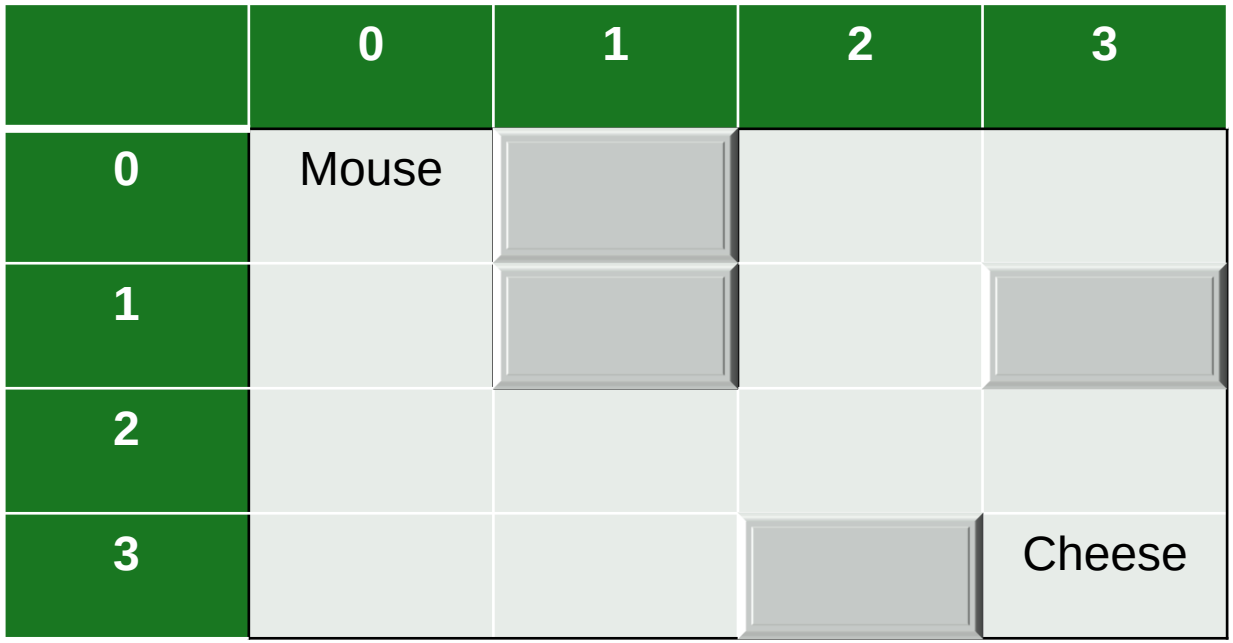


Call a function: *push a new stack frame*

Matching Braces

- Matching
  - {}
  - {{}}
  - {}{}
- Ill-matched
  - }{
  - }
  - {}}
  - {}{
- Can you apply stacks to determine whether a string of open and closes braces is matched or ill-matched?

Searching: Mouse in a Maze



- Mouse can move up, down, left, or right
- A cell is identified by its row and column index (as usual)
- A path is a sequence of (unblocked) cells
- Can you use a stack to help the mouse find a path to the cheese?