Model of Execution

Stack Examples

Lecture Task

- This is Lecture Task 6 from the Pale Blue Dot project -

- **Testing**: In the tests package, complete the test suite named LectureTask6.
  - This suite will test a method in the PaleBlueDot object named “closestCity” that takes a **String** and a **List of Doubles** as parameters and returns a **List of Strings**
  - The inputs represent the filename for the cities file and a latitude/longitude location. This method outputs the country code, name, and region of the city closest to the input location. The upper/lower-case and formatting should match exactly as it appears in the file.
    - Example output format: List("cn", "longjiang", "08")
    - If no closest city is found, the method outputs the empty list: List()

- **Functionality**: Implement the functionality for closestCity.

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Start at the main method
- Create variables x, y

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Call the subtract method
- Add a new Stack Frame for the method call
- Add the parameter variables to the stack

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
<subtract stack frame>
name:x, value:5
name:y, value:3

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Add z to the stack
- Enter a new code block for the for loop
- Add i to the stack

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
<subtract stack frame>
name:x, value:5
name:y, value:3
name:z, value:5
<start for loop block>
name:i, value:0

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Add x to the stack inside the for loop block
- There are 3 variables names x on the stack
- The one in the innermost block is "in scope" and will be used when we ask for the value of x

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
<subtract stack frame>
name:x, value:5
name:y, value:3
name:z, value:5
<start for loop block>
name:i, value:0
name:x, value:20

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- The conditional is false
- Enter the else block
- Put another x on the stack

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
<subtract stack frame>
name:x, value:5
name:y, value:3
name:z, value:5
<start for loop block>
name:i, value:0
name:x, value:20
<start else block>
name:x, value:1

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- We need to compute z - x
- But wait...
  - There are **4** different variables named x on the stack!
  - There's only 1 z, but it's not in this **block**!

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
<subtract stack frame>
name:x, value:5
name:y, value:3
name:z, value:5
<start for loop block>
name:i, value:0
name:x, value:20
<start else block>
name:x, value:1

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Resolve x
  - Start looking in the current block for an **x**
  - We find one -> use it
  - **x** resolves to 1

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
<subtract stack frame>
name:x, value:5
name:y, value:3
name:z, value:5
<start for loop block>
name:i, value:0
name:x, value:20
<start else block>
name:x, value:1

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Resolve z
  - Start looking in the current block for a **z**
  - We don't find one -> expand the search to the next block

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
<subtract stack frame>
name:x, value:5
name:y, value:3
name:z, value:5
<start for loop block>
name:i, value:0
name:x, value:20
<start else block>
name:x, value:1

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Resolve z cont'
  - We don't find **z** in the loop block either -> expand the search to the next block (The stack frame in this case)
  - We find **z** in the stack frame -> Resolves to 5

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
<subtract stack frame>
name:x, value:5
name:y, value:3
name:z, value:5
<start for loop block>
name:i, value:0
name:x, value:20
<start else block>
name:x, value:1

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Resolve z cont'
  - We don't find **z** in the loop block either -> expand the search to the next block (The stack frame in this case)
  - We find **z** in the stack frame -> Resolves to 5

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
<subtract stack frame>
name:x, value:5
name:y, value:3
name:z, value:5
<start for loop block>
name:i, value:0
name:x, value:20
<start else block>
name:x, value:1

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Assign z to 5-1
- End of else block
  - Remove all variables declared in the else block

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
<subtract stack frame>
name:x, value:5
name:y, value:3
name:z, value:4
<start for loop block>
name:i, value:0
name:x, value:20

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Repeat until the loop ends
  - Remove it's block from the stack

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
<subtract stack frame>
name:x, value:5
name:y, value:3
name:z, value:2

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Method returns
  - Remove its stack frame from the stack
  - subtract(x, y) resolves to 2
  - Add z to the stack and assign it the value 2

Memory Stack
<main stack frame>
args
name:x, value:5
name:y, value:3
name:z, value:2

Variable Scope Example

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (isNegative(y)) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}

def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 3
  val z: Int = subtract(x, y)
  println(z)
}
```

- Main method ends
  - Remove it's frame from the stack
- Program ends

Memory Stack

More Memory Examples

Multiple frames on the stack

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Memory Stack
args
<new stack frame>
name:n, value:3
<Used by another program>
<Used by another program>

- Call function
- Create new stack frame

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Memory Stack
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<Used by another program>
<Used by another program>

- Enter if block
- Call function again
- Create new stack frame

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Memory Stack
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<Used by another program>
<Used by another program>

- In next function call, conditional true
- New if block
- New stack frame

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Memory Stack
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<if block>
<new stack frame>
name:n, value:0
<Used by another program>
<Used by another program>

- Repeat, repeat
- Many variables named n on the stack
- Each is in different frame so it's ok

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Memory Stack
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<if block>
<new stack frame>
name:n, value:0
<Used by another program>
<Used by another program>

- Conditional finally false
- return 0

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

- Assign return value to result

Memory Stack
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<if block>
name:result, value:0
<Used by another program>
<Used by another program>

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Memory Stack
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<if block>
name:result, value:1
<Used by another program>
<Used by another program>

- Add value of the n in this stack frame to result
- result is the last expression and is returned

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Memory Stack
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
name:result, value:1
<Used by another program>
<Used by another program>

- Return to function call from previous frame
- Store return value in result

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

- Add value of n from this frame..
- Repeat

Memory Stack
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
name:result, value:3
<Used by another program>
<Used by another program>

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

- Add value of n from this frame..
- Repeat

Memory Stack
args
<new stack frame>
name:n, value:3
<if block>
name:result, value:3
<Used by another program>
<Used by another program>

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Memory Stack
args
<new stack frame>
name:n, value:3
<if block>
name:result, value:6
<Used by another program>
<Used by another program>

- And repeat..
- Imagine if the original input were 1000
  - This is why we use computers

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Memory Stack
args
name:result, value:6
<Used by another program>
<Used by another program>

- Value result in main method gets the last return value

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

- print 6

Memory Stack
args
name:result, value:6
<Used by another program>
<Used by another program>

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  if(n>0) {
    var result: Int = computeGeometricSum(n - 1)
    result += n
    result
  }else{
    0
  }
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

- Free memory

Memory Stack
<Used by another program>
<Used by another program>

More Memory Examples

- We were close to the end of the stack on that example
  - In reality, the stack will be much larger than in this example
- What if this were our code?

```
def computeGeometricSum(n: Int): Int ={
  var result: Int = computeGeometricSum(n - 1)
  result += n
  result
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  var result: Int = computeGeometricSum(n - 1)
  result += n
  result
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Memory Stack
args
<new stack frame>
name:n, value:3
<if block>
<new stack frame>
name:n, value:2
<if block>
<new stack frame>
name:n, value:1
<if block>
<new stack frame>
name:n, value:0
<Used by another program>
<Used by another program>

- At this point the other program was going to return 0 and return back up the stack

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  var result: Int = computeGeometricSum(n - 1)
  result += n
  result
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

Memory Stack
args
<new stack frame>
name:n, value:3
<new stack frame>
name:n, value:2
<new stack frame>
name:n, value:1
<new stack frame>
name:n, value:0
<new stack frame>
name:n, value:-1
<new stack frame>
name:n, value:-2
<Used by another program>
<Used by another program>

- This program keeps adding frames to the stack

Recursive Example

```
def computeGeometricSum(n: Int): Int ={
  var result: Int = computeGeometricSum(n - 1)
  result += n
  result
}

def main(args: Array[String]): Unit = {
  val result: Int = computeGeometricSum(3)
  println(result)
}
```

- STACK OVERFLOW
- Program crashes

Memory Stack
args
<new stack frame>
name:n, value:3
<new stack frame>
name:n, value:2
<new stack frame>
name:n, value:1
<new stack frame>
name:n, value:0
<new stack frame>
name:n, value:-1
<new stack frame>
name:n, value:-2
<Used by another program> <new stack frame>
<Used by another program> name:n, value:-3

Debugger Demo

Lecture Task

- This is Lecture Task 6 from the Pale Blue Dot project -

- **Testing**: In the tests package, complete the test suite named LectureTask6.
  - This suite will test a method in the PaleBlueDot object named “closestCity” that takes a **String** and a **List of Doubles** as parameters and returns a **List of Strings**
  - The inputs represent the filename for the cities file and a latitude/longitude location. This method outputs the country code, name, and region of the city closest to the input location. The upper/lower-case and formatting should match exactly as it appears in the file.
    - Example output format: List("cn", "longjiang", "08")
    - If no closest city is found, the method outputs the empty list: List()

- **Functionality**: Implement the functionality for closestCity.