Recursion & The Stack

AP Computer Science

Recursion

- Recursion occurs when a method calls itself.
  - What is the output of the method below?
  - Will it ever end?

```
run(1);
```

```
public void run(int x) {
  System.out.println(x);
  run(x + 1)
}
```

Output
1
2
3
...

Base Case

- What is different about these methods?

```
public void run(int x) {
  System.out.println(x);
  run(x + 1)
}
```

```
public void run(int x) {
  System.out.println(x);
  if (x < 3)
    run(x + 1);
}
```

Base Case

- A recursive method must have a stop condition, i.e. "base case"
- Recursive calls will continue until the stop condition is met

```
run(1);
```

```
public void run(int x) {
  System.out.println(x);
  if (x < 3) // base case
    run(x + 1);
}
```

Output
1
2
3

Recursion Rules

- Base Case
  - Always have at least one case that can be solved without using recursion
- Make Progress
  - Any recursive call must progress toward a base case.
- A recursive solution solves a small part of the problem and leaves the rest of the problem in the same form as the original

Tracing Recursion

- What's different in this example?
- Will the output change?

```
run(1);
```

```
public void run(int x) {
  if (x < 3) // base case
    run(x + 1);
  System.out.println(x);
}
```

Output
3
2
1

The Stack

Activation Records

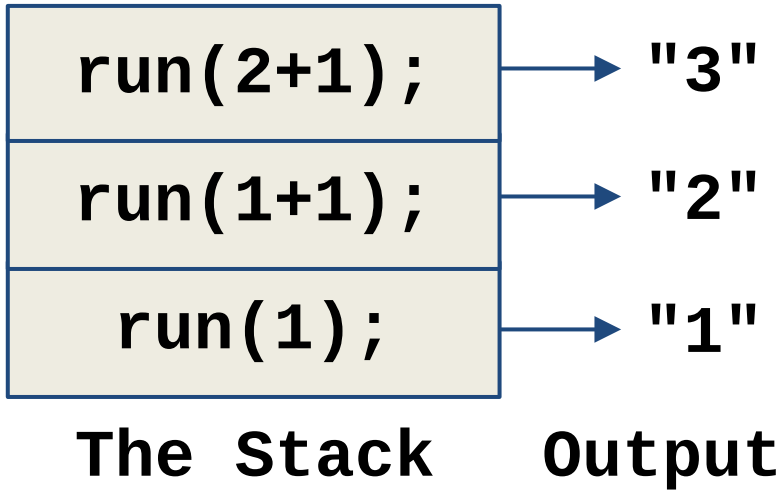
- When we call a method, e.g. **`run(1)`**, all relevant (the **`1`** and where to return once **`run(1)`** is over) is placed in an *activation record*
- The activation record is pushed onto the program stack
  - Think of a stack as a deck of cards, you can only access the top card or "push" more on top of it

Activation Records

- Consider how the activation records are pushed on the stack for this method call
- Once the function is over, it's removed from the stack.

```
run(1);
```

```
public void run(int x) {
  System.out.println(x);
  if (x < 3) // base case
    run(x + 1);
}
```

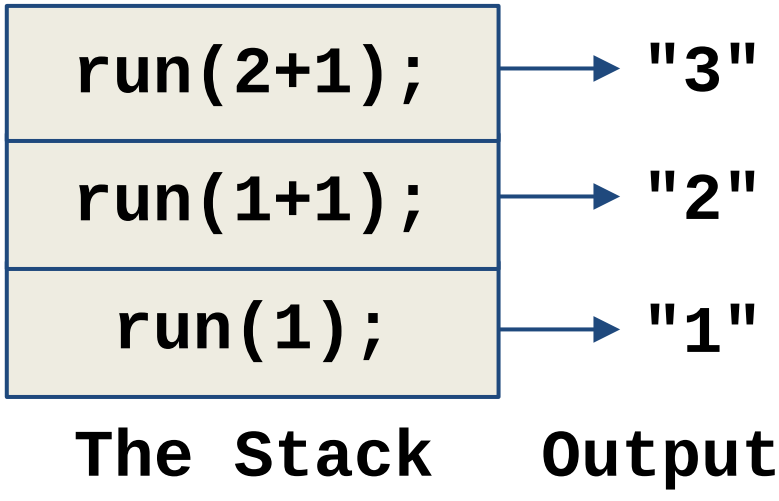


Activation Records

- Notice the difference in the stack modification versus output!

```
run(1);
```

```
public void run(int x) {
  if (x < 3) // base case
    run(x + 1);
  System.out.println(x);
}
```



Evaluating Recursion

- What is returned by **`mystery(3)`**?
  - `m(3) = 3 * m(2)`
  - `m(2) = 3 * m(1)`
  - `m(1) = 3 * m(0)`
  - `m(0) = 1`

```
public int mystery(int n) {
  if (n == 0)
    return 1;
  else
    return 3 * mystery(n-1);
}
```

Returns
27

Evaluating Recursion

- So what does **`mystery`** compute?

```
public int mystery(int n) {
  if (n == 0)
    return 1;
  else
    return 3 * mystery(n-1);
}
```

Evaluating Recursion

- What is returned by **`boogie(5, 6)`**?
  - **`b(5, 6) = b(5, 4) + 5`**
  - **`b(5, 4) = b(5, 2) + 5`**
  - **`b(5, 2) = b(5, 0) + 5`**
  - **`b(5, 0) = 5`**

```
int boogie(int x, int y) {
 if(y < 2)
    return x;
 else
    return boogie(x,y-2) + x;
}
```

Returns
20

Evaluating Recursion

- What is returned by **`fun(3)`**?
  - **`f(3) = 3 + f(2) + f(1)`**
  - **`f(2) = 2 + f(1) + f(0)`**
  - **`f(1) = 1 + f(0) + f(-1)`**
  - **`f(0) = 1, f(-1) = 1`**

```
int fun(int x){
  if(x < 1)
     return 1;
  else
     return x+fun(x-1)+fun(x-2);
}
```

Returns
12

A Reminder

- Recursion is just another tool to use
- It is not a good tool for all problems
  - We will implement several algorithms and methods where a looping solution would work just fine
- You must realize when it's time to use recursion!

Additional Resources

Additional Resources

- Here are some additional resources:
  - Recursion Explained with the Flood Fill Algorithm
  - Flood Fill (Wikipedia)
  - Visualizing Recursion
  - Recursive Methods and Problem Solving
  - Recursion (Online Textbook)
  - Flood Fill (Example shown in class)