Looping in time

while and for loops

CS111 Computer Programming

Department of Computer Science
Wellesley College

Iteration table for factTail

```
public static int factTail (int num, int ans)
{
     if (num == 0)
          return ans;
     else
          return factTail(num-1, num*ans);
}
```

Iteration table shows the values of variables for each invocation

Invocation	num	ans
1	4	1
2	3	4
3	2	12
4	1	24
5	0	24

*Remember factIter(n) invokes factTail(n, 1).

While construct*

1) evaluate boolean expression

```
while (test_expression) {
    // statements
}
// end of while loop
```

2) if true, execute body of loop and goto step 1.

3) if false, goto to statement after while loop.

* Reads: while boolean expression is true, execute body of loop.

How to exit from a while loop?

```
public int countDown (int n)
{
    while (n > 0) {
        System.out.println(n);
        n = n - 1;
    }
    System.out.println("blastoff!");
}
```

countDown(5) →

5
4
3
2
1
blastoff!

Houston, we have a problem

```
public int countDown (int n)
{
     while (n > 0) {
          System.out.println(n);
     }
     System.out.println("blastoff!");
}
```

countDown(5) ⟶

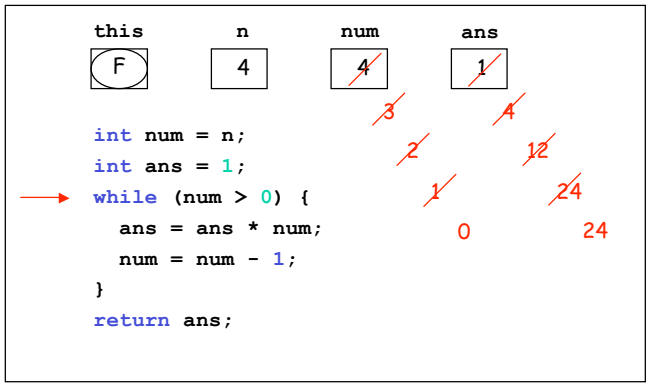
5
5
5
5
5
5
5
5
5
5
5
5

Factorial with a `while` loop

```
public static int factWhile (int n)
{
     int num = n;            // declaration of
     int ans = 1;            // local variables
     while (num > 0) {
          ans = ans * num;   // calculate product
          num = num - 1;     // and decrement num
     }
     return ans;
}
```

Executionland

F .factWhile(4)



Iteration table for `factWhile`

```
public static int factWhile (int n)
{
     int num = n;
     int ans = 1;
     while (num > 0) {
          ans = ans * num;
          num = num - 1;
     }
     return ans;
}
```

Exactly the same table as `factTail()`

Loop #	num	ans
1	4	1
2	3	4
3	2	12
4	1	24
5	0	24

factTail() and factWhile()

```
public static int factIter(int n)
{
  return factTail(n, 1);
}
public static int factTail (int num, int ans) {
  if (num == 0)
    return ans;
  else
    return factTail(num-1, num*ans);
}
```

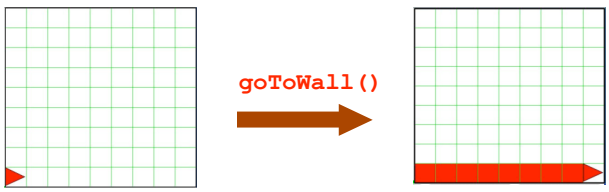
Initialize variables

When done, return ans

```
public static int factWhile (int n)
{
  int num = n;
  int ans = 1;
  while (num > 0) {
    ans = ans * num;
    num = num - 1;
  }
  return ans;
}
```

While not done, update variables

Let's try that again

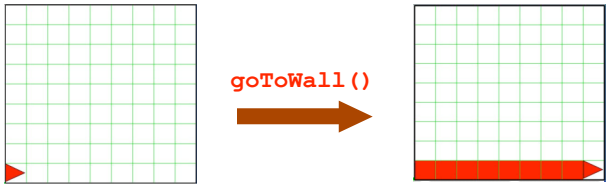


Tail recursive solution

Tail recursion

```
public void goToWallRec()
{
     if (isFacingWall()) {
          // do nothing
     } else {
          forward();
          goToWallRec();
     }
}
```


Implicit update of state



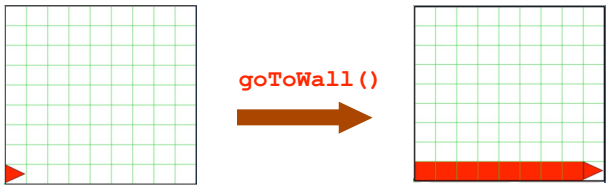
Tail recursion

```
public void goToWallRec()
{
     if (isFacingWall()) {
          // do nothing
     } else {
          forward();
          goToWallRec();
     }
}
```

Iterative

```
public void goToWallWhile()
{
     while (!isFacingWall()) {
          forward();
     }
     // do nothing
}
```

A more elegant tail recursive solution



Tail recursion

```
public void goToWallRec()
{
     if (!isFacingWall()) {
          forward();
          goToWallRec();
     }
}
```


The corresponding iterative solution

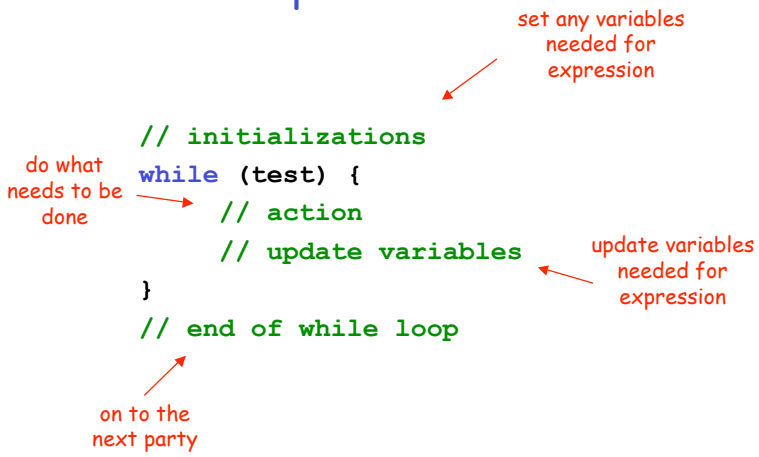
Tail recursion

```
public void goToWallRec()
{
     if (!isFacingWall()) {
          forward();
          goToWallRec();
     }
}
```

Iterative

```
public void goToWallWhile()
{
     while (!isFacingWall()) {
          forward();
     }
}
```

Typical `while` loop



Factorial with a `while` loop

```
public static int factWhile (int n)
{
     int num = n;            // declaration of
     int ans = 1;            // local variables
     while (num > 0) {
          ans = ans * num;   // calculate product
          num = num - 1;     // and decrement num
     }
     return ans;
}
```

factFor()

```
public static int factWhile (int n)
{
     int num = n;
     int ans = 1;
     while ( num > 0 ) {
          ans = ans * num;
          num = num - 1;
     }
      return ans;
}
```

Iteration table n = 4

Loop #	num	ans
1	4	1
2	3	4
3	2	12
4	1	24
5	0	24

```
public static int factFor (int n)
{
     int ans = 1;
     for ( int num = n; num > 0; num = num-1 ) {
          ans = ans * num;
     }
     return ans;
}
```


for loop syntax

a.k.a.
continue condition

```
for ( Initialization of loop counter ; boolean_expr ; update loop counter ) {

        // where the action is

}
// end of for loop
```

As long as the continue condition holds, execute body of for loop

As soon as expr is false drop down to here

Variations on a theme*

```
public static int factForUp (int n)
{
      int ans = 1;
      for ( int i = 1;  i <= n;  ++i) {
            ans = ans * i;
      }
      return ans;
}
```

Iteration table n = 4

Loop #	i	ans
1	1	1
2	2	2
3	3	6
4	4	24
5	5	24

*For those who prefer counting up.

Remember me

```
public void bagelForward(int n)
{
      if (n <= 0) {                    // base case
                                       // do nothing
      } else {                         // recursive case
            dropBagel();
            forward();
            bagelForward(n - 1)
      }
}
```

More succinctly

```
public void bagelForward(int n)
{
     if (n > 0) {
          dropBagel();                  // recursive case
          forward();
          bagelForward(n - 1)
     }
                                        // base case (do nothing)
}
```


bagelForwardWhile()

```
public void bagelForward(int n)
{
     if (n > 0) {
          dropBagel();                  // recursive case
          forward();
          bagelForward(n - 1)
     }                                  // base case (do nothing)
}

 public void bagelForwardWhile(int n)
 {
      while (n > 0) {
           dropBagel();
           forward();
           n = n - 1;
      }
 }
```

bagelForwardFor()

```
public void bagelForward(int n)
{
     if (n > 0) {
          dropBagel();                    // recursive case
          forward();
          bagelForward(n - 1)
     }                                    // base case (do nothing)
}
```

```
public void bagelForwardFor(int n)
{
     for (          ; n > 0; n--)   {
          dropBagel();
          forward();
   }
}
```


Or, if you prefer

```
public void bagelForward(int n)
{
     if (n > 0) {
          dropBagel();                    // recursive case
          forward();
          bagelForward(n - 1)
     }                                    // base case (do nothing)
}
```

```
public void bagelForwardFor(int n)
{
     for (int i = n; i > 0; i--)   {
          dropBagel();
          forward();
     }
}
```

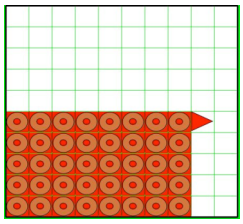
Conversion from `for` to `while` is always possible

```
for (init; test; update) {
  action;
}
```

```
{
  init;
  while (test) {
    action;
    update;
  }
}
```


bagelRectangle(int width, int height)

```
bagelRectangle(8,5);
```



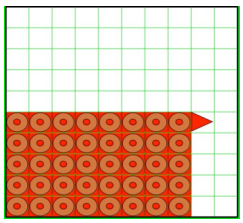
*Slightly different from our original bagelRectangle(); lower left corner always at origin.

Recycling code

```
public void bagelRectangle(int width, int height)
{
     for (int h = 1; h <= height; h++)  {
          setPosition(new Point(1, h));
          bagelForwardFor(width);
     }
}

public void bagelForwardFor(int n)
{
     for (int i = n; i > 0; i--)  {
          dropBagel();
          forward();
     }
}
```

bagelRectangle(8,5);

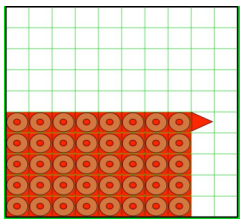


Nested loops

```
public void bagelRectangle(int width, int height)
{
     for (int h = 1; h <= height; ++h)  {
          setPosition(new Point(1, h));

          // bagelForwardFor(width);
          for (int i = width; i > 0; --i)  {
               dropBagel();
               forward();
          }
     }
}
```

bagelRectangle(8,5);

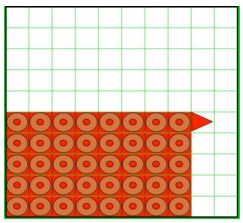


Or, if you prefer

```
public void bagelRectangle(int width, int height)
{
     for (int h = 1; h <= height; ++h)  {
          setPosition(new Point(1, h));

          // drop "width" number of bagels
          for (int w = 1; w <= width; ++w)  {
               dropBagel();
               forward();
          }
     }
}
```

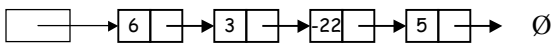
bagelRectangle(8,5);



Iteration with recursive data structures

```
// Returns the integer sum of all elements in L
public static int sumList (IntList lst)
{
     int result = 0;
     if (!isEmpty(lst))  {
          result = head(lst) + sumList(tail(lst));
     }
     return result;
}
```

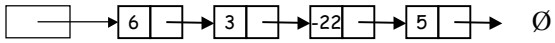
list1



sumListWhile()

```
// Returns the integer sum of all elements in L
public static int sumListWhile (IntList lst)
{
     int result = 0;
     while (!isEmpty(lst))  {
          result = head(lst) + result;     // action
          lst = tail(lst);                 // update
     }
     return result;
}
```

list1



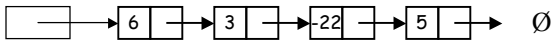

```
init;
while (test) {
  action;
  update;
}
```

```
for (init; test; update) {
  action;
}
```


sumListFor()

```
// Returns the integer sum of all element in L
public static int sumListFor (IntList lst)
{
     for (int result=0; !isEmpty(lst); lst=tail(lst)) {
          result = head(lst) + result;
     }
     return result;
}
```

list1



*This is the straightforward translation. But there is a problem!

Variable scope

```
// Returns the integer sum of all elements in L
public static int sumListFor (IntList lst)
{
     int result;
     for (result=0; !isEmpty(lst); lst=tail(lst)) {
          result = head(lst) + result;
     }
     return result;
}
```