Recursion

Professor: Suman Saha

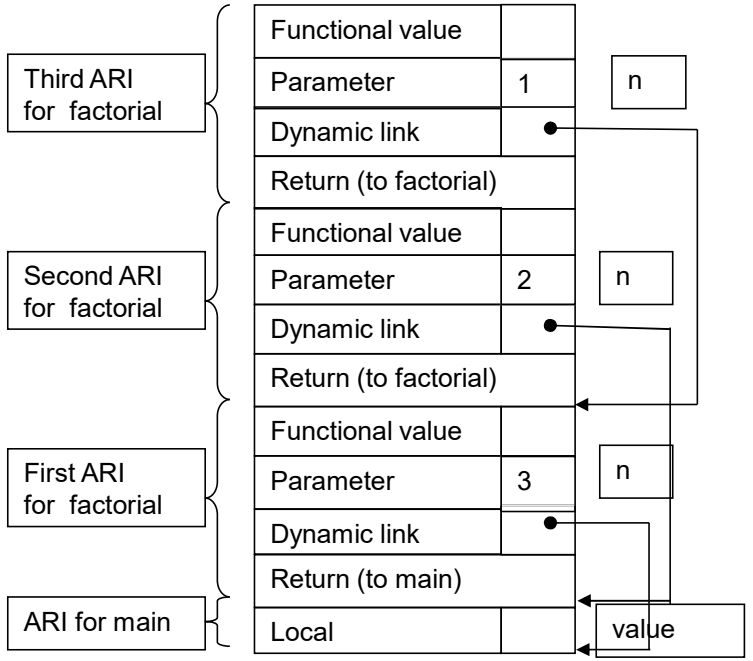
Activation records support recursive functions

PennState

- *Why do recursive functions require that local variables be dynamically allocated?*
  - we do not know until run-time how deep the recursion will go, thus we cannot know how many copies we will need
  - if the language is not recursive, activation records can be statically allocated
    - but this may waste memory, because some functions may never be called

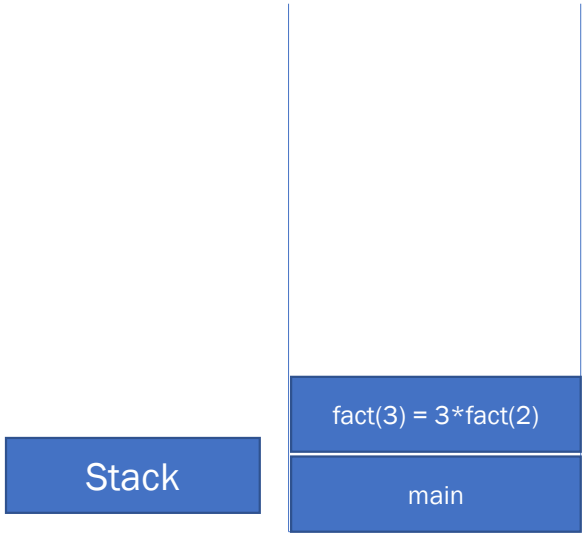
Recursion Example

```
int fact (int n) {
    // enter function
    if (n <= 1)
        return 1;
    else return (n * fact(n-1));
    // exit function
}


void main() {
    int value;
    value = fact(3);
}
```



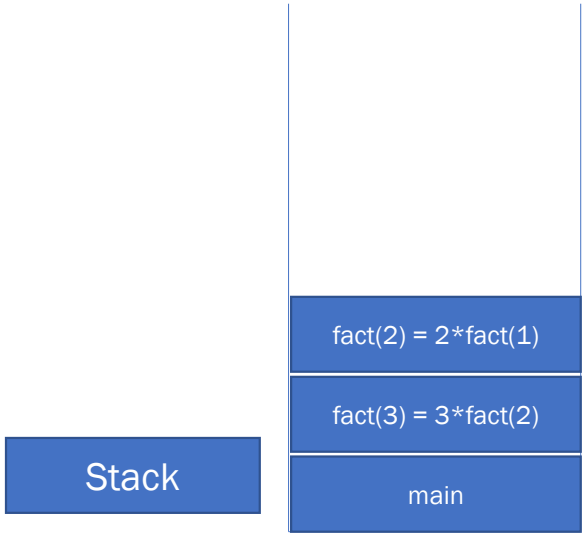
Recursion Example

```
int fact (int n) {
    // enter function
    if (n <= 1)
        return 1;
    else return (n * fact(n-1));
    // exit function
}


void main() {
    int value;
    value = fact(3);
}
```



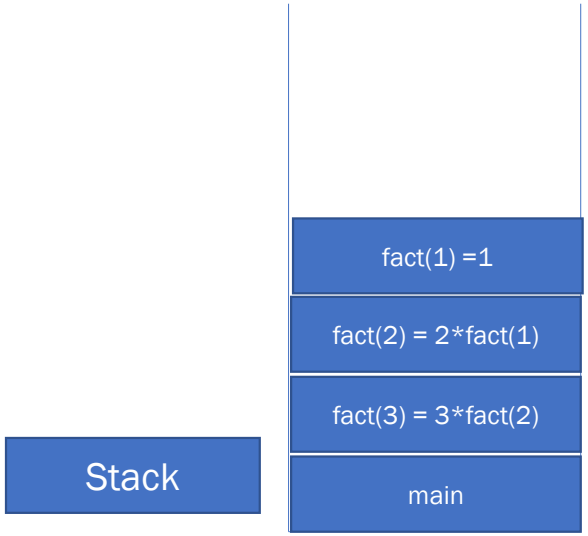
Recursion Example

```
int fact (int n) {
    // enter function
    if (n <= 1)
        return 1;
    else return (n * fact(n-1));
    // exit function
}


void main() {
    int value;
    value = fact(3);
}
```



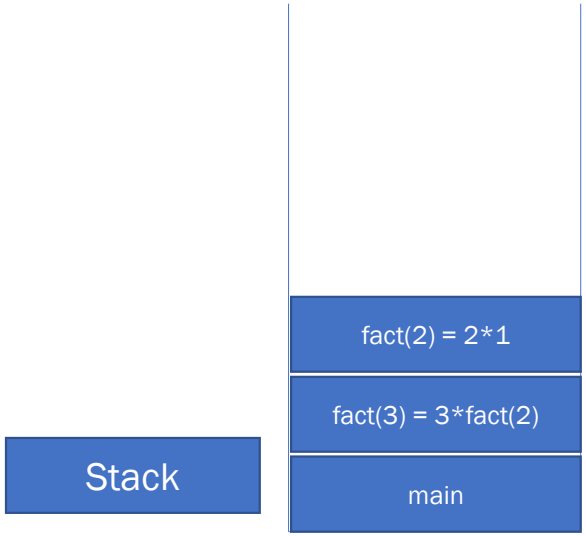
Recursion Example

```
int fact (int n) {
    // enter function
    if (n <= 1)
        return 1;
    else return (n * fact(n-1));
    // exit function
}


void main() {
    int value;
    value = fact(3);
}
```



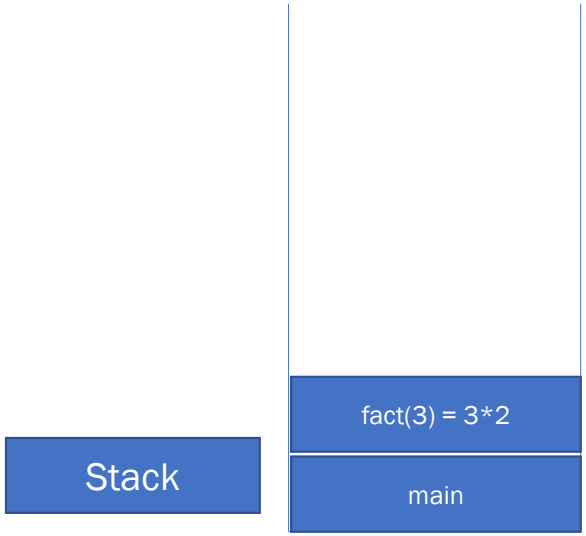
Recursion Example

```
int fact (int n) {
    // enter function
    if (n <= 1)
        return 1;
    else return (n * fact(n-1));
    // exit function
}


void main() {
    int value;
    value = fact(3);
}
```



Recursion Example

```
int fact (int n) {
    // enter function
    if (n <= 1)
        return 1;
    else return (n * fact(n-1));
    // exit function
}


void main() {
    int value;
    value = fact(3);
}
```



Recursion Example

```
int fact (int n) {
    // enter function
    if (n <= 1)
        return 1;
    else return (n * fact(n-1));
    // exit function
}


void main() {
    int value;
    value = fact(10);
}
```

Tail-Recursive Functions

PennState

- A different implement of fact:

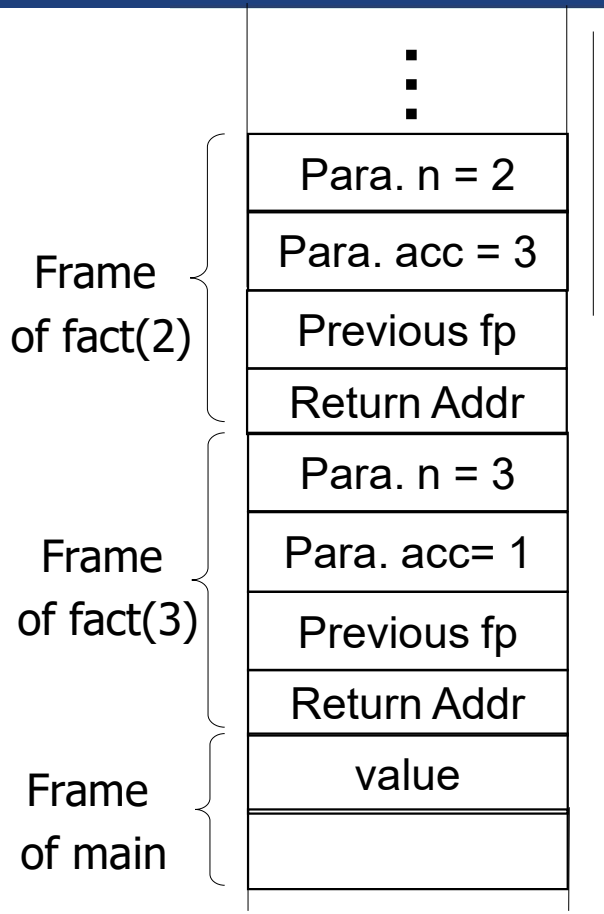
```
int fact(int n, int acc ) {
  if ( n <= 1 ) return acc ;
  else return fact(n-1, acc*n);
}
```

- Function f makes a tail call to function g if the call is the last thing in function f; the return value of calling g is return value of f

Stack of ARs

```
int fact(int n, int acc ) {
  if ( n <= 1 ) return acc ;
  else return fact(n-1, acc*n);
}
```

- Can we destroy the frame of fact(3) before going into fact(2)?

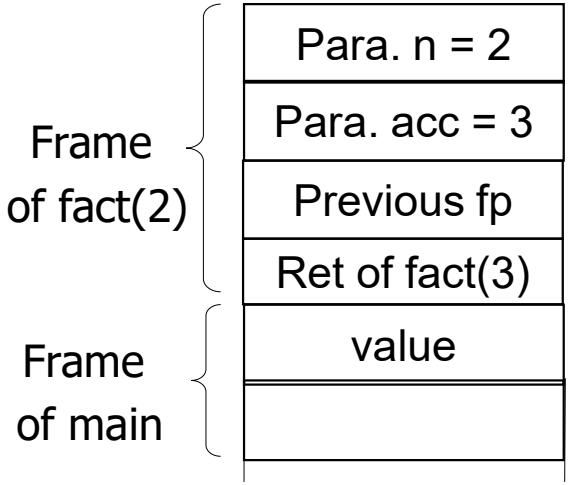


A Different Implementation

PennState

```
int fact(int n, int acc ) {
  if ( n <= 1 ) return acc ;
  else return fact(n-1, acc*n);
}
```

- The compiler is able to use jump statements instead of method calls. This means that calls to itself in the recursion do not add to the call stack.



Tail-Recursive Functions

PennState

- Tail-recursive functions are equivalent to loops

```
int fact(int n, int prev ) {
  if ( n <= 1 ) return prev ;
  else return fact(n-1, prev*n);
}
```

```
int fact(int n, int prev ) {
  while (true) {
    if ( n <= 1 ) return prev ;
    else {prev = prev*n; n--;);
}}
```

Top Hat

PennState