Procedures and Functions

CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

Terminology

Function (Fortran, Ada)*:* returns a value

Procedure (Ada)*, Subroutine* (Fortran): returns no value

C-like language: both are called *function*

Method (C++, Java): a function declared inside a class

Caller & Callee

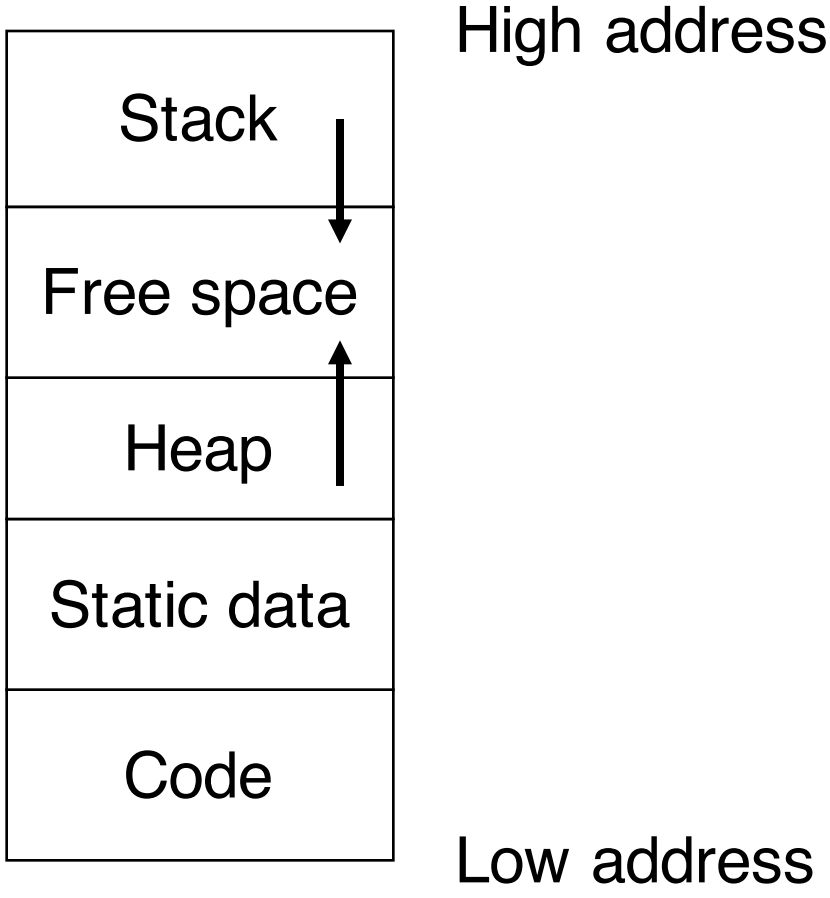
When function A calls function B

- A is the *Caller*
- B is the *Callee*

What needs to be done?

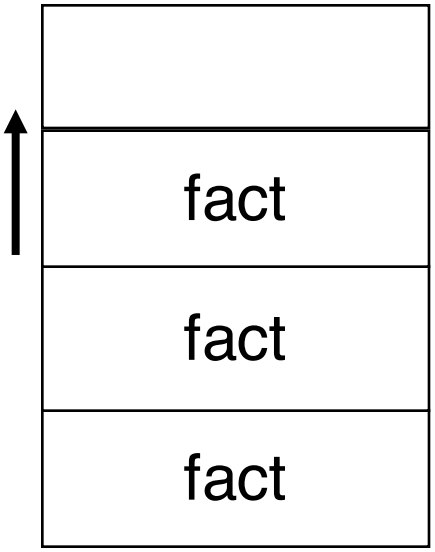
Who is responsible?

Review of Storage Layout



Support for Functions

```
int fact(int n) {
  int tmp;
  if n=0 return 1;
  else
    {tmp = n-1;
     return n*fact(tmp);}
}
```

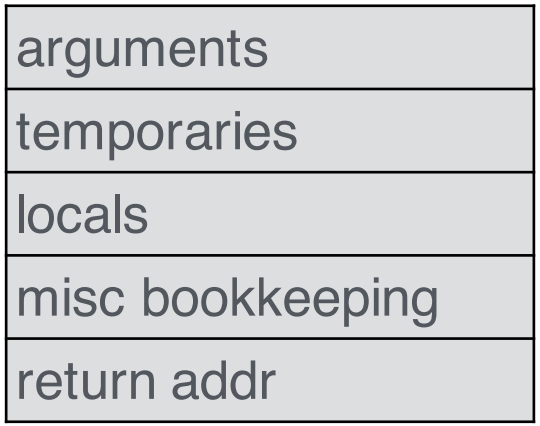


During execution, ***each function call has one frame*** (activation record) on the stack

Stack Frame (Activation Record)

A stack frame contains:

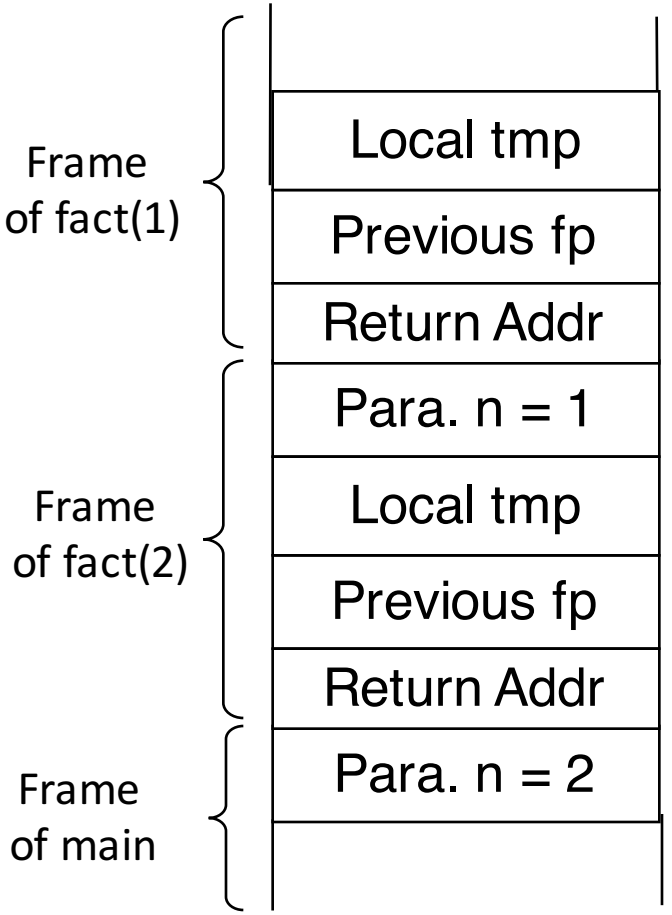
- Local variables
- Temporaries
- Return values
- Bookkeeping info
- Arguments (to the callee)



Stack Frame (Activation Record)

```
int fact(int n) {
  int tmp = 0;
  if n=1 return 1;
  else
    {tmp = n-1;
     return n*fact(tmp);}
}
```

Calling fact(2) in main



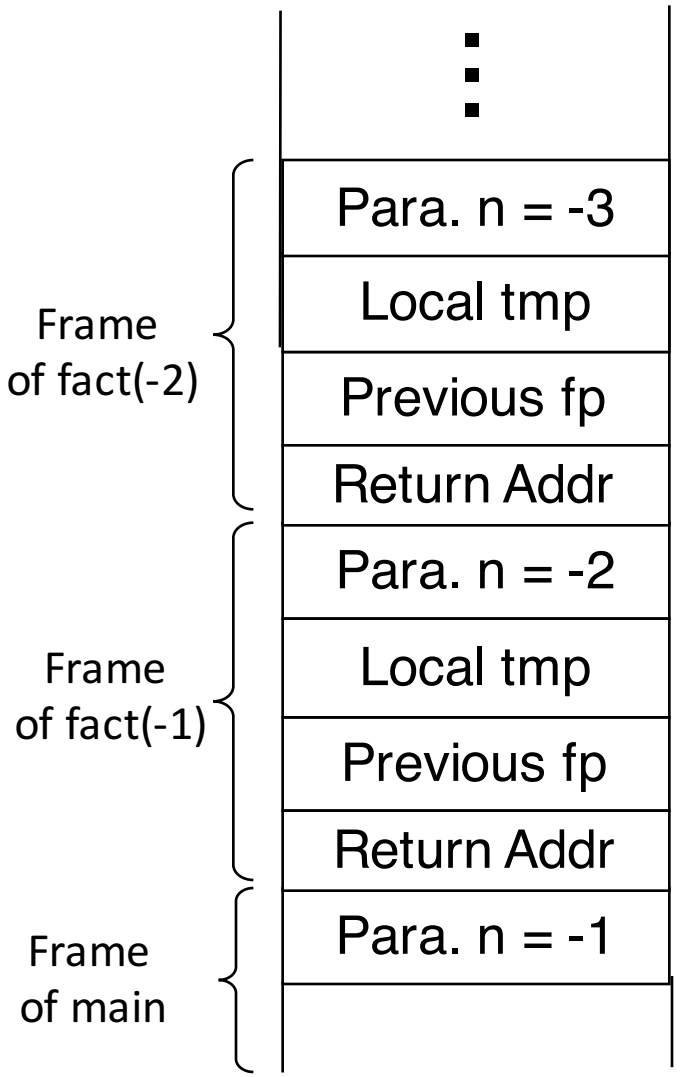
```
int fact(int n) {
  int tmp = 0;
  if n=1 return 1;
  else
    {tmp = n-1;
     return n*fact(tmp);}
}
```

What happens for `fact(-1)`?

Stack is exhausted
(stack overflow)

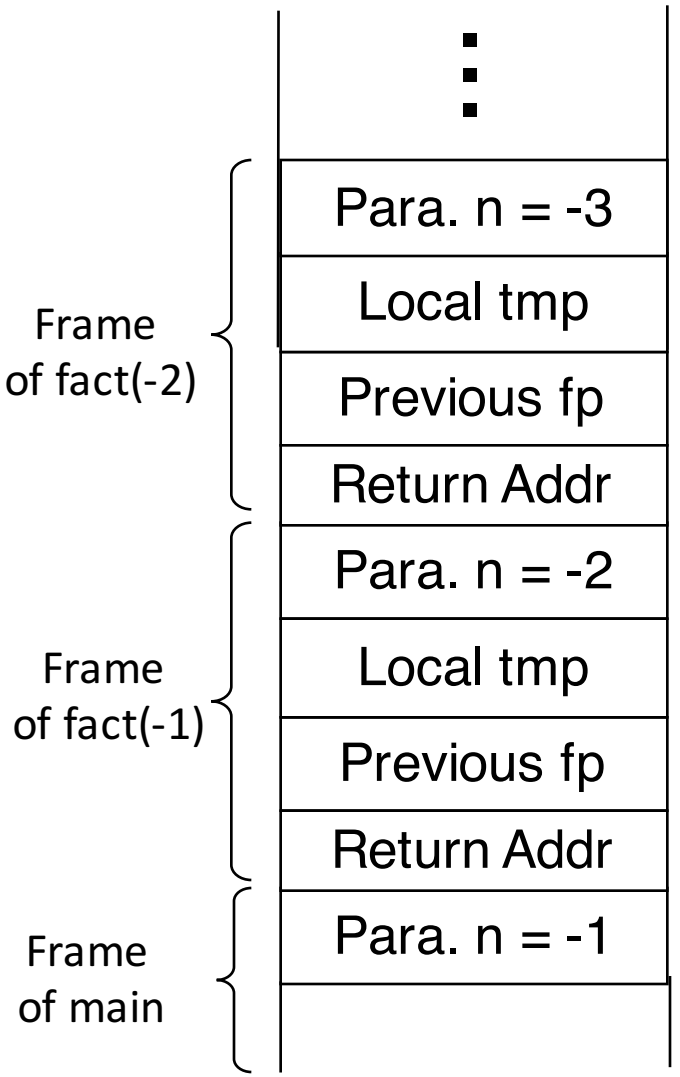
What happens for `fact(1000)`?

Consumes memory



```
int fact(int n) {
  int tmp = 0;
  if n=1 return 1;
  else
    {tmp = n-1;
     return n*fact(tmp);}
}
```

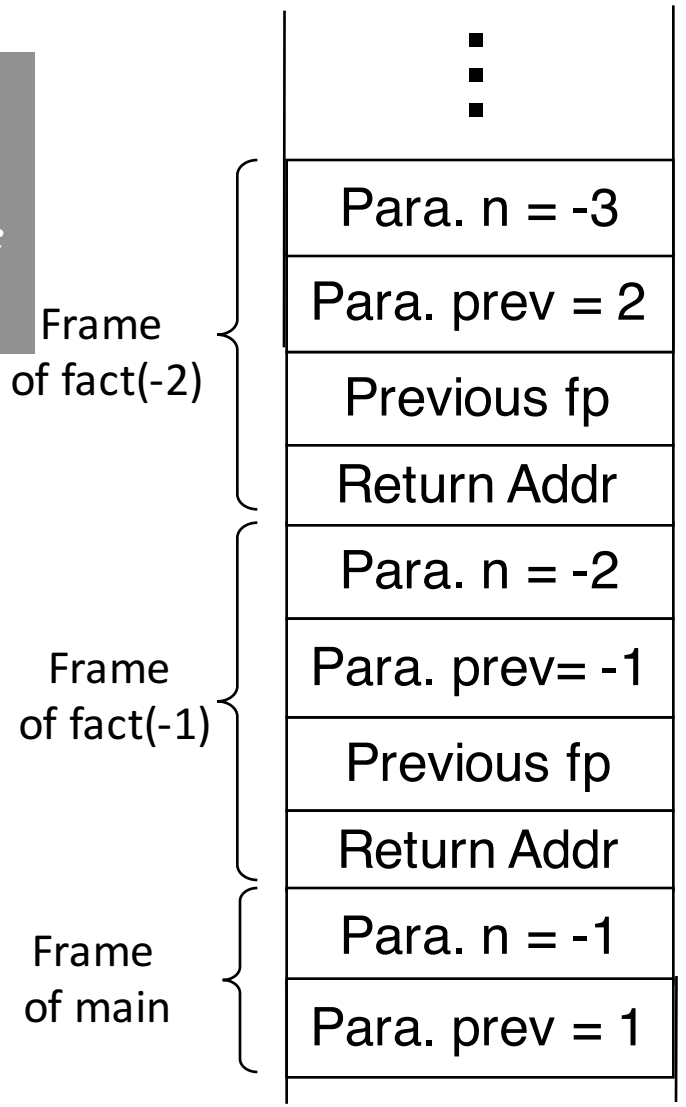
Can we destroy the frame of fact(-1) before going into fact(-2)?



A Different Implementation

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

Can we destroy the frame of fact(-1) before going into fact(-2)?

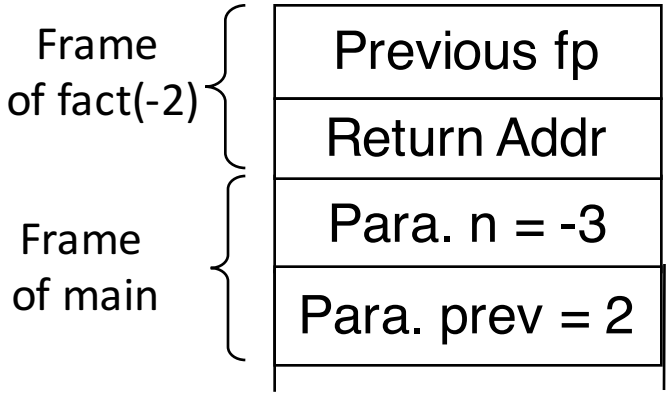


A Different Implementation

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

Can we destroy the frame of fact(-1) before going into fact(-2)?

What change enables the more efficient implementation?



Tail-Recursive Functions

A function that calls itself as the very last thing: the result of the function is a recursive call

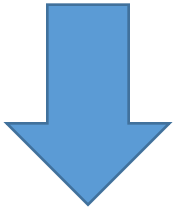
```
int fact(int n, int prev ) {
  if ( n == 1 ) return prev ;
  else return fact(n-1, prev*n);
}
```

The current activation record before recursive call is useless!

Tail-Recursive Functions

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--;);
}}
```

Is this function tail recursive?

```
int search(int[] a, int k, int fst, int lst) {
  if ( fst == lst ) return (a[fst]==k?fst:-1);
  int m = (fst + lst)/2;
  if (k <= a[m]) return search(a, k, fst, m);
  else return search(a, k, m+1, lst);
}
```

Loop version

```
int binSearch(int[] a, int k, int fst, int lst) {
  while (true) {
    if ( fst == lst ) return (a[fst]==k?fst:-1);
    int m = (fst + lst)/2;
    if (k <= a[m]) lst = m;
    else fst = m+1;
  }
}
```

Tail-Recursive Functions to Loops

Tail-recursive

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

Loops

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

In C and Java, the loop version is more efficient.
Manual translation is needed for performance

In most functional languages (e.g., Scheme, Ocaml),
the compiler automatically generate code as efficient as
the loop version