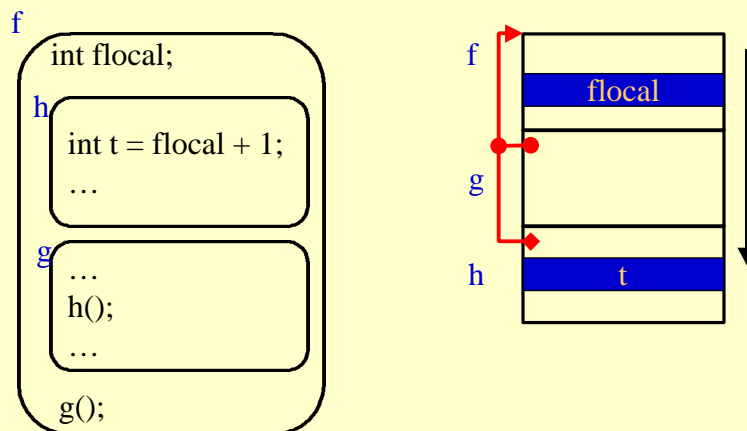


Nested first-class functions

Amer Diwan

Static links

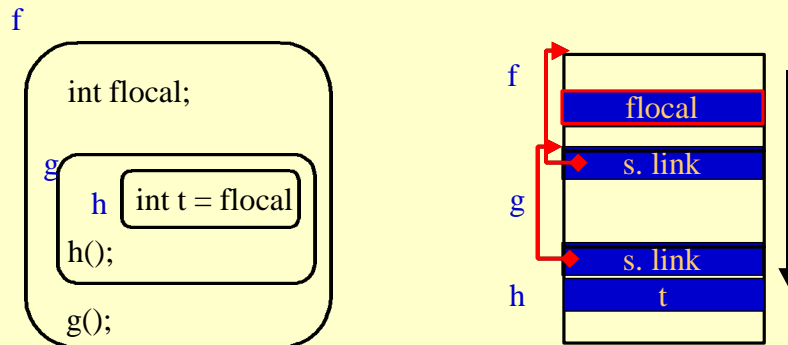


How does "h" access "flocal"? static links is one answer...

Accessing non-local variables

- To access a non-local variable (say `flocal`) declared in the immediately enclosing scope:
 - `*(static_link + offset_of_flocal)`
- To access a non-local variable declared `n` scopes away:
 - follow static links "`n`" times

An example



Passing functions as pointers

Assume that C has nested functions...

```
void do_something(i, l) {  
    int n;  
    int incrn(int elem) {  
        return elem + n;  
    }  
    n = i;  
    map(l, incrn);  
}
```

```
int_list *map(lst, fcn) {  
    if (lst == NULL) return NULL;  
    list *rest = map(lst->tail, fcn);  
    return cons(fcn(lst->head), rest);  
}
```

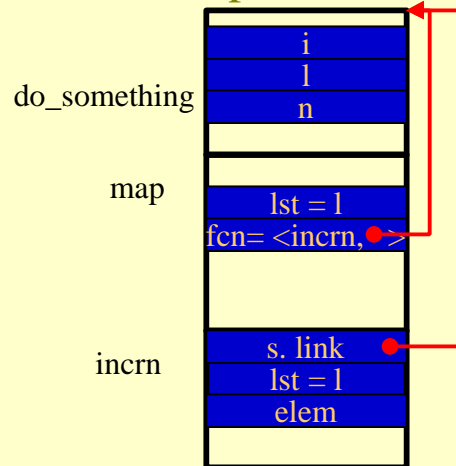
How will incrn access n?

Closures

- A `<code, environment>` pair
- `code` can get to variables in outer scopes through the `environment`
- Example of environment: **the static link**

When you need to pass a function or return a function, pass a closure

Example with closure



What's in a name?

- The closure of a function is the **closed form** of the function
 - no free variables
 - to access previously free variables, must go through the environment component of closure

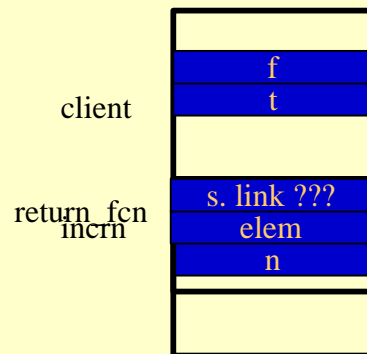
Escaping functions

```
void client() {  
    f = return_fcn(10);  
    t = f(5)  
}
```

```
return_fcn(int i) {  
    int n;  
    int incrn(int elem) {  
        return elem+n;  
    }  
    n = i;  
    return incrn;  
}
```

What's the problem here?

Example continued



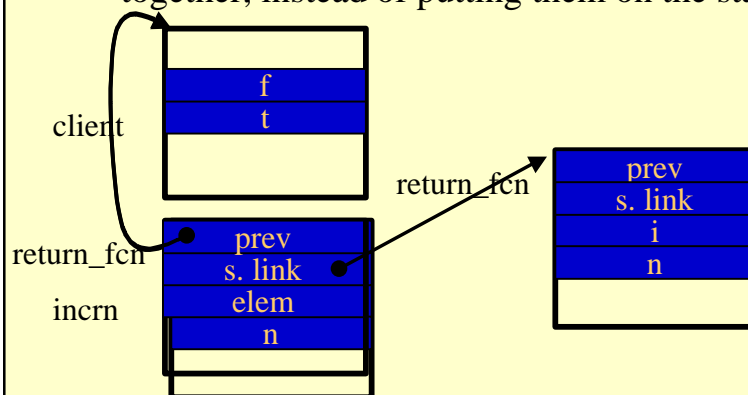
We would like incrn's static link to point to return_fcn's activation record, but that has been popped off the stack!

Solutions

- Don't have nested functions: [C/C++/Java](#)
- Have nested functions but don't allow them to escape: [Modula-3, Pascal](#)
- Keep activation records around even after they have been popped off (if needed): [SML](#)

Keeping activation records around

- Put activation records on the heap and link them together, instead of putting them on the stack



Implementation notes

- Could use smarter techniques to put activation records on the stack and lazily moving them to the heap if needed
 - More on this when we look at continuations

Closures and objects

- From a distance, **closures and objects are very similar**
 - A closure is a single function along with any non-local data that it needs
 - An object is a suite of functions along with any non-local data that they need
- Indeed, in languages such as C++, people often implement closures with objects

Example

```
void do_something(i, l) {  
    int n;  
    int incn(int elem) {  
        return elem + n;  
    }  
    n = i;  
    map(l, incn);  
}
```

```
class incn_class {  
    int n;  
    incn_class(int nvalue) {  
        n = nvalue;  
    }  
    int apply(int elem) {  
        return n + elem;  
    }  
};  
void do_something(i, l) {  
    map(l, new incn_class(i));  
}
```

Discussions

- Are closures a good idea?
 - Allows one to treat functions as first-class values
 - Having only top-level functions is too limiting
- Are closures a bad idea?
 - Implementation cost may be high

Next topic: Exception handling

- Readings: Exception handling in Java, and brief discussion in Modula-3