CSE 3302 Notes 3: Names, Scopes, and Bindings

(Last updated 9/25/12 8:21 AM)

3.1. Binding Time

Binding = Commitment: Existence, Type(s), Value, Representation, Location

Language Design

Language/Library Implementation

Program Writing

Compilation

Linkage

Loading

Execution

3.2. Object Lifetime and Storage Management

Big Side Issues

- Recursion
- Reentrant Code
- Threads/Processes
- Separate Code and Data Address Spaces
- Virtual Memory, Caches, and Mappings

Static Allocation

- Characteristics - single instance, fixed size, global or side-effect (non-pure-functional)
- Pointers?

Stack-Based Allocation (fig 3.1)

Support of recursion and functions in general

Size of stack frame (activation record) and offsets for function are usually known at compile-time

Nested blocks? (or `alloca()`)

Heap-Based Allocation

Most flexible "temporally"

Pascal implementation - same space as stack

Buddy systems - maintain available blocks of size 2^k

One or many free lists?

Fragmentation

External - unallocated space between allocated blocks

Internal - extra space inside allocated block

```
#include <stdio.h>
#include <stdlib.h>

char bigStatic[2000000000];

main()
{
char bigStack[10000000];

char *bigHeap;

printf("Ready to malloc\n");
bigHeap=(char*) malloc(10000000);
printf("malloc successful\n");
}
```

Garbage Collection

Explicit freeing of unneeded space or reachability checking (or reference counts)?

Free lists only or *compact* active memory to remove external fragmentation?

3.3. SCOPE RULES

When is a particular binding of name to . . . relevant?

Static Scoping

Globals

Variables within functions

Nested blocks

Existence independent of execution:

C `static` refers to allocation

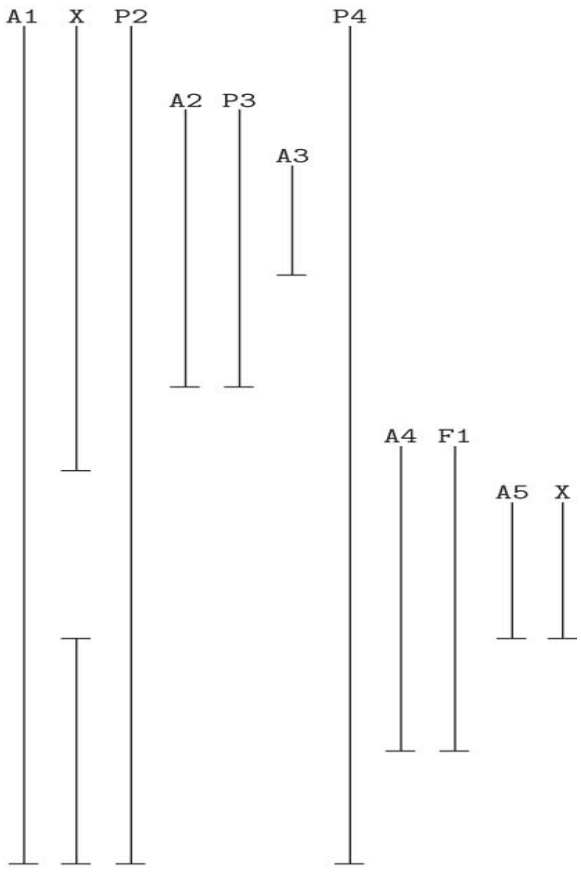
Class variables

Nested Subroutines

Non-nesting: A significant connection between C to COBOL, FORTRAN, and assembler.

Pascal:

```
procedure P1(A1 : T1);
var X : real;
    ...
    procedure P2(A2 : T2);
        ...
        procedure P3(A3 : T3);
        ...
        begin
            ...      (* body of P3 *)
        end;
        ...
    begin
        ...          (* body of P2 *)
    end;
    ...
    procedure P4(A4 : T4);
        ...
        function F1(A5 : T5) : T6;
        var X : integer;
        ...
        begin
            ...      (* body of F1 *)
        end;
        ...
    begin
        ...          (* body of P4 *)
    end;
    ...
begin
    ...              (* body of P1 *)
end
```

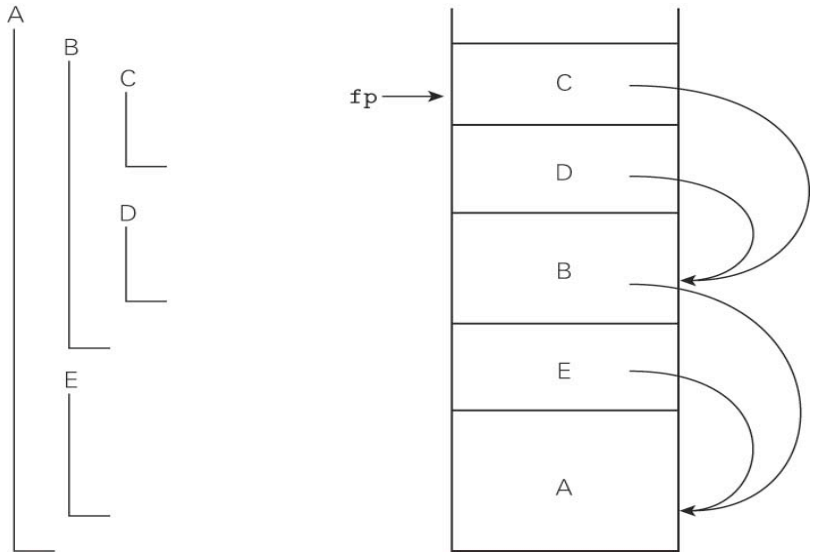


The issue of holes in a scope can be addressed with *scope resolution operators*.

Pascal also allows `forward` declarations to have mutual recursion without nesting or to deal with complicated situations like the Pascal-S interpreter:

1. `pascal-s.structure.txt` provides nesting structure (see `expression`).
2. `syntax6.gif` provides the call graph.
3. `pascals.txt` gives complete code.
4. `pascals.dot` is the call graph as input to Graphviz.

Important detail - for such code (potentially with a variety of call paths and recursion), how are necessary bindings referenced?



Declaration Order

Pascal - Scope of declaration is entire surrounding block. Can't use until declared. (p. 128)

C . . . - Scope of declaration begins with the declaration, but definition may appear later.

Modules - hold for

Chapter 9 - Data Abstraction and Object Orientation

ML modules

Dynamic Scoping

Name operates LIFO as contexts are entered and exited. (figure 3.9)

3.5. Meaning of Names within a Scope

Aliases

```
x^=y;
y^=x;
x^=y;
```

(tagged) unions

Overloading

Arithmetic operations applying to multiple simple types

C++ - use `[]` to treat binary search tree as array

Polymorphism

`poly.c` - *type signature* to determine which of identically named functions gets called

3.6. Binding Time of Referencing Environments

Subroutine Closures

Shallow Binding - ignore (goes along with dynamic scopring, e.g. Perl)

Deep Binding

Static/lexical binding

Static chain pointer created *at same time as* reference to function (*closure*)

Problem when reference persists longer than stack frame

Figure 3.15 - static chain link is created when B is put in argument list

Solution - anything needed for closure gets heap allocation

Like object systems

Like functional languages

JavaScript (`closure.html`)

```
var makeCounter = function(initVal) {
  var privateCounter;
  var funcs= {
    reset: function() {
      privateCounter=initVal;
    },
    up: function(val) {
      privateCounter+=val;
    },
    down: function(val) {
      privateCounter-=val;
    },
    value: function() {
      return privateCounter;
    }
  };
  funcs.reset();
  return funcs;
};
```

3.7. MACRO EXPANSION

The first software tool?

OS generation

I/O routines, record layouts

Part of C/C++ preprocessor

```
/* Basic bit twiddles */
#define bitSelect(x,bit) ((x>>(bit))&1)
#define bitSet(x,bit) (x | (1<<(bit)))
#define bitClear(x,bit) (x & ~(1<<(bit)))
#define bitComplement(x,bit) (x ^ (1<<(bit)))
```

Continuing a macro definition?

Inlining

Loop unrolling?