Today's Topics

Last Time
- Semantics - the meaning of program structures
- Stack model of expression evaluation, the Expression Stack (ES)
- Stack model of automatic storage, the Run Stack (RS)

Today
- Managing the Run Stack, the Dynamic Pointer Stack
- Modelling visibility - Run Stack Display
- Lexical Level / Order Number addressing
Expression Stack (ES) Evaluation

\[ x := 1 + 2 \]

- push 1
- push 2
- add
- pop x

\[ x := (5 + 7) \times 9 \]

- push 5
- push 7
- add
- push 9
- multiply
- pop x
The Run Stack – Pascal Example

```pascal
var x: integer;

procedure p;
  var y: integer;
  z: real;
end p;

p;
```

RS
Managing the Run Stack

• We need to know how much to remove from the Run Stack when we exit each scope

• Scopes are nested, so this suggests another stack, the Dynamic Pointer Stack (DPS)

• The DPS has one entry for each scope that we enter, popped when we leave a scope

• The entry points to the bottom of the space allocated on the Run Stack for the scope

• When we leave a scope, use the top entry on the DPS to pop the RS before removing the element from the DPS
The Dynamic Pointer Stack - Example

```pascal
var x: integer;
procedure p;
    var y: integer;
    z: integer;
end p;
procedure q;
    var w: integer;
    u: integer;
p;
end q;
q;
```

RS

DPS

RS

global scope

DPS
Variable Visibility

- The compiler must implement variable visibility - some variables are still "live" (in memory), but are not currently accessible.

- At the execution point shown in our example, procedure $p$ must not be able to access $w$ and $u$, but must be able to access $x$.

```
var x: integer;
procedure p;
  var y : integer;
  z : integer;
end p;
procedure q;
  var w : integer;
  u : integer;
p;
end q;
qu;
```
There may also be more than one variable with the same name on the RS, due to either

- **Recursion** (multiple instances of the same variable), or

- **Masking** (variable with the same name in an inner scope).
Variable Visibility

• Our Run Stack model of scopes and variables must account for both these problems, so that the compiler properly implements the language

• The points can be summarized as:
  • Given a variable name,
    • Is a variable of that name currently visible?
    • If so, which one of that name is currently accessible?
    • And where is that one on the RS?
  • Or more succinctly, what is the RS position (memory address) of the currently visible variable of a given name?
  • We will attack this problem by refining the RS model to include the concept of lexical levels
Lexical Levels

- A *lexical level* is a level number assigned to each scope based on its static nesting depth (the depth of the scope in the actual source text of the program).

```plaintext
0
  var x: integer;
  procedure p;
  int x;
  void main() {

1
  var y: integer;
  z: integer;
  procedure q;
  int y,
  z;
  {

2
  var w: integer;
  end q;
  int w;
  procedure r;
  int u;
  var u: integer;
  end r;
  end r;
  {
```
Lexical Level, Order Number Addressing

• Each variable in a scope is given an order number (ON) which is the sequence number of its declaration in the scope - the first declared variable in a scope is number 0, the second is 1, and so on

• The ordered pair (LL,ON), consisting of a lexical level LL and an order number ON, refers to "the currently visible variable with order number ON at lexical level LL" - uniquely identifies each visible variable

• Referring to variables in this way is called Lexical Level, Order Number Addressing
Lexical Level, Order Number Addressing

- In our example:

  - \( x \) (0,0)
  - \( y \) (1,0)
  - \( z \) (1,1)
  - \( w \) (2,0)
  - \( u \) (2,0) Note both \( w \) and \( u \) same!

- In our abstract machine model, all variables are accessed through the their \((\text{LL,ON})\) address

- We will arrange that at any given time, each \((\text{LL,ON})\) address refers to only one variable

- If two variables have the same \((\text{LL,ON})\) pair, then at any given time in execution, the model will arrange that the currently visible one is the one referenced by \((\text{LL,ON})\)
Lexical Level Representation

- Lexical Levels are modeled in the abstract machine using another stack-like structure called the *Display*.
- The *Display* keeps track of the currently active scope at each LL.
- For (LL,ON) addressing to work, we must maintain (i.e., keep current during execution) the *Display*.
- On entry to a scope at lexical level LL, we must set Display[LL] to point to the base of the new scope's RS space (its "stack frame").
- On exit from a scope at lexical level LL, we must reset Display[LL] to point to the base of the previous scope that was at that level.

```
global
procedure p
procedure r
```

```
0 1 2
x y z
```

```
Display          RS            DPS
```

procedure r
procedure p
global
Lexical Level Representation

• Example: on entry to \( q \), must set \text{Display}[2] \text{ to the } \text{RS space for } q

• On exit from \( q \), must reset \text{Display}[2] \text{ to the } \text{RS space for } r
(LL,ON) Address Calculation

- At run time, the address of a variable in the Run Stack is computed from its (LL,ON)

- RS [Display [LL] + ON] is the storage for variable (LL,ON)

- Display [LL] is the base of the storage for the scope, and ON is the displacement of the particular variable within that storage
Addressable Variables

• In our abstract machine model, all variables are referred to using (LL,ON) addressing

• Example: for expression $z+x$
  
  push (1,1)  
  push value of $RS[Display[1] + 1]$ on ES  
  push (0,0)  
  push value of $RS[Display[0] + 0]$ on ES  
  add

• Variables not in visible scopes are not addressable on the RS (i.e., there is no (LL,ON) way to refer to them)

  • If they are in internal scopes (i.e. a higher lexical level) then there is not yet an entry for their LL in the Display

  • If they are in sibling scopes (at the same or lower lexical level) then their (LL,ON) address is masked by the addresses of the variables in the current scope

  • This also handles the problem of which instance of a variable is visible when we have recursive procedures
Summary

Modelling Scopes and Visibility

- Managing the Run Stack, the Dynamic Pointer Stack
- The Run Stack Display
- (LL,ON) addressing

Next Time

- Maintaining the Display
- Modelling procedure call / return, parameter passing, functions