15 Static and Dynamic Typing and Scope

Friday, November 1, 2024 12:42 PM

"Static" :- as in the text.
"Dynamic" :- as the program runs.

- TYPE BINDING:

  ¿ When does a variable gets it's type?

  - Static type Binding

    type stated somewhere in the text

    - Expicit Statement

      C++ int x; string s;

      Pascal
      ```
      VAR
        X : INTEGER;
        S : STRING;
      ```

    - Implicit Declaration

      FORTRAN I J K are integes
      everything else is float
      (unless otherwise specified)

      PERL
      name carries type.
      @x array
      %x hashtable / Dictionary
      $x scalar

    - Static Type Prediction

      C++ auto

      type is deduced from initialization.

      Go VAR float x

Go

```
VAR float x
  ⋮
x = 3.14
```

```
VAR x := 3.14
```
← use type deduction

- Dynamic type Binding
  - type is bound when a value is assigned as the program runs
  - Type can change, by another assignment.

    Python, JavaScript, Ruby, Lua

- SCOPE:
  - The scope of a variable, is the range of statements over which the variable is visible.
    - Global

      BASIC
    - Static Scope:

      range is determined by program text

      C / C++

      scope is based on "blocks"

      { block }

      e.g

```
int z;  ← Global

int foo ()
{
   int z;  ← local variable masks global variable.
   ⋮
   {
      int z;  ← inner scope masks outer scope.
      ≡
      z = 3 + 7;
      ⋮
   }
}
```

Pascal scope is Modul or Function based.

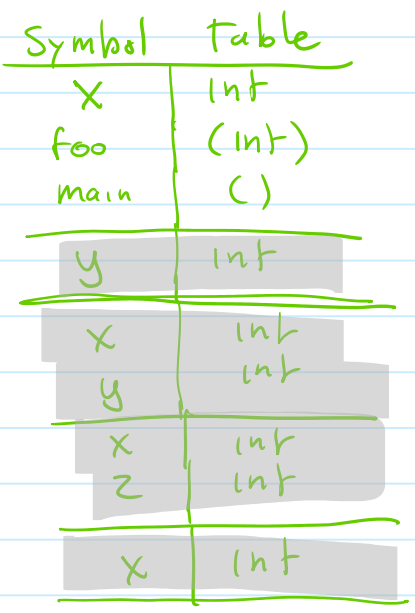
```
VAR
   z: INTEGER;

FUNCTION foo( ): INTEGER
VAR
   z: INTEGER
BEGIN
   ≡
   z := 3+7;
   ≡
END.
```

} the scope of z is the whole function.

- How does the symbol table handles nested scopes
  - Spit symbol table into "frames"
  - As program is parsed:
    - new scope :- push a new frame
    - end scope :- pop a frame
  - Symbol table is a stack.

e.g. C++

```
int x;
void foo ( int y )
{
  x = y * 10;
}

int main()
{
  int x, y;
  cin >> x >> y;
  if ( x<y ) {
    int x,z;
    cin >> z;
    x = y + z;
  }
  for( int x=0; x<y; x++) {
    cout << x;
  }
  cout << x;
}
```



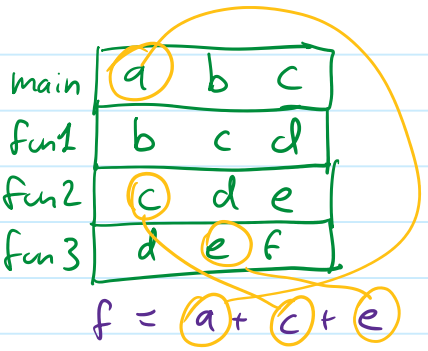
- Dynamic Scope.
  - used in some esoteric programming Languages
    Lisp, Bash
  - '70s important among the lisp community

- ¿ When is a variable visible? Depends on program execution!
  - if fun1() calls fun2()
    then the variables of fun1() are available in fun2()

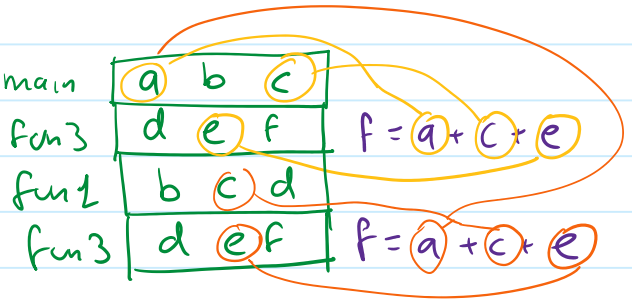
E.G. Imagine C++ w/ Dynamic Scope.

```
void main ()
{
  int a, b, c
  … …
}

void fun1 ()
{
  int b, c, d
  … …
  b = a + 1
}

void fun2 ()
{
  int c, d, e
  … …
}

void fun3 ()
{
  int d, e, f
  … …
  f = a + c + e
}
```

Suppose
main()
└ fun1
  └ fun2
    └ fun3



Suppose
main
  fun3
    fun1
      fun3



EOF