15 Dynamic and Static Typing and Scope

Monday, October 30, 2023 11:06 AM

- STATIC vs DYNAMIC TYPING
  - When does a variable gets its type?
  - Static type Binding

    Type is bound somewhere in the text
    - Explicit statement.

      C++ "Static" .. as in the text
      "Dynamic" .- as the program runs

      int x; string s;

      Pascal VAR

      ```
      x : INTEGER;
      s : STRING;
      ```
    - Implicit declaration.

      Fortran: I J K are integers
      other names real, unless otherwise specified.

      Perl:

      @X array

      %X hashtable / Dictionary

      $X scalar
    - Type deduction.

      Type is deduced by initialization.

      Go

      VAR float x:
      ≡
      x = 3.14

      VAR x := 3.14
      { apply type deduction.
  - Dynamic type Binding

    - type is bound when a value is assigned, as the program runs.
    - type can change on another assignment.

      Python JavaScript Ruby Lua Lisp.

      ```
      def foo(x):
          return x*2

      print( foo(8) )
      foo = 3.14
      ≡
      foo = "hello"
      ```

      foo = foo(3)
- SCOPE
  - The scope of a variable is the range of statements over which the variable is visible.
  - Static Scope
    - determined by program text.

- Static Scope
  - determined by program text.

C++
scope is based on a "block"
{ block }

```
int z;
int foo()
{
  int z;
  ....
  {
    int z;
    =
  }
}
```

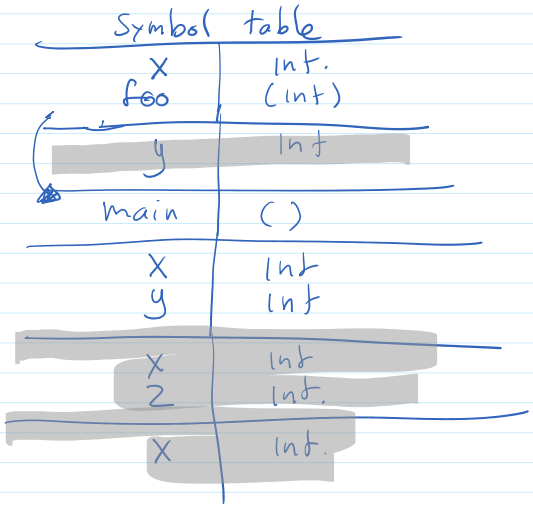
Pascal Scope is Module or Function based.

```
FUNCTION foo():INTEGER
  VAR
    x: INTEGER;
  BEGIN
    x:= 3.14;
    Writeln(x * z);
  END.
```

- Symbol table and Nested Scopes.
  - Split symbol table into "frames"
  - new scope → push a new frame
  - end scope → pop a frame.
  - Symbol table is now 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;
  }
```



```
    for( int x=0; x<y; x++) {
      cout << x;
    }
    cout << x;
  }
```

- Dynamic Scope.

  (used in some esoteric interpreted languages)
  '70s - important among the LISP community.

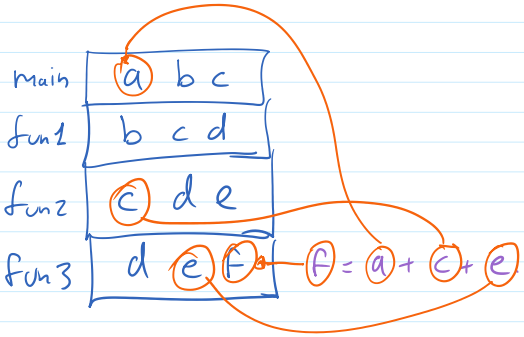
  - Depends on program execution.
  - If you have a function call fun1() → fun2()
    the variables from fun1 are
    visible in the code of fun2.

E.G.

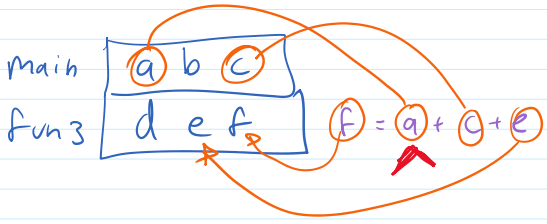
Imagine C++ w/ Dynamic Scope

```
int 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



Main
↳ fun3



EOF