15 Static and Dynamic Typing and Scope

Friday, April 12, 2024 12:00 PM

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

TYPE BINDING

¿when does a variable gets its type?

- Static type Binding

  Type is bound somewhere in the text.

  - Explicit statement

    C++ `int x; string s;`

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

  - Implicit Declaration

    FORTRAN: I J K are intigers
    every thing else is a float.
    (unless otherwise specified)

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

  - Static Type Deduction

    Type is deduced by initialization.

    Go
    ```
    VAR float x          VAR x := 3.14
    :                          ↑
    x = 3.14                   apply type deduction.
    ```

- Dynamic type Binding

  - Type is bound when a value is assigned, as the program runs.

- 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 variables mask global variables
  :
  {
    int z;      <-- inner scope masks outer scope.
    ≡
  }
```

Pascal

Scope is Module or Function based.

```
VAR z : INTEGER

FUNCTION foo():INTEGER;
VAR z : INTEGER
BEGIN
  ≡
END.
```

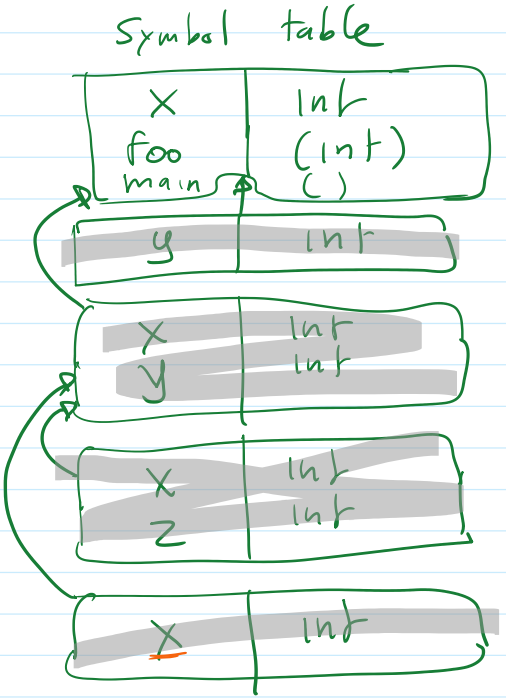
Declarations allowed here only.
The scope of z is the whole function.

- How Does the symbol table handles nested scopes?

- How Does the symbol table handles 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;
  }
  cout << x;
}
```



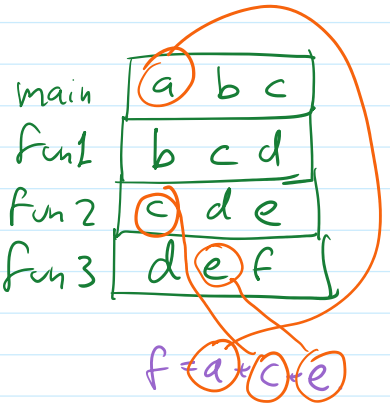
- Dynamic Scope.
  - Used in some esoteric programming Languages.
  - '70 :- important among the LISP community
- When is a variable visible? - Depends on program execution
  - If you have a function call: fun1() calls fun2() then the variables from fun1() are available in fun2().

E.G.

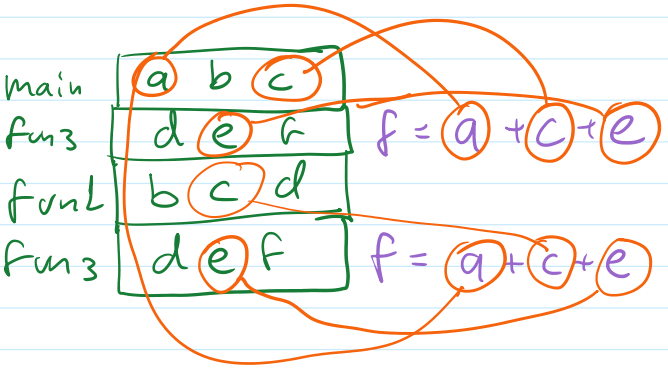
Imagine C++ w/ Dynamic Scope

```
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



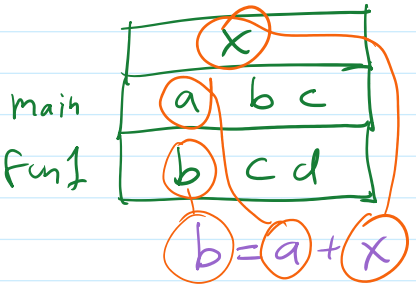
E.6 #2

"C++" w/ Dynamic scope.

```
int x

void main ()
{
  int a, b, c
  … …
}

void fun1 ()
{
  int b, c, d
  … …
  b = a + x
```

Suppose
main
↳ fun1

call stack/symbol table.



}

—o—o— Eof.