CSCI-2320
Principles of Programming Languages

Names
Ref: Tucker-Noonan [Ch. 4]

Syntactic Issues

• Lexical rules for names
• Collection of reserved words or keywords
• Case sensitivity
  • C-like: yes
  • Early languages: no
  • PHP: partly yes, partly no
Binding

L-value vs. R-value

L-value - use of a variable name to denote its address.
   Example:  \texttt{x = ...}

R-value - use of a variable name to denote its value.
   Example:  \texttt{... = ... x ...}
#include <stdio.h>

int main() {
    int x = 50, y = 10;
    int *p = &y;
    *p = x; // r-value of x is assigned to r-value of p
    x = *p + 100; // r-value of r-value of p + 100 is assigned to l-value of x
    printf("p = %p, *p = %d\nx = %d, y = %d\n", p, *p, x, y);
    p = &x;
    printf("After changing pointer: np = %p, *p = %d\nx = %d, y = %d\n", p, *p, x, y);
    return 0;
}
A reference to a name is nonlocal if it occurs in a nested scope of the scope where it's defined; otherwise it's local.

```java
1 void sort (float a[], int size) {
2    int i, j;
3    for (i = 0; i < size; i++)  // i, size local
4        for (j = i + 1; j < size; j++) //i,j, size local
5            if (a[j] < a[i]) {
6                float t;
7                t = a[i];  // t local; a, i nonlocal (note the braces)
8                a[i] = a[j];
9                a[j] = t;
10            }
11    }
12 }
```

Another example of scope

```java
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

```java
i = 0; // invalid reference to i
```
Design Issues

- Dynamic or static?
- Allow forward reference?
  - Using an identifier before its declaration

---

```
#include <stdio.h>

int main() {
    printf("%d\n", first(2));
    return 0;
}

int first(int x) {
    if (x == 0)
        return 1;
    else
        return second(x-1); // forward reference to second
}

int second(int x) {
    if (x == 0)
        return 0;
    else
        return first(x-1); // backward reference to first
}
```

Fine with C11, but not with C99. C99: Need “forward declarations” like int first(int);
Implementation Issue

• How to implement scoping and name resolution?

Implementation: Symbol Table
Symbol Table

- *A symbol table* is a data structure to keep track of each declared name and its bindings.
- Any implementation of a dictionary, where the name is the key.
  - *Scopes can be nested => stack of dictionaries*

```c
void sort (float a[ ], int size) {
    int i, j;
    for (i = 0; i < size; i++)
        for (j = i + 1; j < size; j++)
            if (a[j] < a[i]) {
                float t;
                t = a[i];
                a[i] = a[j];
                a[j] = t;
            }
}
```

Stack of dictionaries

At line 7:
- `<t, 6>`
- `<j, 2>, <i, 2>, <size, 1>, <a, 1>`
- `<sort, 1>`

At line 4 and 11:
- `<j, 2> <i, 2> <size,1> <a, 1>`
- `<sort, 1>`

Dictionaries (each line above is a dictionary)
Key: variable/function name
Value: line number of declaration (for simplicity)
Rules for resolving name reference

Data structure: a stack of dictionaries

• Enter scope: push a new dictionary onto the stack.
• Exit scope: pop the top dictionary off the stack.
• For each newly declared name, generate a binding and enter the name-binding pair into the top dictionary of the stack.
• Given a name reference, search the dictionary on top of the stack:
  – If found, return the binding.
  – Otherwise, repeat the process on the next dictionary down in the stack.
  – If the name is not found in any dictionary, report an error.

Resolving References: Static Scoping

Each function has its own symbol table
1 int h, i;
2 void B(int w) {
3     int j, k;
4     i = 2*w;
5     w = w+1;
6     ...
7 }
8 void A (int x, int y) {
9     float i, j;
10    B(h);
11    i = 3;
12    ...
13 }
14 void main() {
15     int a, b;
16     h = 5; a = 3; b = 2;
17     A(a, b);
18     B(h);
19     ...
20 }

1. Outer scope: <h, 1> <i, 1> <B, 2> <A, 8> <main, 14>
2. Function B: <w, 2> <j, 3> <k, 4> <h, 1> <i, 1> <B, 2> <A, 8> <main, 14>
3. Function A: <x, 8> <y, 8> <i, 9> <j, 9> <h, 1> <i, 1> <B, 2> <A, 8> <main, 14>
4. Function main: <a, 15> <b, 15> <h, 1> <i, 1> <B, 2> <A, 8> <main, 14>

Line | Reference | Declaration
--- | --- | ---
4 | i | 1
10 | h | 1
11 | i | 9
16 | h | 1
18 | h | 1

1 int h, i;
2 void B(int w) {
3     int j, k;
4     i = 2*w;
5     w = w+1;
6     ...
7 }
8 void A (int x, int y) {
9     float i, j;
10    B(h);
11    i = 3;
12    ...
13 }
Dynamic Scoping

One symbol table for the whole program

```
int h, i;
void B(int w) {
    int j, k;
    i = 2*w;
    w = w+1;
    ... 
}
void A(int x, int y) {
    float i, j;
    B(h);
    i = 3;
    ... 
}
void main() {
    int a, b;
    h = 5; a = 3; b = 2;
    A(a, b);
    B(h);
    ... 
}
```

Call history: main (17) → A (10) → B

Reference to i (4) resolves to <i, 9> in A.
1 int h, i;
2 void B(int w) {
3   int j, k;
4   i = 2*w;
5   w = w+1;
6   ...
7 }
8 void A (int x, int y) {
9   float i, j;
10  B(h);
11  i = 3;
12  ...
13 }

14 void main() {
15   int a, b;
16   h = 5; a = 3; b = 2;
17   A(a, b);
18   B(h);
19   ...
20 }

Call history: main (18) → B
B: <w, 2> <j, 3> <k, 3>
main: <a, 15> <b, 15>
     <h, 1> <i, 1> <B, 2> <A, 8> <main, 14>
Reference to i (4) resolves to <i, 1> in global scope.

Another example: swapping C program (static scoping)

#include <stdio.h>
void myFunction();
void swap();

int x = 5, y = 10;

void myFunction()
{
  int x = 100, y = 200;
  swap(); //local var x & y will not be modified
  printf("After swap: x = %d, y = %d\n", x, y);
}

void swap()
{
  int t = x;
  x = y;
  y = t;
}

int main()
{
  myFunction();
  return 0;
}

After swap: x = 100, y = 200
Swapping
Perl program (dynamic scoping)

```perl
$x = 5;
y = 10;
myFunction(); # call subroutine - no argument

sub myFunction
{
    local($x, $y);
x = 100, $y = 200;
    swap(); # no argument
    print "After swap: x = $x, y = $y\n";
}

sub swap
{
    $t = $x; # Is this the x in myFunciton or the global x?
x = $y;
y = $t;
}
```

After swap: x = 200, y = 100

Other issues

- Visibility (re-declaration)
- Overloading
- Lifetime (vs. scope)
  - Variable can be out-of-scope temporarily but can still be living