

CS 460

Programming Languages

Fall 2008

Dr. Watts

(Week 6)



Questions

- Scheme questions?

```
(define sort
  (lambda (L)
    (if (null? L)
        '()
        (combine (sort (half L)) (sort (otherhalf L))))))
```

- Project questions?
- Remember to check submission pages





Names

- Definition: name is a string of characters used to identify some entity in a program
- Issues
 - Length of string
 - Actual length
 - Significant characters
 - Implementation
 - Internal vs External (linked) names
 - Allowable characters
 - _ and -
 - Case sensitivity
 - Camel Notation – eg. quickSort or mergeSort



Names

- Special Words
 - Keyword vs Reserved word
 - Predefined names



Names

- Variables
 - Abstract name for a value stored in a location in computer memory
 - Name
 - Address
 - Type
 - Value
 - r-values vs l-values



Bindings

- Definition: Association between an symbol and its meaning
- When does binding occur?
 - Compiler design time
 - for, int, ;
 - Compile time
 - Meaning of operator + dependent on operand types
 - Execution time
 - Value stored in a variable



Bindings

- Static vs Dynamic
- Static – occurs before run time and remains unchanged during program execution
- Static Type Binding
 - Explicit declaration
 - `int x;`
 - Implicit declaration
 - Some languages permit undeclared variables
 - FORTRAN : I, J, K, L, M, or N – all integers
 - Special symbols
 - BASIC : X – numeric; X\$ - string
 - Perl : \$ - scalar; @ - array; % - hash structure



Bindings

- Dynamic – determined at execution time when a value is assigned to the variable
- Dynamic Type Binding
 - Type of a variable may change during program execution
 - Scripting Languages typically allow both static and dynamic typing
 - Scheme

Binding



- Type inference
 - Type inferred by context
 - C++ : result of `x / y`
 - C++ : `int f ()` vs `int f`
 - Pascal : `const c = 1.5;`
 - Scheme : `(car X)`
 - Type coercion
 - C++ : `int i; float f; ... i = f; f = i;`
 - Pascal : `i : integer; f : real; ... i := f;`

Storage Bindings and Lifetimes



- Static variables
 - Value determined at compile time and remains unchanged during execution
 - C++ : `const PI = 3.14159;`
- Stack-Dynamic variables
 - Global and local declared variables
- Heap-Dynamic variables
 - Explicit – C++ : `int * intArray = new int [size];`
 - Implicit – JavaScript : `values = [1,3,5,7,9];`



Type Checking

- Definition: Activity of ensuring that the operands of an operation are of compatible types
- Compatible types: declared or coerced
- Type error: use of incompatible types
- Static type checking
 - At compile time – C++
- Dynamic type checking
 - At run time – Scheme



Type checking

- Strong Typing
 - 1970's – Pascal, Ada
 - Type errors always detected; generally prevented
 - Pascal: `x, y, z : integer; ...`
`z := x / y;`
`z := x div y;`
- Weak Typing
 - Permits coercion
 - Union types contrary to strong typing
- Migration to stronger typing



Type Equivalence

- Name type equivalence vs structure type equivalence

- Pascal :

```
type recType =  
    record x : integer; y : real; end;  
var a, b : recType;  
    c : record x : integer; y : real; end;  
...  
a := b;      is Name type equivalence  
a := c;      is Structure type equivalence
```



Scopes

- Definition : scope of a variable is the textual range of statements in which the variable can be used.
- Static scoping – scope determined at compile time – ALGOL 60 and descendants
- Global; local; blocks
- Dynamic scoping – scope determined at run time – APL, some versions of LISP, Perl
- Based on sequence of function calls and stack frames

ScopeExample.c



```
#include <stdio.h>

void A (int p);
void C (int p);

int v = 0;

int main ()
{
    int v = 0;
    A (3);
    printf ("in main, v = %d\n", v);
    return 0;
}

void A (int p)
{
    void B (int p)
    {
        int v = 0;
        v++;
        C(p);
        printf ("in B, p = %d and v = %d\n", p, v);
    }

    if (p)
    {
        v++;
        B(p-1);
        printf ("in A, p = %d and v = %d\n", p, v);
    }
}

void C (int p)
{
    A(p);
    printf ("in C, p = %d and v = %d\n", p, v);
}
```

Static Scoping Evaluation



```
in C, p = 0 and v = 3
in B, p = 0 and v = 1
in A, p = 1 and v = 3
in C, p = 1 and v = 3
in B, p = 1 and v = 1
in A, p = 2 and v = 3
in C, p = 2 and v = 3
in B, p = 2 and v = 1
in A, p = 3 and v = 3
in main, v = 0
```

Dynamic Scoping Evaluation



```
in C, p = 0 and v = 1
in B, p = 0 and v = 1
in A, p = 1 and v = 2
in C, p = 1 and v = 2
in B, p = 1 and v = 2
in A, p = 2 and v = 2
in C, p = 2 and v = 2
in B, p = 2 and v = 1
in A, p = 3 and v = 1
in main, v = 0
```

Static vs. Dynamic Scoping



- Static
 - Can be determined from text of program
 - Easier for programmer to predict
- Dynamic
 - Must be determined from run stack
 - Harder to predict
 - Useful for recursive programming