Announcements

- Quiz 3
- HW2 should be graded by Friday/Saturday
  - Regrade requests accepted by next Friday
  - Send regrade requests by email to the list
- Rainbow grades over the weekend
  - HW1 and 2, and Quiz 1 and 2
- No homework coming out tonight, will merge HW4 and HW5 into one homework after test

Exam Topics

- Regular Expressions
- CFGs
  - Derivation, parsing, ambiguity, operator precedence and associativity
  - LL(1) grammars and parsing
    - FIRST and FOLLOW sets, LL(1) parsing table
  - SLR(1) grammars and parsing
    - CFSM and SLR(1) parsing tables
    - Conflicts in SLR(1)

Last Class

- Concluded Prolog
  - Lists
  - Arithmetic
  - Backtracking cut
  - Negation by failure
  - Generate and test paradigm
    \[ \text{solve}(.,S) :- \text{generate}(.,S), \text{test}(S). \]

Names, Scopes, and Binding

Read: Scott, Chapter 3.1, 3.2 and 3.3.1, 3.3.2 and 3.3.6
Today’s Lecture Outline

- Notion of binding time
- Object lifetime and storage management
- Scoping
  - Static scoping
  - Dynamic scoping
- Review for Exam 1

Names and Binding

- Name - character string, which represents a programming language construct (e.g., variable, subroutine)
- Binding - an association of a name to the object it represents
  - Can happen at different times during translation and execution

Notion of Binding Time

- Static
  - Before program execution
- Dynamic
  - During program executes

Examples of Binding Time Decisions

- Binding a variable name to memory location
  - Static or Dynamic
  - Determined by scoping rules
- Binding a variable/expression to type
  - Static or Dynamic
- Binding of a call to a target subroutine
  - Static (as it is in C, mostly)
  - Dynamic (virtual calls in Java, C++)

Binding Time and Programming Language Design Choices

- Many PL design choices relate to binding time
- Static binding (also referred to as compile-time binding, early binding)
  - More efficient
  - Less flexible
- Dynamic binding (also referred to as run-time binding, late binding)
  - Less efficient
  - More flexible

Example: Binding Variables to Locations

```c
int x, y;
void foo(int x) {  
  y = x;
  int y = 0;
  if (y) {
    int y;  
    y = 1;
  }
}
```
General View of Binding Time

- Binding time (Scott): the time an answer becomes associated to an open question
- Examples of bindings and binding time
  - Binding of variable to values
    - Dynamic
  - Binding of call to target
    - Static (in C: a call f(x)())
    - Dynamic (in Java: a call a.m())
  - Binding of variable to type
    - Static (in Java, C++: declared types for variables)
    - Dynamic (in Smalltalk, Python: actual run-time types of objects)

General View of Dynamic Binding

- Dynamic binding
  - What are the advantages of dynamic binding?
  - Disadvantages?
- An example: Cost of dynamic binding of call to target method in OO languages

Example: Cost of Dynamic Dispatch in C++

Source: Driesen and Hölzle, OOPSLA’96

Virtual function tables (VFTs)
Capital characters denote classes, lowercase characters message selectors, and numbers method addresses.

Extra instructions: cost extra!

Other Choices Related to Binding Time

- Pointers: introduce "heap variables"
  - Good for flexibility – allows dynamic structures
  - Bad for efficiency – directly cost: accessed indirectly; indirect cost: compiler unable to perform optimizations!
- Most PLs support pointers
  - Issues of management of heap memory
    - Explicit allocation and deallocation
    - Implicit deallocation (garbage collection)
- PL design choices – many subtle variations
  - No pointers (FORTRAN 77)
  - Explicit pointers (C++ and C)
  - Implicit pointers (Java)

Lecture Outline

- Notion of binding time
- Object lifetime and storage management
- Scoping
  - Static Scoping
  - Dynamic Scoping
- Review for Exam 1

Storage Allocation Mechanisms

- Again, note on use of term object: anything that can have a name (typically, variable location)
  - Static storage – an object is given absolute address, which is the same throughout execution
    - What is an example of static data?
  - Stack storage – stack objects are allocated on a run-time stack at subroutine call and deallocated at return
    - Needs a stack management algorithm
    - What is an example of stack data?
- Heap storage - long-lived objects are allocated and deallocated at arbitrary times during execution
  - Needs the most complex storage management algorithm
Combined View of Storage

- Stack contains 1 activation record (or stack frame) per executing subroutine.
- Heap contains objects allocated and not yet de-allocated.

Examples of Static Data

- Static variables
- Program code
- Tables of type data (e.g., inheritance structure)
- Dispatch tables (VFTs) and other tables
- Other

Examples of Stack Data

- What data is stored on the stack?
- Parameters
- Local variables
- Compiler-generated temporaries (i.e., for expression evaluation)
- Bookkeeping (stack management) information

Run-time Stack

- Stack contains frames of all subroutines which have been entered and not yet exited from
- Frame contains all information necessary to update stack when a subroutine is exited
- Addresses for local variables are encoded as fp (frame pointer) + offset
- Idea: When subroutine is entered, its frame is placed on the stack and sp (the stack pointer --- i.e., next available location on stack), and fp (the frame pointer --- i.e., the current frame pointer) are updated. All accesses for local variables use this frame. When subroutine exits, its frame is removed from the stack and sp and fp updated.

Frame Details

- Fixed length portion: same length for every subroutine
  - Return address (to code within caller)
  - Miscellaneous bookkeeping information
    - called-by link: pointer to beginning of caller’s frame
    - saved state (register values of caller)
    - link for accessing non-local variables
  - Variable length portion: length varies by subroutine
    - local variable storage (including parameters)
    - compiler-generated temporary storage for subexpressions

Frame Example

```c
void foo(double rate, double initial) {
    double position = initial + rate*60.0; ...
    return;
}
```
Assume `foo()` is called by `bar()`.

Frame of `foo()`

<table>
<thead>
<tr>
<th>Temp</th>
<th>Locals</th>
<th>miscinfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>temp1</td>
<td>position</td>
<td>info</td>
</tr>
<tr>
<td>temp2</td>
<td>initial</td>
<td>info</td>
</tr>
<tr>
<td></td>
<td>rate</td>
<td>info</td>
</tr>
<tr>
<td></td>
<td>called-by info</td>
<td>info</td>
</tr>
<tr>
<td></td>
<td>return address</td>
<td>info</td>
</tr>
<tr>
<td></td>
<td>Return address in code of caller</td>
<td>info</td>
</tr>
</tbody>
</table>
Question

- `sp` (stack pointer) points to the next available location on stack, right after current frame `Q`
- `fp` (frame pointer) points to start of current frame `Q`

- When `Q`'s routine exits (`Q` is popped off the stack), what are the new values of `sp` and `fp`?
  - `sp` (new) = `fp`
  - `fp` (new) gets its value from the called-by link of `Q`

Lecture Outline

- Notion of binding time
- Object lifetime and storage management
- Scoping
  - Static Scoping
  - Dynamic Scoping
- Review for Exam 1

Scoping

- In most languages a variable name can be used multiple times
- **Scoping rules**: map variable names to declarations
- **Scope**: region of program text where declaration is visible (region of program where a binding is active)
- Most languages use **static scoping**
  - Mapping from uses to declaration is made at compile time
- Block-structured programming languages
  - Nested subroutines (Pascal, ML, Scheme, etc.)
  - Nested blocks (C, C++, etc.)

Static Scoping in Block Structured Programming Languages

- Also known as lexical scoping
- Block structure and nesting of blocks gives rise to the closest nested scope rule
  - There are local variable declaration within a block
  - A block inherits variable declarations from enclosing blocks
  - Local declarations take precedence over inherited ones
    - Hole in scope of inherited declaration
    - In other words, inherited declaration is hidden
  - Lookup for non-local variables proceeds from inner to outer enclosing blocks

Example - Block Structured PL

```
program a, b, c: integer;
procedure P();
    c: integer;
    procedure S();
    c, d: integer;
        procedure R();
            //end R
        end S
    end S
R();
S();
end P

procedure R(){
a: integer;
= a, b, c;
} //end R
end R
...
; P(); ...
end program
```

Rule: a variable is visible if it is declared in its own block or in a textually surrounding block and is not "hidden" by a binding to it in a closer block (i.e., hole in scope)
Example with Frames

```c
main()
    a, b, c: integer;/*1*/
    procedure P() {/*3*/
        c: integer;
        procedure S() {/*8*/
            c, d: integer;
            procedure R() {/*10*/
                ...
            } //end R;/*11*/
        } //end S;/*12*/
    } //end P;/*13*/
    procedure R(){ /*5*/
        a: integer;
        = a, b, c;
    } //end R; /*6*/
    P();/*2*/
    ...
} //end main /*14*/
```

Example

```c
main()
    a, b, c: integer;/*1*/
    procedure P() {/*3*/
        c: integer;
        procedure S() {/*8*/
            c, d: integer;
            procedure R() {/*10*/
                ...
            } //end R;/*11*/
        } //end S;/*12*/
    } //end P;/*13*/
    procedure R(){ /*5*/
        a: integer;
        = a, b, c;
    } //end R; /*6*/
    P();/*2*/
    ...
} //end main /*14*/
```
Example

```c
main{
    a, b, c: integer;
    procedure P(){
        c: integer;
        procedure S(){
            c, d: integer;
            procedure R(){
                ...
            } // end R;
            R();
        } // end S;
    } // end P;
    procedure R(){
        a: integer;
        = a, b, c;
    } // end R;  
    ... P();
} // end main
```

Observations

- The static link of a procedure's frame always points to a frame of the same procedure, no matter where the procedure is called from.
- Used to implement static scoping using a display.
- The dynamic link may point to a different procedure's frame, depending on the procedure is called from!

Dynamic Scoping

- Allows for local variable declaration
- Inherit non-local variables from subroutines which are live when current subroutine is invoked.
- Use of variable is resolved to the declaration of that variable in the most recently invoked and not yet terminated frame. I.e., lookup proceeds from closest predecessor on stack to furthest.
- (old) Lisp, APL, Snobol, Perl

Example

```c
main{
    procedure Z(){
        a: integer;
        = a = 1;
        Y();
        output a;
    } // end Z;
    procedure W(){
        a: integer;
        = a = 2;
        Y();
        output a;
    } // end W;
    procedure Y(){
        a = 0;  
    } // end Y;
    Z();
    W();
} // end main
```
Example

main calls Z, Z calls Y, Y sets Z.a to 0.

main{
    procedure Z()
    a: integer;
    a := 1;
    Y();
    output a;
    }//end Z;

    procedure W()
    a: integer;
    a := 2;
    Y();
    output a;
    }//end W;

    procedure Y()
    a := 0; /*1*/
    }//end Y;

    Z();
    W();
} //end main

Example

main calls W, W calls Y, Y sets W.a to 0.

main{
    procedure Z()
    a: integer;
    a := 1;
    Y();
    output a;
    } //end Z;

    procedure W()
    a: integer;
    a := 2;
    Y();
    output a;
    } //end W;

    procedure Y()
    a := 0; /*1*/
    } //end Y;

    Z();
    W();
} //end main

Static vs. Dynamic Scoping

Static Scoping:
- a bound to R.a,
- b to main.b,
- c to main.c

Dynamic Scoping:
- a bound to R.a,
- b to main.b,
- c to P.c

Dynamic Scoping is considered a bad idea. Why?

Exam Review

- Regular Expressions
- CFGs
  - Derivation, parsing, ambiguity, operator precedence and associativity
  - LL(1) grammars and parsing
    - FIRST and FOLLOW sets, LL(1) parsing table
  - SLR(1) grammars and parsing
    - CFST and SLR(1) parsing tables
    - Conflicts in SLR(1)

- Prolog
  - Concepts: Search trees, Unification, Backtracking

  One type of question. Given a Prolog predicate and a query, “Find the first binding” or “Find all bindings”

  Second type of question. Given a description, write a predicate. 3-4 lines of code.
Quiz 1

Question 1. (2pts) Consider the operator grammar

\[
\begin{align*}
\text{expr} &\rightarrow \text{expr} \; \text{id} \; \text{expr} \\
\text{term} &\rightarrow \text{id} \\
\text{id} &\rightarrow \text{id} \; \text{id}
\end{align*}
\]

How many parse trees are there for string id \; id \; id \; id \; id \; id \; id \; id ?

(a) 0
(b) 1
(c) 2

Question 2. (2pts) Consider the following grammar:

\[
\begin{align*}
p(I, \text{[2|S]}) &\rightarrow \text{append}([\text{b}|I1], [\text{a}|I2], I), p(I1, S1), p(I2, S2), \text{append} (S1, S2, S) \\
p(I, \text{[1|S]}) &\rightarrow \text{append}([\text{a}|I1], [\text{b}|I2], I), p(I1, S1), p(I2, S2), \text{append} (S1, S2, S)
\end{align*}
\]

(a) rightmost
(b) leftmost
(c) neither

Fall 18 CSCI 4420, A Milamova

Quiz 2

Question 3. (2pts) There are context-free grammars that generate regular languages.

(a) true
(b) false

Question 4. (2pts) Consider the grammar:

\[
S \rightarrow a \text{NS} \mid b \text{bS} 
\]

A depth-first top-down parser with backtracking produces one of the following when parsing string abab (At each step, expanded nonterminal is underlined.)

(a) \text{S} \rightarrow \text{aNS} \rightarrow \text{abNS} \rightarrow \text{abab} \rightarrow \text{abab}
(b) \text{S} \rightarrow \text{bS} \rightarrow \text{abS} \rightarrow \text{abab} \rightarrow \text{abab}
(c) \text{S} \rightarrow \text{aNS} \rightarrow \text{abNS} \rightarrow \text{abab} \rightarrow \text{abab}

Fall 18 CSCI 4420, A Milamova

Quiz 3

Question 1. (2pts) Draw a parse tree for string if b then a then b

Question 2. (2pts) As we discussed in class, this grammar is not LL(1). Specifically, there is conflict in rule else_part, alias as both else_part \rightarrow else stmt and else_part \rightarrow apply on token else. How can you resolve this conflict, so that an else would associate with the nearest unmatched then?

(a) Always choose else_part \rightarrow else stmt
(b) Always choose else_part \rightarrow apply

Question 3. (2pts) The grammar is SLR(1).

(a) true
(b) false

Question 4. (2pts) Give the closure of LR item start \rightarrow stmt$\$

(a) start \rightarrow stmt$

fall 18 CSCI 4420, A Milamova

Quiz 3

Question 1. (2pts) Consider the classmates Prolog program:

\[
\begin{align*}
takes &\rightarrow \text{takes}([\text{ajit}, \text{art}], X), \text{takes}([\text{ajit}, \text{art}], X) \\
takes &\rightarrow \text{takes}([\text{ajit}, \text{cs}], X), \text{takes}([\text{ajit}, \text{cs}], X) \\
takes &\rightarrow \text{takes}([\text{ajit}, \text{cs}], X), \text{takes}([\text{ajit}, \text{cs}], X) \\
\end{align*}
\]

(a) \text{classmates(X,Y)} \rightarrow \text{takes(X,Z), takes(Y,Z)}

(b) \text{takes(ajit, art)}

(c) \text{takes(ajit, cs)}

(d) \text{takes(ajit, cs)}

Fall 18 CSCI 4420, A Milamova

Quiz 4

Question 1. (2pts) Consider the grammar:

\[
\begin{align*}
gcd &\rightarrow \text{gcd}(A, B, R) \\
gcd &\rightarrow A = B, R = A \\
gcd &\rightarrow \text{gcd}(A, B1, R), B1 = B - A
\end{align*}
\]

(a) \text{gcd}(a, b, c)

(b) \text{gcd}(a, b, c)

(c) \text{gcd}(a, b, c)

(d) \text{gcd}(a, b, c)

Fall 18 CSCI 4420, A Milamova