CS354 Notes

08/22/2018
PL paradigms

Cd ~jbuffenb/classes/354/pub for lecture presentations

Different types of instructions (ARM/INTEL?) CISC/RISC

Pascal has “with” function

Ex.
```
with a > b > c do begin
    v=x;
    v=y;
End
```

Functional language: pass functions into functions

RPG = report program generator

C++ might be the hardest language, if you learn the whole thing

MIT use “Lisp” as their first programming language

PL: C - imperative programs - loops, second assignments to variables

PL: Scheme - functional programs - Single assignment variables and uses recursion.
comments are interperted via comments.
-In the Ex. comments have a interpreter file path to indicate what interpreter to use.
Function examples f(x) == (f x) for Scheme

PL: Prolog - logic program - true/false functions, also called predicates. Tests different parts of the function, testing and checking if conditions (functions, parameters, etcetera) are true or false
Set theory => logic => arithmetic (Prolog is a logic based program with arithmetic, but the arithmetic is not used much in Prolog, at least not recommended…) Can run functions backwards. Example gcd(15,25,G) => gcd(G,25,5), but cannot actually be used because of an inhibitor.

Interpreter vs. Compiler

-Interpreter translates step by step (example debugging), while the compiler translates the entire program package
-Takes up more space b/c the the interpreter itself is taking up space, while the compiler is only needed once.
-Compiler does not test if the program can run or not

Implementation strategies
-preprocessor: removes comments, turns into spaces. Groups characters into tokens (Ex. letters ‘i’ and ‘f’ are ‘if’ which has its own meaning. Even ‘==’)

check out easy to understand translation process on slide “Implementation strategies (2 of 7)”

Preprocessor tells the compiler what line of code it is compiling

-can translate one PL into another PL
Ex. C++ => C

Pcode in the example is an intermediary language that the interpreter can read, like java file to .class file.

JIT (just in time compilation) Ex. Interpreter compile long iterations of code, like a loop. Very faster.

Using java in the shell is running a java virtual machine (JVM)

Idea* build a computer that only runs on java on the machine code level. Much faster. Not so much compilation. Has already been done.

09/05/2018

Homework run-down

-create an assignment turn-in folder

Language assignment 1: In slides/choose scheme slides for homework

Scheme is interchangeable with programs and data
Dynamically typed (like no doubles or ints, that’s static typing)
Statically scoped

__Program Structure__

**refer to slides for function definitions

(3 of 4) has a really good scoping example for scheme.

Languages that use a scanner are regular languages (scanner)
The ones that don't are called context-free languages (parser) method checker
Context-sensitive language is the next hierarchy (CSL) (Semantic Analyzer)
Recursively enumerable languages
Anything outside is not recognizable

Identifier - made up
Built in type - already made up

Semantic Analyzer
Checks over the entire code

Append function not a good function to use for the super-duper program

Cons = construct, makes a new pair in memory.

2-dimensional recursion for scheme.

09/24/2018
Regular-Definition

Context-Free Grammars (CFGs)
-terminal symbols (tokens)
    End of tree
-non-terminal represent a string of terminals
-Start symbol, appears at the root of parse tree
    Non-terminal
-Productions,
nonTerminal (production) terminal
Left   right

```
Ex. Factor -> id
               | terminal
               | terminal
```

Operands (order of operations: Precedence)

(CFG list)

Derivation or tree.

2.17
Add productions, change productions. Might add non-terminals as well

Chapter 3:

Identifier
Name
Object - memory object
Binding - naming memory objects
Scope - within the body of a method
Environment - place named objects in memory are stored

Binding time is complicated, and can occur based on the setup of the machine

Module-entry time: calling a binding from somewhere else
    Public static example
Elaboration time: when the compiler see's it
Statement execution time< for (int i=0) >
Scope rules decide which variable with the same name is pulled from memory
Lifetime of binding is longer than scope most of the time

Static allocation:
Storing in the same location in memory for the entire lifetime of the program
Static or own variables

Stack allocation:
Begins when the method is called, then expires when the method returns

HW SmlTlk
Accrue needs to be an abstract method in account;
Example in hw file.

Have to agree on the stack frame if using different compilers

-3 types of stack allocation
Static, stack, and heap allocation

Cons allocates heap memory, always the same size. No memory fragmentation

External fragmentation is when freed heap storage, for example, is a space between two currently allocated chunks of memory.

Garbage collection can re-stack allocated memory to get rid of checkerboard like stacks

Stack grows down
Heap grows up

Maximal size - i variable always refers to itself within the scope

Holes in scopes, when function call is within function

Static Scope