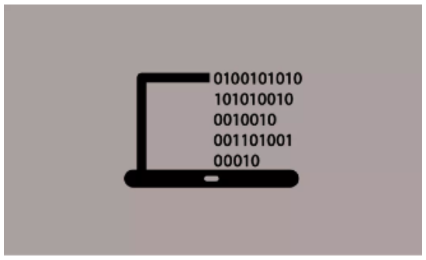
Tuesday, August 28, 2018

CECS 444 Compiler Constructions

Seminar Notes

August 28, 2018



<u>Syllabus</u>

Things to cover:

- Treewalking (binary)

Textbook:

Fisher, Cytron, Leblanc

- Crafting a Compiler (2009 ~720pg)

Grading:

Cumulative Exams

20% Exams I

20% Exams II

33% Final

20% Projects (Will build on each other)

7% Quiz, Paper, Participation

MGR Types: (Manager Types)

Good: 10% - Super people

Bad: 80% - Need people to do the job

- They buy programmers “By the Yard”

Ugly: 10% - Backstab

Mini- SWE (Software Engineering) Rules

** Reasonable Person STD (Standard)

- Due Diligence (Everybody has their own view)

- Pace yourself

Tuesday, August 28, 2018

- AIO: (Adapt, Improvise, and Overcome)

** "Smart" Person STD

- Always be ready to show your work (Show your progress)

★ Most Important Things in SW(Software): **MORALE**

Rules:

0. Get to working Software Fast!

(Go ugly early)

Why!

1. You can see it work

* 2. Users can see it & tell you it sucks

- Get users feedback faster

(MVP = Minimum Viable Product)

1. Never Pre-Optimize (Usually 1% of code is too slow)

- Change this 1% and program increases more in speed

*** Optimize ONLY when proven needed

2. No "BUG HUNTS"

I. Compile-Time Errors $\leq$ 5 mins to fix

II. Usually 90% of DEV Time spend on Run-Time Bugs

- How to get rid of it?

• Force all bugs into small box (look there!)

★ Use "Add-A-Trick"

- Add 1-N Lines, Compile, then Test

3. EIO (Expected Input/Output)

*** Build Before Coding (Slice it into Itty-Bitty Stepping Stones)

- It focus design on what is important

*** Avoid "Gold-Platting"

Tuesday, August 28, 2018

Continued on August 30, but placed here since it continue ——>

- Making things look nice with nothing to functionality

4. Clean The Page. (~ 50 to n lines of code per page)

    - Usually one page for a Function so easy to read

——

August 30, 2018

Homework: Read Fischer

Chapter 1 Intro - 30pg

Chapter 2 Compiler Parts - 25pg

Chapter 3 Scanner/Lexer - 50pg

Mini Study Rules:

1. Textual Mean

    - Build/Use “Flash-Cards” (3x5)

2. Visual Memory

    IE: Charts, Graphs, etc

    - Draw it twice, looking

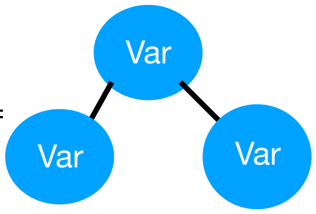
    - Draw it Blind

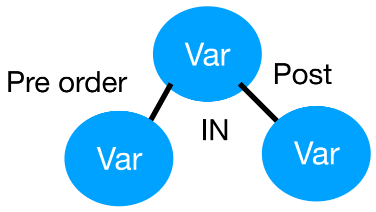
        • win 3x   • include labels

TreeWalking:

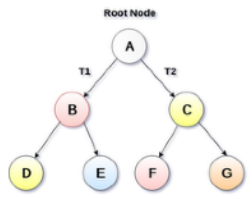
- Consist of:

    Left / Right / Lollypop =





```
CLASS Node
{
    INT VAL;
    NODE LKid;
    NODE RKid;
}
```

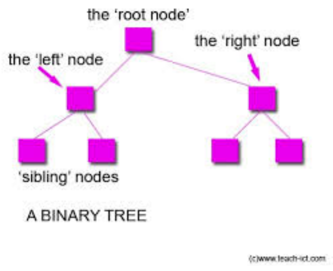


Binary Tree

Tuesday, August 28, 2018

To Do For TreeWalking:
1. Header
2. Basic Step
    - Do manually
3. Left/Right Recur
4. Deal with Lollypop
5. Glue

```
Void printTree(NODE root)
{
        # Basic Step
        If (NULL == root)
        {
                RGT; #Abbr. for returning nothing
        }
        # Left Recur
        printTree(root.LKid);
        # Right Recur
        printTree(root.RKid);
        # Deal with LollyPOP
        System.out.println(root.VAL);
        # GLUE
        // None
}
```



```
Void countTree(NODE RP)
{
        # Basic Step
        If (NULL == root)
        {
                RGT; #Abbr. for returning nothing
        }
        # Left Recur
        INT Lx = countTree(RP.LKid);
        # Right Recur
        INT Rx = countTree(RP.RKid);
        # Deal with LollyPOP
```

Tuesday, August 28, 2018

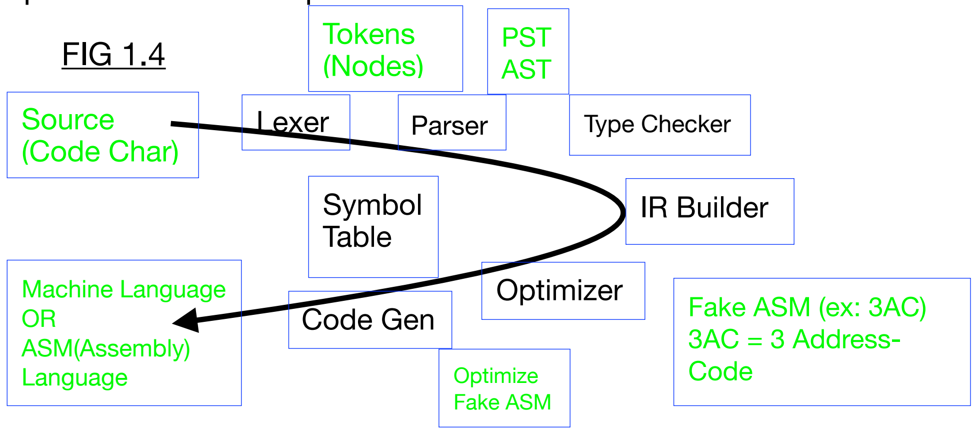
```
        Px = 1;
        # GLUE
        Return Lx + Rx + Px;
}

Void sumValTree(NODE RP)
{
        ....
        # Deal with LollyPOP
        Px = RP.VAL;
        ...
}

Void sumValForKind(NODE RP, INT RK)
{
        ....
        # Left Recur
        .... RK
        # Right Recur
        ..... RK
        # Deal with LollyPOP
        Px = (RK == RP.kind
                ? RP.VAL
                : θ );
}
```

Chapter 1 Parts of Compiler:

FIG 1.4



Lexer = Lexical Analysis

- Lang. REGEXES

Parser = Syntactic Analysis

- CFG (Context Free Grammar) Rules

Type Checker & IR Builder = Semantic Analysis (Good meaning)

- IR Builder (Intermediate Representation Builder)
    - In each stages, since they are not source or final, they are IR
- AST + Decoration

Optimizer

Code Generation = Final representation (Emiter Phase)

- “Emits” Machine/ASM/Byte Code
    - Bytecode usually mean for JAVA since it is old
        - For interpreter/VM Architecture
- Machine Architecture Description

Symbol Table:

- Contains all user-define names (names = symbols)
- Are builded into debugger

Front End:

- Between beginning to Syntactics Analysis

Back End:

- After Syntactics Analysis to end

PST (Parse Tree): Convert to AST (through Parser)

AST (Abstract Syntax Tree): In one simple operation from PST —> AST

———————————————————————————————

September 4, 2018

September 6, 2018