CSE211: Compiler Design

Oct. 6, 2021

- **Topic**: Finish PLY overview, go over symbol tables.
- **Questions**:
  - *Has anyone started on the homework? Any issues?*

from: https://en.wikipedia.org/wiki/Yak

Announcements

- Homework 1 is out
  - Due on the 18th
  - Get started early!
- Office hours tomorrow (2-3pm, E2-233)
- if you have ideas for projects, we can start discussing!
- Keep an eye out for homework questions/clarifications on slack

CSE211: Compiler Design

Oct. 6, 2021

- **Topic**: Finish PLY overview, go over symbol tables.

- **Questions**:
  - *Has anyone started on the homework? Any issues?*

from: https://en.wikipedia.org/wiki/Yak

Review: Parser generators

- Specify:
  - Tokens
  - Production Rules
  - Production Actions
- Parser generator gives you a function in which you can pass strings
  - Executes production actions
  - Error reporting

Review: PLY:

- How did we specify tokens?

- What are token actions?

- How did we specify production rules?
  - Are you allowed to in your homework?

- How did we specify precedence and associativity?

Review: PLY:

- Catch-up on the calculator example

Simplifying binary operations with Lambdas

```
def p_expr_bin(p):
    """
    expr : expr PLUS expr
         | expr MINUS expr
         | expr MULT expr
    """
    if p[2] == '+':
        p[0] = p[1] + p[3]
    elif p[2] == '-':
        p[0] = p[1] - p[3]
    elif p[2] == '*':
        p[0] = p[1] * p[3]
    else:
        assert(False)
```

Can be changed to (next slide)

Simplifying binary operations with Lambdas

```
def p_plusp(p):
    ”plusp : PLUS"
    p[0] = lambda x,y: x+y


def p_multp(p):
    ”multp : MULT"
    p[0] = lambda x,y: x*y


def p_minusp(p):
    ”minusp : MINUS"
    p[0] = lambda x,y: x-y
```

```
def p_expr_bin(p):
    """
    expr : expr plusp expr
         | expr minusp expr
         | expr multp expr
    """
    p[0] = p[2](p[1], p[3])
```

Can be changed to (next slide)

Multiline calculator example

- *A sequence of expressions?*

```
to_print = []

def p_expression_list(p):
    "expr_list : expr SEMI"
    to_print.append(p[1])


def p_expression_list_rec(p):
    "expr_list : expr_list expr SEMI"
    to_print.append(p[2])
```

Is this order important?

Multiline calculator example

- *A better error function?*

```
def p_error(p):
    print("Syntax error in input on line: %d" % p.lineno)
    exit(1)
```

What are other options? try to recover?

Multiline calculator example

- *Attempting to recover:*

```
def p_error(p):
    print("Syntax error in input on line: %d" % p.lineno)
    print("trying to recover")
    while True:
        tok = parser.token()
        if tok.type == 'SEMI': break
    print("trying restart after the ; on line %d" % p.lineno)
    to_print.append("ERROR")
    parser.restart()
```

How to handle keywords and ids

- How to differentiate keywords from ids:
  - e.g. “if”, from “x”
  - token for id is “[a-zA-Z]+”
  - it will also match keywords...

How to handle keywords and ids

```
tokens = ["IF", "ELSE", "ID"]


t_ID = "[a-zA-Z]+"
t_IF = "if"
t_ELSE = "else"
t_ignore = ' '

def t_error(t):
    print("Illegal character '%s'" % t.value[0])
    print("line number: %d" % t.lexer.lineno)
    exit(1)

lexer = lex.lex()

lexer.input("if")
```

parses ”if” as an ID!

How to handle keywords and ids

```
reserved = {
   'if'         : 'IF',
   'else'       : 'ELSE’
}

tokens = ["ID"] + list(reserved.values())


def t_ID(t):
    "[a-zA-Z]+"
    t.type = reserved.get(t.value, 'ID')
    return t
```

This will work!

Conclusion: lots of interesting features

- Modern parser generators are really great!
- I highly suggest reading the PLY readme
  - Even more examples and interesting functionality
- PLY was largely developed for educational purposes, but it's been reliable for me for several projects, especially other parts of your project are in Python.
- While I have never used it, Antlr is highly recommended. If anyone is interested in doing any of homework in Antlr let me know!

Back to presentation mode

- To discuss symbol tables!

One consideration: Scope

- What is scope?
- Can it be determined at compile time? Can it be determined at runtime?
- C vs. Python
- Anyone have any interesting scoping rules they know of?

One consideration: Scope

- Lexical scope example

```
int x = 0;
int y = 0;
{
  int y = 0;
  x+=1;
  y+=1;
}
x+=1;
y+=1;
```

What are the final values in x and y?

How to track scope?

- Symbol table
- Global object, accessible (and mutable) by all production actions

- two methods:
  - **`lookup(id)` :** `lookup an id in the symbol table. Returns None if the id is not in the symbol table.`

  - **`insert(id,info)` :** `insert a new id (or overwrite an existing id) into the symbol table along with a set of information about the id.`

What information might we store about an id?

a very simple programming language

VARIABLE_NAME = [a-z]+

INCREMENT = “\+\+”

TYPE = “int”

LB = “{“

RB = “}”

SEMI = “;”

```
int x;
x++;
int y;
y++;
```

statements are either a declaration or an increment

a very simple programming language

VARIABLE_NAME = [a-z]+

INCREMENT = “\+\+”

TYPE = “int”

LB = “{“

RB = “}”

SEMI = “;”

```
int x;
{
  int y;
  x++;
  y++;
}
y++;
```

statements are either a declaration or an increment

a very simple programming language

VARIABLE_NAME = [a-z]+

INCREMENT = “\+\+”

TYPE = “int”

LB = “{“

RB = “}”

SEMI = “;”

```
int x;
{
  int y;
  x++;
  y++;
}
y++;
```

statements are either a declaration or an increment

How to track scope?

- `SymbolTable ST;`

declare_variable: TYPE VARIABLE_NAME SEMI

`{}`

Say we are matched string:
`int x;`

`lookup(id)` `: lookup an id in the symbol table. Returns None if the id is not in the symbol table.`

`insert(id,info)` `: insert a new id (or overwrite an existing id) into the symbol table along with a set of information about the id.`

How to track scope?

- `SymbolTable ST;`

declare_variable: TYPE VARIABLE_NAME SEMI

```
{ST.insert(C[1],C[0])}
```

Say we are matched string:
`int x;`

In this example we are storing a type

How to track scope?

- `SymbolTable ST;`

Say we are matched string:
`x++;`

variable_inc: VARIABLE_NAME INCREMENT SEMI

```
{}
```

`lookup(id)` `: lookup an id in the symbol table. Returns None if the id is not in the symbol table.`

`insert(id,info)` `: insert a new id (or overwrite an existing id) into the symbol table along with a set of information about the id.`

How to track scope?

Say we are matched string:
`x++;`

- `SymbolTable ST;`

variable_inc: VARIABLE_NAME INCREMENT SEMI

```
{if not ST.lookup(x):
    raise SymbolTableException;
 else:
    ... // continue}
```

How to track scope?

- `SymbolTable ST;`

```
statement : variable_inc
          | declare_variable

statement_list : statement_list statement
               | statement
```

why do we have the statement list declared like this?

How to track scope?

- `SymbolTable ST;`

```
statement : variable_inc
          | declare_variable

statement_list : statement_list statement
               | statement
```

adding in scope

How to track scope?

- `SymbolTable ST;`

```
statement : variable_inc
          | declare_variable
          | LBAR statement_list RBAR

statement_list : statement_list statement
               | statement
```

How to track scope?

- `SymbolTable ST;`

statement : LBAR statement_list RBAR

start a new scope S

remove the scope S

How to track scope?

- Symbol table
- four methods:
  - **lookup(id)** : lookup an id in the symbol table. Returns None if the id is not in the symbol table.
  - **insert(id,info)** : insert a new id into the symbol table along with a set of information about the id.
  - **push_scope()** : push a new scope to the symbol table
  - **pop_scope()** : pop a scope from the symbol table

How to track scope?

- `SymbolTable ST;`

statement : LBAR statement_list RBAR

start a new scope S

remove the scope S

How to track scope?

- `SymbolTable ST;`

statement : LBAR statement_list RBAR

start a new scope S

remove the scope S

How to write a production action here?

How to track scope?

- `SymbolTable ST;`

statement : start_scope statement_list RBAR

start_scope : LBAR

add a new production rule!

How to track scope?

- `SymbolTable ST;`

```
statement : start_scope statement_list RBAR
{}

start_scope : LBAR
{ }
```

How to track scope?

- `SymbolTable ST;`

statement : start_scope statement_list RBAR
`{ST.pop_scope()}`

start_scope : LBAR
`{ST.push_scope()}`

How to implement a symbol table?

- Thoughts? What data structures are good at mapping strings?

- Symbol table
- four methods:
  - **lookup(id)** : lookup an id in the symbol table. Returns None if the id is not in the symbol table.

  - **insert(id,info)** : insert a new id into the symbol table along with a set of information about the id.

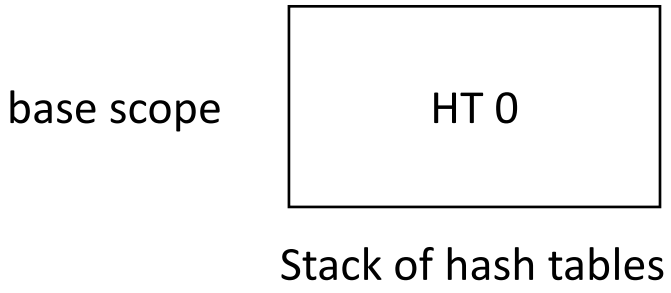
  - **push_scope()** : push a new scope to the symbol table

  - **pop_scope()** : pop a scope from the symbol table

How to implement a symbol table?

- Many ways to implement:

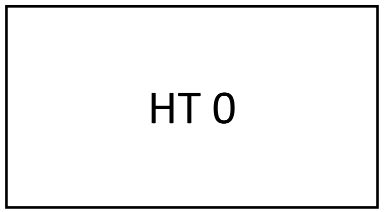
- A good way is a stack of hash tables:



How to implement a symbol table?

- Many ways to implement:
- A good way is a stack of hash tables:

`push_scope()`



Stack of hash tables

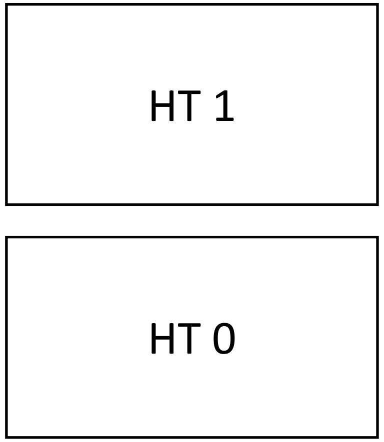
How to implement a symbol table?

- Many ways to implement:

- A good way is a stack of hash tables:

adds a new
Hash Table
to the top of the stack

`push_scope()`



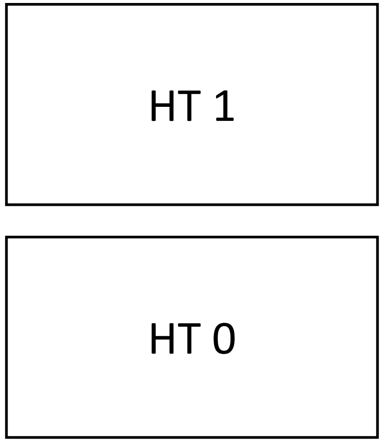
Stack of hash tables

How to implement a symbol table?

- Many ways to implement:

- A good way is a stack of hash tables:

`insert(id,data)`



Stack of hash tables

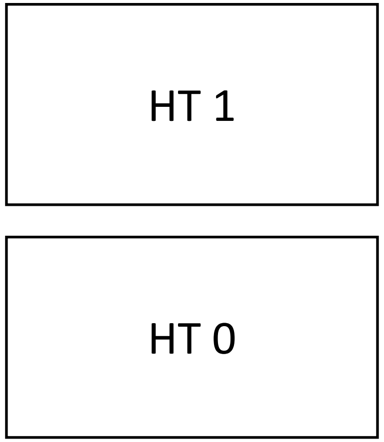
How to implement a symbol table?

- Many ways to implement:

- A good way is a stack of hash tables:

insert `(id -> data)` at top hash table

`insert(id,data)`



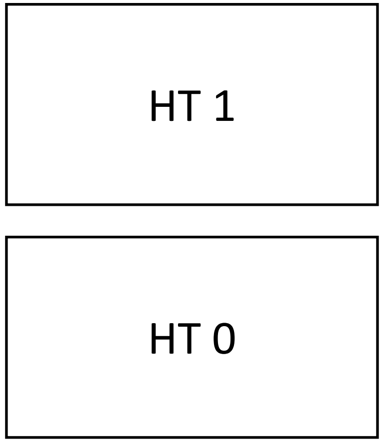
Stack of hash tables

How to implement a symbol table?

- Many ways to implement:

- A good way is a stack of hash tables:

`lookup(id)`



Stack of hash tables

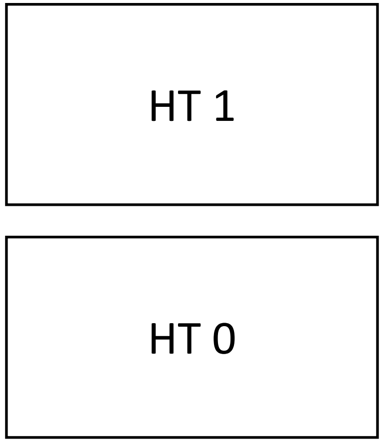
How to implement a symbol table?

- Many ways to implement:

- A good way is a stack of hash tables:

check here
first

`lookup(id)`

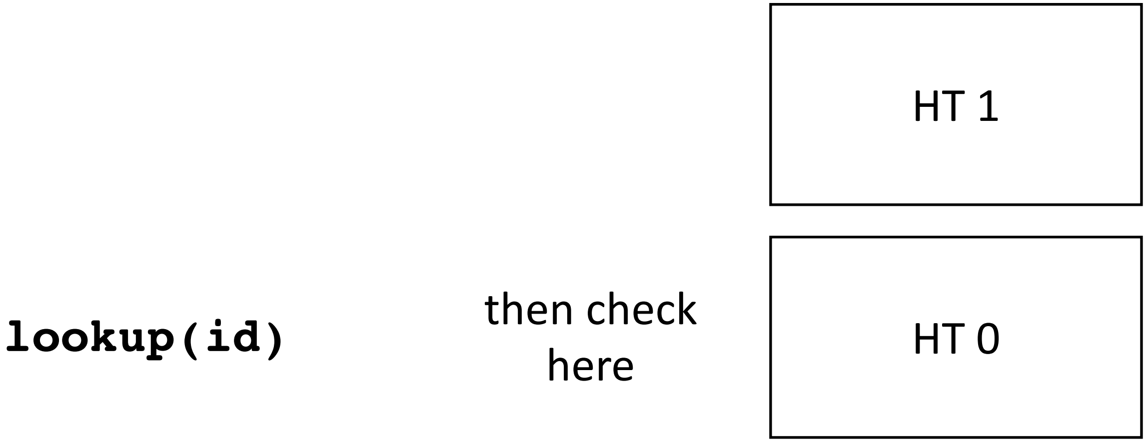


Stack of hash tables

How to implement a symbol table?

- Many ways to implement:

- A good way is a stack of hash tables:



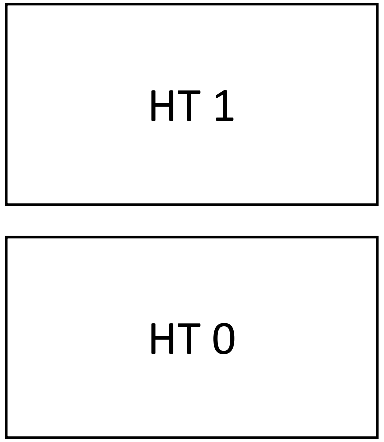
Stack of hash tables

How to implement a symbol table?

- Many ways to implement:

- A good way is a stack of hash tables:

`pop_scope()`

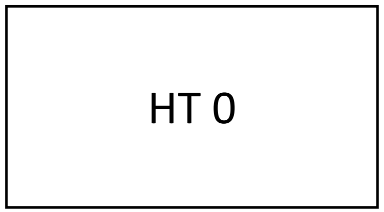


Stack of hash tables

How to implement a symbol table?

- Many ways to implement:

- A good way is a stack of hash tables:



Stack of hash tables

How to implement a symbol table?

- Example

```
int x = 0;
int y = 0;
{
  int y = 0;
  x++;
  y++;
}
x++;
y++;
```

HT 0

x = 2

y = 1

Stack of hash tables

See you on Friday!

- You should have everything you need to know to work on Homework part 1!

- Next class: Parsing regular expressions with derivatives

- Office hours tomorrow: (2 - 3 pm)