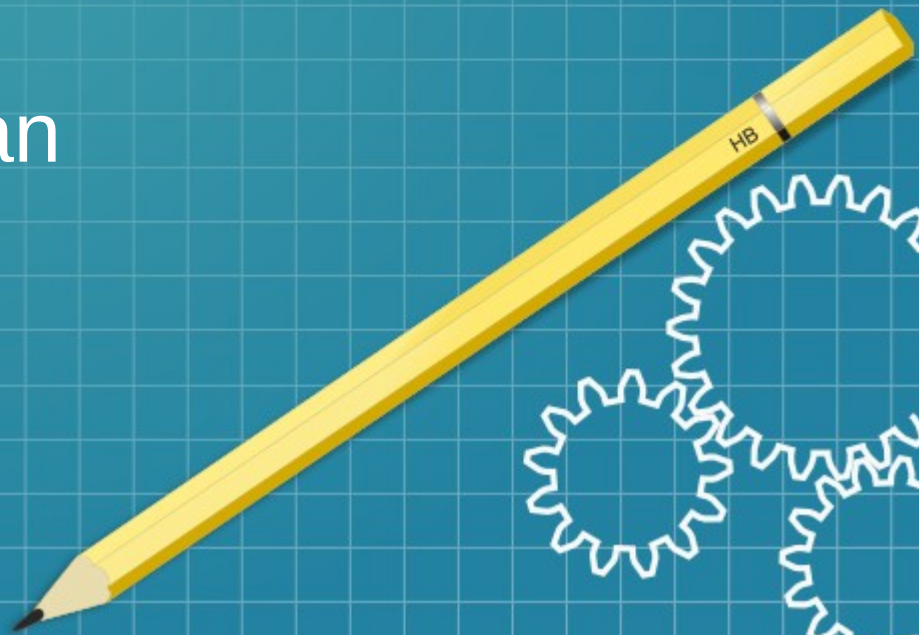
CS152 Phase 3: Code Generation

Daniel Tan



Generate Copy Statements

c := 100 = c, 100

Generate Arithmetic Operations

c := a + b	+ c, a, b
c := a – b	- c, a, b
c := a * b	* c, a, b
c := a / b	/ c, a, b
c := a % b	% c, a, b

Generate Array Operators

Write to an array

a[0] := 100 [ ]= a, 0, 100

Read from an array

b := a[0] =[ ] b, a, 0

Generate Array Operators

a[0] := a[1]

=[ ] _temp, a, 1

[ ]= a, 0, _temp

How to call a Functions

```
c := add(a, b);
```

```
param a
param b
call add, c
```

Push 1st parameter “a” first, then push the 2nd parameter “b” second, then call the “add” function

How to get function parameters

```
function add;
beginparams
a : integer;
b : integer;
endparams
beginlocals
endlocals
beginbody
        return a + b;
endbody
```

```
func add
. a
. b
= a, $0
= b, $1
. _temp0
+ _temp0, a, b
ret _temp0
endfunc
```

Do **“= a, $0”** and **“= b, $1”** to get the 1st/2nd parameter from the stack

Create a Code Node Struct

- Create a Code Node Struct
- “code” is the code in code associated with the node
- “name” is a register associated with the node

```
#include <string>

struct CodeNode {
    std::string code;
    std::string name;
};
```

Get an identifier

- name is set to the identifier name
- Code is empty, since no code is needed to get an identifier
- Check that the identifier was declared before using it
- $$ = node, pass it up the grammar

```
var: IDENT {
  CodeNode *node = new CodeNode;
  node->code = "";
  node->name = $1;
  std::string error;
  if (!find(node->name, Integer, error)) {
    yyerror(error.c_str());
  }
  $$ = node;
}
```

a = ...something

```
statement: IDENT ASSIGN expression {

  std::string var_name = $1;
  std::string error;
  if (!find(var_name, Integer, error)) {
    yyerror(error.c_str());
  }

  CodeNode *node = new CodeNode;
  node->code = $3->code;
  node->code += std::string("= ") + var_name + std::string(", ") + $3->name + std::string("\n");;
  $$ = node;
}
```

- Set the node code equal to the expression's code
- Add the "= ident, expression.name" to the code
- $$ = Code Node

Addition

```
expression: multiplicative_expression ADD multiplicative_expression {
  std::string temp = create_temp();
  CodeNode *node = new CodeNode;
  node->code = $1->code + $3->code + decl_temp_code(temp);
  node->code += std::string("+ ") + temp + std::string(", ") + $1->name + std::string(", ") + $3->name + std::string("\n");
  node->name = temp;
  $$ = node;
}
```

- Take the two expressions' code, concatenate the code together
- Add "+ _temp, a, b"