Phase 3: Code Generation

CS152 Compiler Design

Tan, Daniel

02 May 2023

Generate Copy Statements

c := 100 = c, 100

Generate Math Statements

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 Statements

c[0] := a;	[ ] = c, 0, a
a := c[0];	=[ ] a, c, 0

Generate Array Statements

c[0] := c[1];

. temp

=[ ] temp, c, 1

[ ]= c, 0, temp

Calling Functions

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

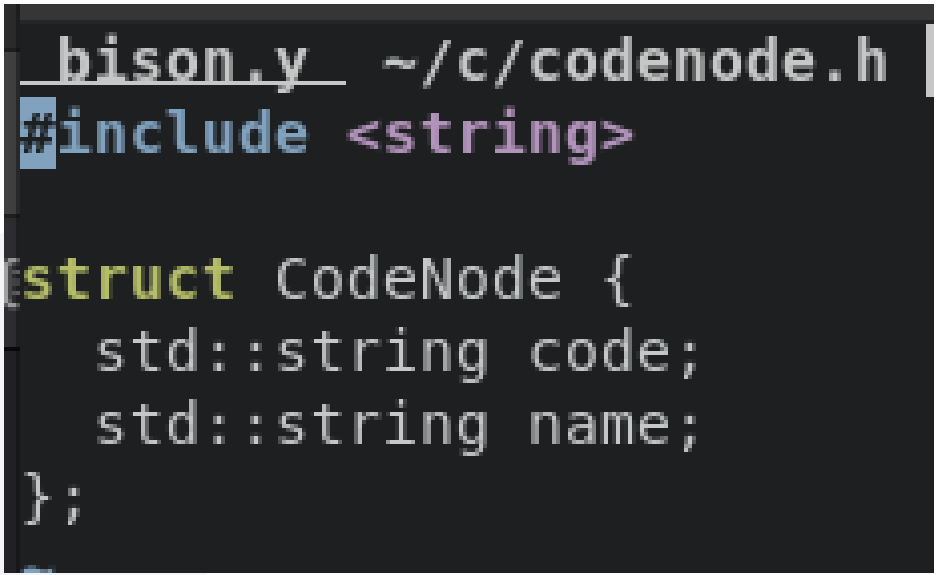
```
param a
param b
call add, c
```

Getting Function Parameters

```
add(a, b){
    return (a + b);
}
```

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

Code Node Struct



- “code” is the code generated by the node
- “name” is the name of the result register of the code node

Handling variables

```
var: IDENT L_SQUARE_BRACKET expression R_SQUARE_BRACKET {
  std::string temp = create_temp();
  CodeNode *node = new CodeNode;
  node->code = $3->code + decl_temp_code(temp);
  node->code += std::string("=[] ") + temp + std::string(", ") + s
 std::string("\n");
  node->name = temp;

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

Assigning variables

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

Combining Statements

```
statements: statement SEMICOLON statements {
   CodeNode *node = new CodeNode;
   node->code = $1->code + $3->code;
   $$ = node;
}
          | %empty {
   CodeNode *node = new CodeNode;
   $$ = node;
}
```

Utility Functions

```
bool has_main() {
  bool TF = false;
  for(int i=0; i < symbol_table.size(); i++) {
    Function *f = &symbol_table[i];
    if (f->name == "main")
      TF = true;
  }
  return TF;
}

std::string create_temp() {
  static int num = 0;
  std::string value = "_temp" + std::to_string(num);
  num += 1;
  return value;
}

std::string decl_temp_code(std::string &temp) {
  return std::string(". ") + temp + std::string("\n");
}
```