College of Engineering and Computer Science
UNIVERSITY OF CENTRAL FLORIDA

Code Generation for Variables

COP-3402 Systems Software
Paul Gazzillo

Overview

- Variable declaration emits an alloca, updates symtab
- Variable assignment uses expression()
    - expression() is already done
        - API: returns a temp register (or value)
        - Composes with assignment statement
- Read statement is like print, but also emits a store
- Update factor() to support IDENTIFIER
    - Emit a load to a new temporary register

Declarations Allocate Stack Space

- Parse the identifier
- Check for previous definition
- Emit alloca instruction
- Save address in symtab

```
declaration():
  assert consume() == 'int'
  ident = consume()
  assert consume() == ';'
  if (contains(ident))
error()
  result = newtemp()
  emit result "= alloca"
  put(ident, result)
```

Read Stores a Value at Variable's Address

- Get variable's address (if declared)
- Emit read_integer call
- Emit store to variable address

```
read():
  assert consume() == 'read'
  ident = consume()
  assert consume == ';'
  addr = lookup(ident) or error()
  result = newtemp()
  emit result " = read_integer()"
  emit "store " result ", " addr
```

Assign Stores a Value at Variable's Address

- First evaluate the right-hand-side expression
  - This returns the thing to store (temp or value)
- Then lookup address and emit store
- expression() already done in project 1
  - Composes with assignment function

```
assign():
  ident = consume();
  assert consume() == '='
  result = expression()
  assert consume() == ';'
  addr = lookup(ident) or error()
  emit "store " result ", " addr
```

Factor Gets a Value from Variable's Address

- Variable's value is stored at an address
- Symbol table tracks each variable's address
    - Represented with an LLVM variable name
- Using a variable is loading its value from memory
- Lookup variable's address
- Emit a load to new temp
- Return new temp
    - Composes with term()

```
factor():
  // ... (the rest of the function)
  elif (next is IDENTIFER):
    ident = consume()
    addr = lookup(ident) or error()
    result = newtemp()
    emit result " = load " addr
    return result
  // ... (the rest of the function)
```

Expression Parser Simplification

- Replace the right-recursive grammar (E', T') with a loop

E -> T E'

E' -> + T E' | - T E | ε

E -> T (+ T)* | T (- T)*

```
expression():
  left = term()
  while (next is PLUS or MINUS):
    op = consume()
    right = term()
    result = newtemp()
    emit result " = " opname(op) " " left ", " right
    left = result
  return left
```

Demo: Code Generation for Variables

```
int x;
int y;
read x;
y = 1 + x * 7;
```