CMPE 152: Compiler Design

November 21 Lab

Department of Computer Engineering
San Jose State University

Fall 2017
Instructor: Ron Mak
www.cs.sjsu.edu/~mak

Speed Optimization: Strength Reduction

- Replace an operation by a faster equivalent operation.

Speed Optimization: Strength Reduction, *cont'd*

- Example: Suppose the integer expression **`5*i`** appears in a tight loop.
  - Given: Multiplication is more expensive than addition.
  - One solution: Generate code for **`i+i+i+i+i`** instead.
  - Another solution: Treat the expression as if it were written **`(4*i)+i`** and do the multiplication as a shift left of 2 bits.
    - Generate the code to shift the value of **`i`** and then add the original value of **`i`**.

Speed Optimization: Dead Code Elimination

- Suppose we have the **WHILE** statement:

```
WHILE i <> i DO
    BEGIN
        ...
    END
```

If there are no statement labels, none of the statements in the compound statement can ever be executed.

- Don’t emit any code for this **WHILE** statement.

Speed Optimization: Loop Unrolling

- Loop overhead: initialize, test, and increment.
- Example:

```
FOR i := 1 TO n DO BEGIN
    FOR j := 1 TO 3 DO BEGIN
        s[i,j] := a[i,j] + b[i,j]
    END
END
```

- Unroll the inner loop by generating code for:

```
FOR i := 1 TO n DO BEGIN
    s[i,1] := a[i,1] + b[i,1];
    s[i,2] := a[i,2] + b[i,2];
    s[i,3] := a[i,3] + b[i,3];
END
```

Common Subexpression Elimination

- Example:

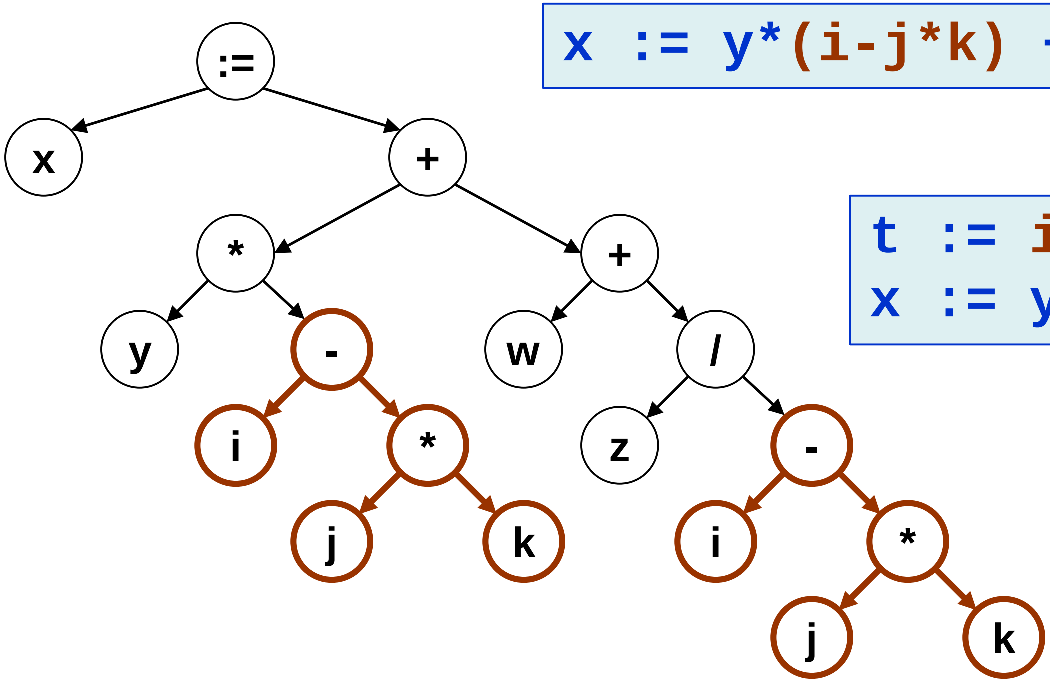
```
x := y*(i-j*k) + (w + z/(i-j*k))
```

- Generate code as if the statement were instead:

```
t := i-j*k;
x := y*t + (w + z/t);
```

- This may not be so easy for the back end to do!

Common Subexpression Elimination, *cont'd*



```
x := y*(i-j*k) + (w + z/(i-j*k))
```

```
t := i-j*k;
x := y*t + (w + z/t)
```

- How do you recognize the <u>common subexpression</u> in the parse tree?

Debugging Compiler

- AKA development compiler
- Used during program development
- Fast compiles = fast turnaround
- Doesn’t change the order of the generated code.
- Easy for debuggers (such as Eclipse) to set breakpoints, single-step, and monitor changes to the values of variables.

Optimizing Compiler

- AKA production compiler
- Used after a program has been "thoroughly" debugged.
- Can optimize for speed, memory usage, or power consumption.
- Different levels of optimization.

Compiling Object-Oriented Languages

- Extra challenges!
- Dynamically allocated objects
  - Allocate objects in the heap.
- Method overloading
- Inheritance
- Virtual methods