

University of Kansas | Drew Davidson

ECS 665

COMPILER *CONSTRUCTION*

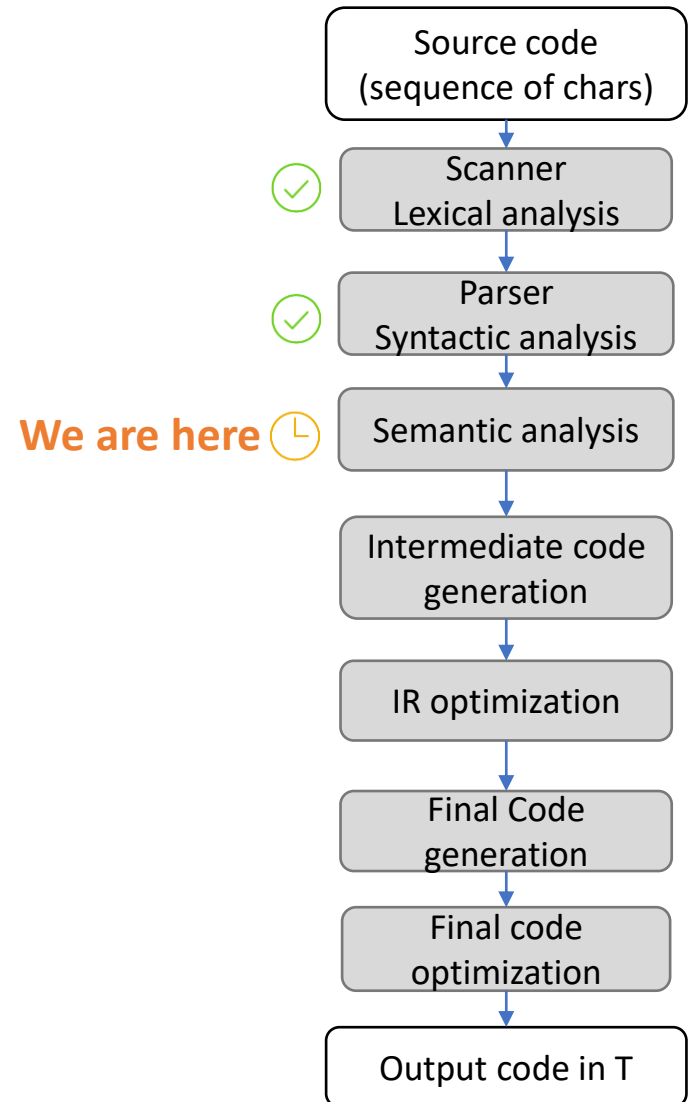
Parameters

Compiler Construction: Progress

You are here

Continuing exploration of language semantic features

- Informs design/behavior of our semantic analysis



Last Time

Lecture Review – Error Checking

The Limits of Error Checking

- Halting problem

Partial Correctness

- Partial Correctness
- Soundness
- Completeness



Semantics

Today's Outline

Parameters

Vocabulary

- lval/rval
- Memory references
- Calls

Parameter Passing

- Call by value
- Call by reference
- Call by name
- Call by value-result



Semantics

Vocabulary

Parameters

- Define a couple of terms that are helpful to talk about parameters
 - We already encountered some of these in passing



L and R Values

Parameters

- L-Value
 - A value with a place of storage
- R-Value
 - A value that may not have storage

```
a = 1;  
a = b;  
b++;
```

Bad Use of R-Values

Parameters

- Can prevent programs that are valid in pass by value from working in pass by reference
 - Literals (for example) do not have locations in memory
- The type checker should catch these errors.

Memory References

Parameters

- Pointer
 - A variable whose value is a memory address
- Aliasing
 - When two or more variables reference the same address

Calls

Parameters

Caller

- The *source* function initiating the call

Callee

- The *target* function receiving the call

Call Site

- The location within the caller where the call happens



Calls

Parameters

Caller

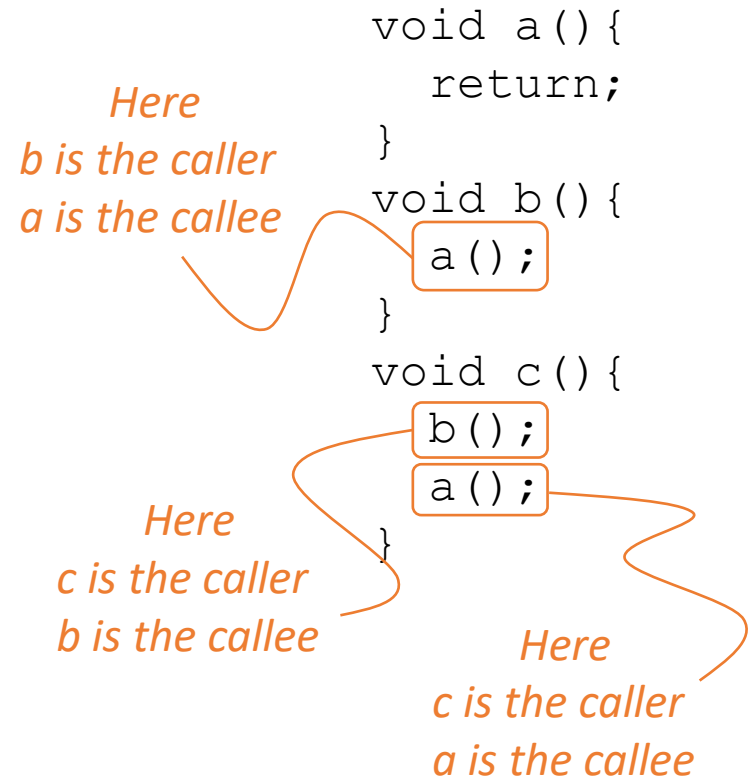
- The *source* function initiating the call

Callee

- The *target* function receiving the call

Call Site

- The location within the caller where the call happens



Call Chains

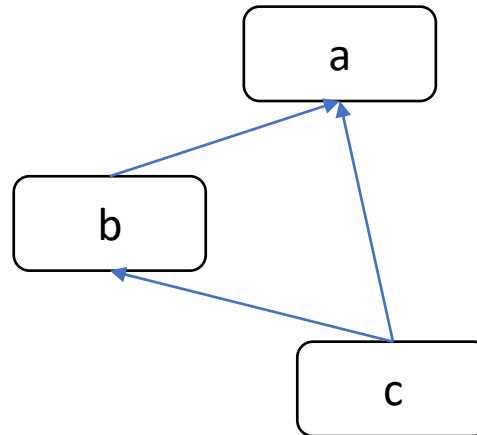
Parameters

Call Graph

- A node-based representation of possible call structure
 - Edge: caller -> callee

Call chain

- A realizable path of calls (c,b,a)



```
void a() {  
    return;  
}  
void b() {  
    a();  
}  
void c() {  
    b();  
    a();  
}
```

Arguments

Parameters

- In definition:

```
void v(int a, int b, bool c) { ... }
```

- Terms

- Formals / formal parameters / parameters

- In call:

```
v(a+b,8,true);
```

- Terms

- Actuals / actual parameters / arguments



Parameter Passing Schemes

Parameters

- We'll talk about some different varieties
 - Some of these are more used than others
 - Each has advantages / uses

Pass by Value

Parameters

- On function call
 - *Values* of actuals are copied into the formals
 - C and java always pass by value

```
void fun(int a) {  
    int a = 1;  
}  
  
void main() {  
    int i = 0;  
    fun(i);  
    print(i);  
}
```

Pass by Reference

Parameters

- On function call
 - The address of the actuals are *implicitly* copied

```
void fun(int a) {  
    a = 1;  
}  
  
void main() {  
    int i = 0;  
    fun(i);  
    print(i);  
}
```

Language Examples

Parameters

- Pass by value
 - C, Java, C#
- Pass by reference
 - Allowed in C++ and Pascal

Java and C# Are Pass by Value?!

Parameters

- All non-primitive L-values are references

```
void fun(int a, Point p) {  
    a = 0;  
    p.x = 5;  
}  
  
void main() {  
    int i = 0;  
    Point k = new Point(1, 2);  
    fun(i,k);  
}
```

Pass by Value-Result

Parameters



- When function is called
 - Value of actual is passed
- When function returns
 - Final values are copied back to the actuals
- Used by Fortran IV, Ada
 - As the language examples show, not very modern

Pass by Name

Parameters



- Conceptually works as follows:
 - When a function is called
 - Body of the callee is rewritten with the **text** of the argument
 - Only really makes sense with non-local scope rules
 - Like macros in C / C++

COMPILER

Optimization

Execution

Codegen

Intermediate
Representation

Runtimes

SDD

Semantics

Parsing

Lexical
Analysis

Syntactic
Definition

LAND

GLOPPY

CANDY CASTLE

KING KANDY

QUEEN FROSTING

GRAMMA NUTT

PRINCESS LOLLY

LOST IN LOLLOPOP WOODS

SPUCK ON A BERRY BUSHGROW

LORD LICORICE

LICORICE CASTLE

JOLLY

PLUMPY