Turn in that Homework!

- As promised, I'll destroy anything over 5 minutes late

Announcement: It's Election Day!

- Polls close at 8:00 PM
- https://myvote.wi.gov/
  - Sample ballot
  - Polling info

Announcement: UWCS Nest

- Software Competition
  - https://contest.cs.wisc.edu/
  - Unlimited team size
  - Flexible project scope (seemingly anything)
- Probably lots of prize money
  - last year
    - $3K 1st place, $2K 2nd place, $1K 3rd place, $1K for "Wisconsin Idea"
- **Nov. 11, 2014 Kickoff**
  5:00 PM, 2310 Computer Science Building

Announcement: Midterm Mitigation

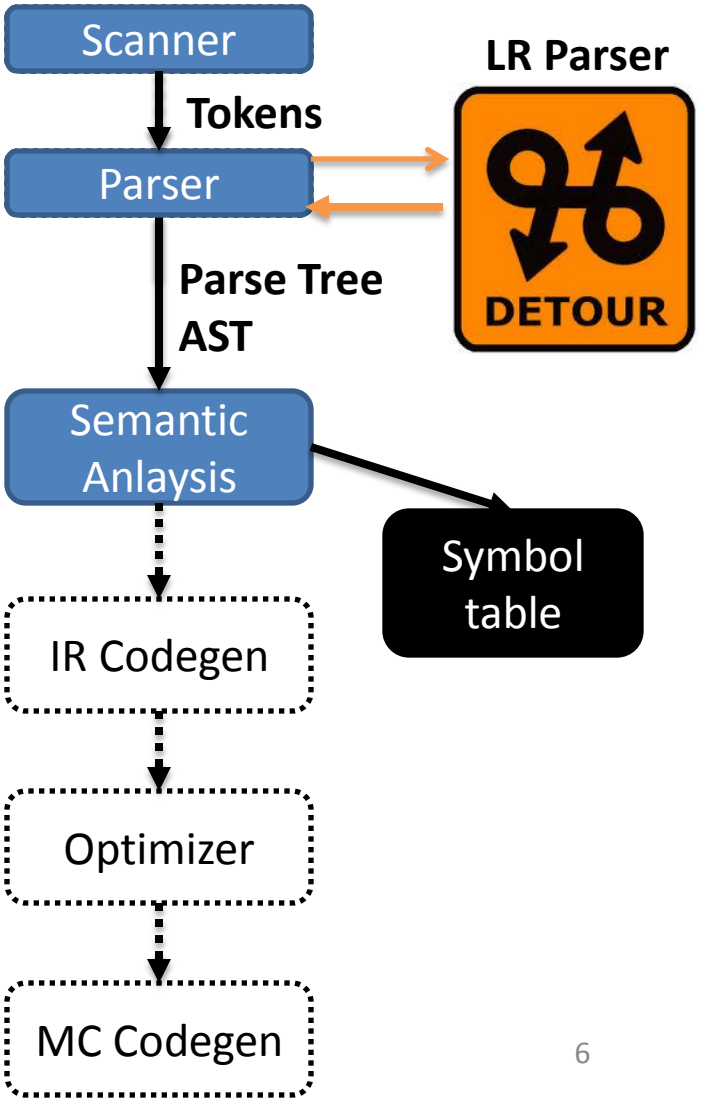
- Going with the extra weight
  - If you do better on the final than the midterm, final grade replaces your midterm
  - If you do worse on the final, then you keep your midterm grade
- Final will have less time pressure, may be slightly more challenging questions
  - Still will allow 1 pieces of standard letter paper
  - Hoping for much better pictures this time

CS536

Types

Roadmap

- Back from our LR Parsing Detour
- Name analysis
  - Static v dynamic
  - Scope
- Today
  - Type checking



Lecture Outline

- Type Safari
  - Type system concepts
  - Type system vocab
- C-Flat
  - Type rules
  - How to apply type rules
- Data representation
  - Moving towards actual code generation
  - Brief comments about types in memory

Say, What *is* a Type?

- Short for “data type”
  - Classification identifying kinds of data
  - A set of possible values which a variable can possess
  - Operations that can be done on member values
  - A representation (perhaps in memory)

Type Intuition

You can't do this:

```
int a = 0;
int * pointer  = &a;
float fraction = 1.2;
a = pointer + fraction;
```

… or can you?

Components of a type system

- Primitive types
  - int, bool, void
- Rules for building types (type constructors)
  - struct (in our language)
- Means of determining if types are compatible
  - bool cannot be used as an int in our language
- Rules for inferring type of an expression

Type Rules

- For every operator (including assignment)
  - What types can the operand have?
  - What type is the result?

- Examples

```
double a;
int b;
a = b;
b = a;
```

Legal in Java, C++

Legal in C++, not in Java

Type Coercion

- Implicit cast from one data type to another
  - Float to int
  - Narrow form: type promotion
    - When the destination type can represent the source type
    - float to double

Types of Typing I: **When** do we check?

- Static typing
  - Type checks are made before execution of the program (compile-time)
- Dynamic typing
  - Type checks are made during execution (runtime)
- Combination of the two
  - Java (downcasting v cross-casting)

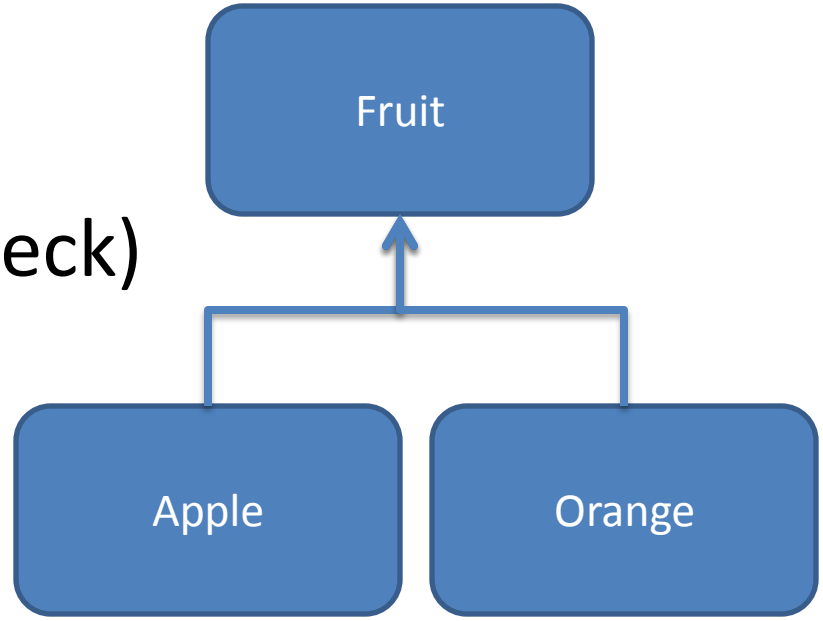
Example: Casting

- Cross-casting (static check)

```
Apple a = new Apple();
Orange o = (Orange)a;
```

- Downcasting (dynamic check)

```
Fruit f = new Apple();
if ( … ) {
  a = new Orange();
}
Apple two = (Apple)f;
```



Static v Dynamic Tradeoffs

- Statically typed
  - Compile-time optimization
  - Compile-time error checking
- Dynamically typed
  - Avoid dealing with errors that don’t matter
  - Some added flexibility

Duck Typing

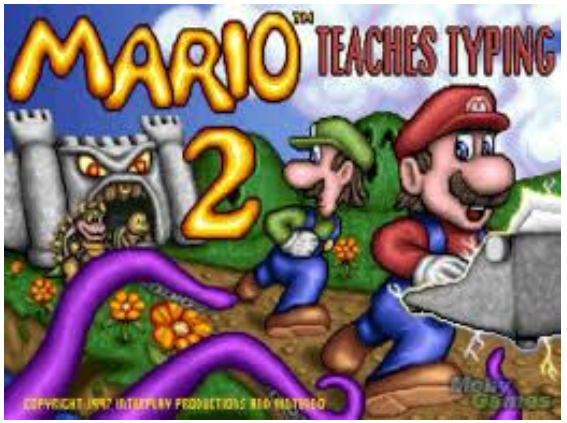
- Type is defined by the methods

```
class bird:
    def quack(): print("quack!")
class mechaBird:
    def quack(): print("101011…")
```

- Duck Punching
  - Runtime modifications to allow duck typing

Types of Typing II: **What** do we check?

- Strong vs weak typing
  - Degree to which type checks are performed
  - Degree to which type errors are allowed to happen at runtime
  - Continuum without precise definitions



Strong v Weak

- No universal definitions but…
  - Statically typed is often considered stronger (fewer type errors possible)
  - The more implicit casts allowed the weaker the type system
  - The fewer checks performed at runtime the weaker

Strong v Weak Example

- C (weaker)

```
union either{
    int i;
    float f;
} u;
u.i = 12;
float val = u.f;
```

- SML (stronger)

```
real(2) + 2.0
```

Type Safety

- Type safety
  - All successful operations must be allowed by the type system
  - Java was explicitly designed to be type safe
    - If you have a variable with some type, it is guaranteed to be of that type
  - C is not
  - C++ is a little better

Type Safety Violations

- C
  - Format specifier

```
printf("%s", 1);
```

  - Memory safety

```
struct big{
    int a[100000];
};
struct big * b = malloc(1);
```

- C++
  - Unchecked casts

```
class T1{ char a};
class T2{ int b; };
int main{
    T1 * myT1 = new T1();
    T2 * myT2 = new T2();
    myT1 = (T1*)myT2;
}
```

Type System Tradeoffs

- Determine the type of each expression subtree
- Find type errors

Type System of C-Flat

C♭

C-Flat type system

- Primitive types
  - int, bool, void
- Type constructors
  - struct
- Coercion
  - bool cannot be used as an int in our language (nor vice-versa)

C-Flat Type Errors I

- Arithmetic operators must have **int** operands
- Equality operators **==** and **!=**
  - Operands must have same type
  - Can't be applied to
    - Functions (but CAN be applied to function results)
    - struct name
    - struct variables
- Other relational operators must have **int** operands
- Logical operators must have **bool** operands

C-Flat Type Errors II

- Assignment operator
  - Must have operands of the same type
  - Can't be applied to
    - Functions (but CAN be applied to function results)
    - struct name
    - struct variables
- For cin >> x;
  - x cannot be function struct name, struct variable
- For cout << x;
  - x cannot be function struct name, struct variable
- Condition of if, while must be boolean

C-Flat Type Errors III

- Calling something that's not a function
- Calling a function with
  - Wrong number of args
  - Wrong types of args
    - Also will not allow struct or functions as args
- Returning a value from a void function
- Not returning a value in a non-void function function
- Returning wrong type of value in a non-void function

C-Flat Type Errors III

- Calling something that's not a function
- Calling a function with
  - Wrong number of args
  - Wrong types of args
    - Also will not allow struct or functions as args
- Returning a value from a void function
- Not returning a value in a non-void function function
- Returning wrong type of value in a non-void function

Looking Towards Next Lecture

- Starting Thursday
  - Look at data (and therefore types) is represented in the machine
  - Start very abstract, won't talk about an actual architecture for awhile
  - No intrinsic notion of types. We'll have to add code for type checking ourselves