Type

- Definition
- Examples
- Question 1: Will the values be bounded?
- Question 2: Will the types be pre-defined?
- Type errors
- Type systems
Can’t we detect type errors just by looking at the value?

- Machine level:
  0100 0000 0101 1000 0000 0000 0000 0000

- The floating point number 3.375
- The 32-bit integer 1,079,508,992
- Two 16-bit integers 16472 and 0
- Four ASCII characters: @ X NUL NUL

Types of Type Systems

- Static
- Dynamic
- Strongly typed language
Java: Why dynamic typing for type casting?

```java
public class Dyn_Typing {
    public static Object f() {
        int x = Integer.parseInt(System.console().readLine("Enter a number: "));
        if (x%2 == 0) //x is even
            String st = "Even number!";
            return st;
        else
            return new Integer(x);
    }

    public static void main(String[] args) {
        String st = (String) f();
        System.out.println(st);
    }
}
```

Confusion around strongly typed language

- [http://ericlippert.com/2012/10/15/is-c-a-strongly-typed-or-a-weakly-typed-language/](http://ericlippert.com/2012/10/15/is-c-a-strongly-typed-or-a-weakly-typed-language/)
- [http://c2.com/cgi/wiki?StronglyTyped](http://c2.com/cgi/wiki?StronglyTyped)
Is C strongly typed?

- No, because errors can go undetected.

```c
#include <stdio.h>
union {int i; float f;} u;
int main()
{
    float x = 0.0;
    u.i = 987654321;
    x = x + u.f;
    printf("%f
", x);
    return 0;
}
```

Basic vs. non-basic types

- Basic: int, float, bool, char
  - Implicit type conversion: narrowing vs. widening
  - C vs Java vs Python
- Non-basic: pointer, string, array, list, etc. + programmer-defined types
  - Implementation issues
Implementation of non-basic types
Arrays in C and Java
Other non-basic: read the book

List vs. array

- Size of list may vary at run-time
- List elements may not be contiguous in memory
- List may contain heterogeneous data (Python)
Other issues (read the book)

- Functions as types in C
- void qsort(
  void *array,
  size_t nitems,
  size_t size,
  int (*compare)(const void *, const void*)
)
- Java?
- Sub-types
  - OOP: Subclass
  - Polymorphism

Type System
Ch. 6
Type System for CLite

- Static typing and type checking at compile time
- Single function: main
  - Is static typing the same as dynamic typing?
- Single scope: no nesting, no global variables

Next: Type rules

Example Clite Program

```c
void main ( ) {
    int n;
    int i;
    int result;
    n = 8;
    i = 1;
    result = 1;
    while (i < n) {
        i = i + 1;
        result = result * i;
    }
    print result;
}
```
Type Rule 1

- All referenced variables must be declared.
- Type map is a set of ordered pairs
  E.g., {<n, int>, <i, int>, <result, int>}
  - Can implement as a hash table

Type Rule 2

- All declared variables must have unique names.

```c
void main ( ) {
  int n;
  int i;
  int result;
  n = 8;
  i = 1;
  result = 1;
  while (i < n) {
    i = i + 1;
    result = result * i;
  }
  print result;
}
```
Type Rule 3

- Identifier must not be a keyword
- How to implement?
- Can enforce it before semantic analysis!
  (Assignment 1)

Type checking

- A program is valid if
  - Declarations are valid and
  - Rest is valid wrt Declarations

```c
void main ( ) {
  int n;
  int i;
  int result;
  n = 8;
  i = 1;
  result = 1;
  while (i < n) {
    i = i + 1;
    result = result * i;
  }
  print result;
}
```
Type checking

- **Validity of a Statement:**
  - Assignment statement is valid if
    - Its target *Variable* is declared
    - Its source *Expression* is valid
    - If the target *Variable* is float, then the type of the source *Expression* must be either float or int
    - Otherwise if the target *Variable* is int, then the type of the source *Expression* must be either int or char
    - Otherwise, the target *Variable* must have the same type (e.g., bool) as the source *Expression*

Type checking

- A conditional stmt (or if stmt) is valid if:
  - Its test *Expression* is valid and has type bool
  - Its body is valid
- A while loop is valid if:
  - Its test *Expression* is valid and has type bool
  - Its body is valid
Group work: Expressions

- Define the validity of any expression w.r.t. type.
- That is, define the compatibility between values and operations in any expression.
- Cases:
  - Single term
  - More complex: binary operations, unary operations
  - Binary operations: think about all possible kinds of binary operations w.r.t. type (be simplistic)

Type checking

- **Validity of an Expression:**
  - A Value is always a valid Expression.
  - A Variable is a valid Expression if it appears in the type map.
  - A binary operation is valid if:
    - Its Expressions term1 and term2 are valid
    - If its Operator op is arithmetic, then both Expressions must be int or both must be float
    - If op is relational, then both Expressions must have the same type
    - If op is && or ||, then both Expressions must be bool
  - A unary operation is valid if:
    - Its Expression term is valid,
    - ...

Validity of an Expression:

- A Value is always a valid Expression.
- A Variable is a valid Expression if it appears in the type map.
- A binary operation is valid if:
  - Its Expressions term1 and term2 are valid
  - If its Operator op is arithmetic, then both Expressions must be int or both must be float
  - If op is relational, then both Expressions must have the same type
  - If op is && or ||, then both Expressions must be bool
- A unary operation is valid if:
  - Its Expression term is valid,
  - ...

Validity of an Expression:
Group work: type of an expr.

- Define the type of an expression

Type checking

- The type of an Expression $e$ is:
  - If $e$ is a Value, then the type of that Value.
  - If $e$ is a Variable, then the type of that Variable.
  - If $e$ is a Binary op $\text{term1}$ $\text{term2}$, then:
    - If op is arithmetic, then the (common) type of $\text{term1}$ or $\text{term2}$
    - If op is relational, $\&\&$ or $\mid\mid$, then bool
  - If $e$ is a Unary op $\text{term}$, then:
    - If op is $!$ then bool
    - ...