# Language Design Overview of COOL

Lecture 2

# Lecture Outline

- Today's topic: language design
  - Why there are so many programing languages?
  - Why are there new languages?
  - Good-language criteria
- History of ideas:
  - Abstraction
  - Types
  - Reuse
- Cool
  - The  Course Project

# Why So Many Languages?

Application domains have distinctive and conflicting needs

1) Scientific applications:

- high-performance numerical computations
- handle arrays well
- parallelization
- GPU

FORTRAN, Python, etc

2) System applications:

- Control of resource
- Real time constrains

C, C++

# Why So Many Languages? (continue)

3) Business application:
   - Report generation
   - Data analysis
   - Persistence
   - Shared by multiple user
   - Distributed or Centralized storage
   SQL, Node, Java, .NET, Sala

# Programming Language Economics 101

- Languages are adopted to fill a void
  - Enable a previously difficult/impossible application

- Programmer training is the dominant cost
  - And rewriting code
  - Languages with many users are replaced rarely
  - Popular languages become ossified
  - But easy to start in a new niche . . .

# Good Programing language

# Good Programing language

- No universally accepted metrics for design
- Claim: “A good language is one people use”

# Trends

- Language design
  - Many new special-purpose languages
  - Popular languages stick around (perhaps forever)
    - Fortran and Cobol

- Compilers
  - Ever more needed and ever more complex
  - Driven by increasing gap between
    - new languages
    - new architectures
  - Venerable and healthy area

# Why Study Languages and Compilers ?

5. Increase capacity of expression

4. Improve understanding of program behavior

3. Increase ability to learn new languages

2. Learn to build a large and reliable system

1. See many basic CS concepts at work

## Symbol Table

A symbol table may serve the following purposes depending upon the language in hand:

- To store the names of all entities in a structured form at one place.
- To verify if a variable has been declared.
- To implement type checking, by verifying assignments and expressions in the source code are semantically correct.
- To determine the scope of a name (scope resolution).

# Symbol Table

**Implementation:** If a compiler is to handle a small amount of data, then the symbol table can be implemented as an unordered list, which is easy to code, but it is only suitable for small tables only. A symbol table can be implemented in one of the following ways:

- Linear (sorted or unsorted) list
- Binary Search Tree
- Hash table

Among all, symbol tables are mostly implemented as hash tables.

# Symbol Table

It is used by various phases of the compiler as follows:-

**1- Analysis (front end)**

**- Lexical Analysis:** Creates new table entries in the table, for example like entries about tokens.

**- Syntax Analysis:** Adds information regarding attribute type, scope, dimension, line of reference, use, etc in the table.

**- Semantic Analysis:** Uses available information in the table to check for semantics i.e. to verify that expressions and assignments are semantically correct(type checking) and update it accordingly.

# Symbol Table

It is used by various phases of the compiler as follows:-

**1- Analysis (front end)**

**- Intermediate Code generation:** Refers symbol table for knowing how much and what type of run-time is allocated and table helps in adding temporary variable information.

**2- Synthesis(back end)**

**- Code Optimization:** Uses information present in the symbol table for machine-dependent optimization.

**- Target Code generation:** Generates code by using address information of identifier present in the table.

## Symbol Table

# Operations:

A symbol table, either linear or hash, should provide the following operations.

| Operation | Function |
|---|---|
| allocate | to allocate a new empty symbol table |
| free | to remove all entries and free storage of symbol table |
| lookup | to search for a name and return pointer to its entry |
| insert | to insert a name in a symbol table and return a pointer to its entry |
| set_attribute | to associate an attribute with a given entry |
| get_attribute | to get an attribute associated with a given entry |

## Error handling

- Another activity that occurs across several phases is error handling.
- Most error handling is done in the analysis (front end)phase.
- It is used by various phases of the compiler as follows:-
  - The scanner keeps an eye out for stray tokens.
  - The syntax analysis phase reports invalid combinations of tokens.
  - The semantic analysis phase reports type errors and the like.

# Error handling

Sometimes these are unhanded fatal errors stop the entire process, while others are less serious and can be circumvented so the compiler can continue.

## Functions of Error Handler:

- Error Detection
- Error Report
- Error Recovery

# Error handling

There are three types of error: logic, run-time and compile-time error:

**1) Logic errors**

- Logic errors occur when programs operate incorrectly but do not terminate abnormally (or crash).
- Unexpected or undesired outputs or other behavior may result from a logic error, even if it is not immediately recognized as such.

# Error handling

## 2) Compile-time error

- Compile-time errors rise at compile-time, before the execution of the program.
- Syntax error or missing file reference that prevents the program from successfully compiling is an example of this.

# Error handling

## 3) Run-time error

- A run-time error is an error that takes place during the execution of a program and usually happens because of adverse system parameters or invalid input data.
- The lack of sufficient memory to run an application or a memory conflict with another program and logical error is an example of this.