What are translators?

- Compiler
- Interpreter
- Mixed

Programming Language Examples?

- Bill Kinnersley has site on history of languages: http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm

A Short History of Compilers

- First, there was nothing.
- Then, there was machine code.
- Then, there were assembly languages.
- Then, there came higher-level languages.
- Then, fourth-generation languages.
- Lastly, fifth-generation languages.
Why Study Compilers?

- Excellent software-engineering example --- theory meets practice.
- Essential software tool.
- Influences hardware design, RISC vs. CISC.
- Tools (mostly “optimization”) for enhancing software reliability and security.

John Backus

- “I’m a terribly unscholarly person, and lazy. That was my motivating force in most of what I did, was how to avoid work.”
- Led the team that developed widely used high-level programming language (FORTRAN)
- Well known for Backus-Naur Form (BNF)

From Description to Implementation

- **Lexical analysis:** Identify logical pieces of description
- **Syntax analysis:** Identify how those pieces relate to each other.
- **Semantic analysis:** Identify the meaning of those relations.
- **IR Optimization:** Simplify the intended structure.
- **Code Generation:** Fabricate the structure.
- **Optimization:** Improve the resulting structure.
source program
  Preprocessor
modified source program
  Compiler
  target assembly program
  Assembler
  relocatable machine code
  Linker/Loader
  library files
  relocatable object files
  target machine code

Figure 3.5: A language-processing system
Compiler-Construction Tools

- Generators for these phases
  - Scanner, parser, syntax-directed, code-gens, etc.
- We won’t cover these

Language Basics

- Environments and States
- Block Structure
- Explicit Access Control
- Dynamic Scope
- Parameter Passing Mechanisms
- Aliasing
Environments and States

• \( x = y + 1 \)

![Diagram showing environment and state](image)

Figure 1.8: Two-stage mapping from names to values

Static vs. Dynamic Binding

```c
... int i; /* global i */
... void f(...) {
  int i; /* local i */
  ... 1 = 3; /* use of local i */
  ...
  x = i + 1; /* use of global i */
...}
```

Figure 1.9: Two declarations of the name \( i \)

Static vs. Dynamic Scope

• What is static vs. dynamic scope?
  ```c
  int x = 0;
  int f() { return x; }
  int g() { int x = 1; return f(); }
  ```

• What kind of scope is C?
Dynamic Scope Example

```cpp
#define a (x+1)
int x = 2;

void b() { int x = 1; printf("%d\n", a); }
void c() { printf("%d\n", a); }
void main() { b(); c(); }
```

Figure 1.12: A macro whose names must be scoped dynamically

- What is another example of dynamic scope?

Explicit Access Control

- Public
- Private
- Protected
Parameter Passing Mechanisms

• Pass by Value
• Pass by Reference
• Pass by Name

Aliasing

• What is this?
• Where do we see this?
• Ex.
  public class test {
      public static void main (String[] args) { 
          int a[] = new int[1];
          q(a,a);
      } 
      public static void q(int x[], int y[]) { 
          y[0]=2;
          x[0]=23;
          System.out.println(y[0]);
      } 
  }

Your First Milestone

• Learn a new language
• Get a Makefile working
• Write a Milestone report
• Review Milestone 1...