CS480
Translators

Introduction to Compilers
Chap. 1
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](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.

SQL constraint-based lang.
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.
**Figure 1.5: A language-processing system**
flip3 ~ 29%
cat test1.c
#include<limits.h>
#include<stdio.h>

int main(void) {
    int x;
    x=INT_MAX-50;
    printf("%f\n", (float)x);
    return(0);
}
flip3 ~ 30%
extern char *ctermid (char *__s) __attribute__ ((nothrow));
# 908 "/usr/include/stdio.h" 3 4
extern void flockfile (FILE *__stream) __attribute__ ((nothrow));

extern int ftrylockfile (FILE *__stream) __attribute__ ((nothrow));

extern void funlockfile (FILE *__stream) __attribute__ ((nothrow));
# 938 "/usr/include/stdio.h" 3 4

# 3 "test1.c" 2

int main(void) {
    int x;

    x=2147483647 -50;

    printf("%f\n", (float)x);

    return(0);
}
flip3 ~ 33%
.globl main
.type main, @function

main:
.LFB0:
    .cfi_startproc
    pushq %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq %rsp, %rbp
    .cfi_def_cfa_register 6
    subq $16, %rsp
    movl $2147483597, -4(%rbp)
    cvtsi2ss -4(%rbp), %xmm0
    unpcklps %xmm0, %xmm0
    cvtps2pd %xmm0, %xmm0
    movl $.LC0, %eax
    movq %rax, %rdi
    movl $1, %eax
    call printf
    movl $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc

.LFE0:
.size main, .-main
.ident "GCC: (GNU) 4.4.6 20110731 (Red Hat 4.4.6-3)"
.section .note.GNU-stack,"",@progbits
```
flip3 ~ 79% gcc -c test.s
flip3 ~ 80% more test.o

******* test.o: Not a text file *******

flip3 ~ 81% cat test.o
ELF>H@@
UHáHiÇEűűý6*EűÀAZÀ,HČ,è,ÉÁ%f
GCC: (GNU) 4.4.6 20110731 (Red Hat 4.4.6-3)zRx
n
.symtab.strtab.shstrtab.rela.text.data.bss.rodata.comment.note.GNU-stack
.rela.eh_frame @30
    0tlt9oX-B¥W"RØ
    ṛā
    3中级

yyyy yyyy flip3 ~ 82% VT102VT102VT102VT102VT102
VT102VT102VT102VT102VT102: Command not found.
flip3 ~ 83% gcc test.o
flip3 ~ 84% a.out
2147483648.000000
flip3 ~ 85%

Connected to flop.engr.oregonstate.edu
```

- `-C` (C, o, obj) files
- `-l` linker

OSU Oregon State University
Figure 1.6: Phases of a compiler
Figure 1.7: Translation of an assignment statement
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

- $x = y + 1$

Figure 1.8: Two-stage mapping from names to values

- Static/dynamic binding
- Environments
- Names
- Locations (variables)
- Values
- Static/dynamic binding
- Static/global
- Dynamic/all other
- Static const
- Dynamic all other
Static vs. Dynamic Binding

```c
...  
int i;  /* global i */
...  
void f(...) {
    int i;  /* local i */
    ...
    i = 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?
  ```
  int x = 0;
  int f() { return x; }
  int g() { int x = 1; return f(); }
  ```

• What kind of scope is C?
  Perl: dynamic
  Static
main() {
    int a = 1;
    int b = 1;
    {
        int b = 2;
        {
            int a = 3;
            cout << a << b;
        }
    {
        int b = 4;
        cout << a << b;
    }
    cout << a << b;
}
    cout << a << b;
}

Figure 1.10: Blocks in a C++ program
Dynamic Scope Example

```c
#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

? Algol
Aliasing

• What is this?
• Where do we see this?
• Ex.

```java
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...