Introduction:
Compiler Design

CSC532

Outline

• Course related info
• What are compilers?
• Why learning?
• Introductory Anatomy of Compiler
CS432 Syllabus

- **Instructor**: Box Leangsuksun [www.latech.edu/~box](http://www.latech.edu/~box)
- **URL**: [http://www.latech.edu/~box/compiler](http://www.latech.edu/~box/compiler)
- **Office**: Nethken 237
- **Office Hours**
  - M-W-F 8am-10:00am.
  - T-Th 2:00pm-4:00pm or (STOP BY ANYTIME, I am here for you!)
- **Phone**: 257-3291
- **e-mail**: [box@latech.edu](mailto:box@latech.edu)
- **.NET passport**: [leangsuksun@hotmail.com](mailto:leangsuksun@hotmail.com) (IM, VOIP SIP phone)
- **Grader**: Sonal Dekhan, ssd002@latech.edu

CS432 Course Books and ref

- **Textbook**: “Dragon” book
- **Online materials** will be provided later
- **Other useful info (books)** Tiger book
Evaluation

• 3-5 Programming Projects 40%
• 3-4 Homework Assignments 10%
• Midterm examination 20% January 21
• Final Exam (comp.) 25% February 27
• Class Participation 5%

• Grad students (105 based score)
  – Term paper 5%

Questions?

– General Application developer
  • Desktop
  • Web based
– Compiler writer
– OS developer
– Embedded system/application developer
What we plan to cover (19/21)

- **Lexical Analysis** \(\text{token, valid symbol/string}\)
- **Syntax Analysis** \(\text{correct syntax/grammar}\)
- **Semantic Analysis** \(\text{valid operation, type}\)
- **Code Generation**
- **Intro Optimization**
- **Automatic Compiler Generation tools**
- **XML parsers and tools**

Compiler design -> computer language engineering

- **Lexical Analysis** \(\text{token, valid symbol/string}\)
- **Syntax Analysis** \(\text{correct syntax/grammar}\)
- **Semantic Analysis** \(\text{valid operation, type}\)
- **Code Generation** \(\text{target language output}\)
- **Intro Optimization** \(\text{better performance code}\)
Projects

• Design and implement a simple language compiler
  – Tokenizer/Symbol table
  – Parsers
  – Code Generator
  – Code optimizer ??

• 2-3 person team (Extreme Programming, intro)

• Language, tools and Environments
  – Java/C/C++ (preferred java)
  – Windows/Linux
  – Build tools, e.g. Ant, Make etc…

What are compilers?

• A program that takes a source program in one language and translates it into a target language.
Compiler development

- programming languages,
- machine architecture,
- language theory
- algorithms
- software engineering.

(Programming) Languages

- Input and output
- Application files

Objectives:
- How to give instructions to a computer
- How to make the computer carry out the instructions efficiently
(Programming) Languages

• Natural languages?
  – English?
  – "Drive straight 5 miles, turn left at the right"
  – We are getting there but not yet!!

• Use a programming language
  – Java, C, C++, Fortran, Basic, C#

Programming Languages

Should have the following properties
• unambiguous
• precise
• concise
• expressive
• a high-level (lot of abstractions)
– But Natural Languages are not
Machine Architectures

• 16 or 32 or 64 bits
• Vector/pipeline, Hyper-threading
• Specially instructions
• Parallel computing architecture
  – SIMD
  – MIMD
  – Distributed Computing
    • NUMA
  – How to give instructions to a computer
  – How to make the computer carryout the instructions efficiently

Others (compiler development)

• language theory
• algorithms
• software engineering
Others (compiler development)

- language theory
- algorithms
- software engineering

Anatomy of a compiler
Phases of A Compiler

Phases of A Compiler (CONTD.)

- Example source statement:
  \[
  \text{position} := \text{initial} + \text{rate} \times 60
  \]

- After lexical analysis:
  \[
  \text{id1} := \text{id2} + \text{id3} \times 60
  \]

- 3 symbols are entered in the symbol table:
  1. position
  2. initial
  3. Rate

- After syntax analysis:
Phases of A Compiler (CONTD.)

After syntax analysis:

```plaintext
position := initial + rate * 60
```

CONTD.

• After semantic analysis:

```plaintext
position := initial + rate * 60
```
CONTD.

• After intermediate code generation:
  temp1 := inttoreal(60)
  temp2 := id3 * temp1
  temp3 := id2 + temp2
  id1 := temp3

• After code optimization:
  temp1 := id3 * 60.0
  id1 := id2 + temp1

• After final code generation:
  MOVF id3, R2
  MULF #60.0, R2
  MOVF id2, R1
  ADDF R2, R1
  MOVF R1, id1

Example (input program)*

from Dr. Amarasinghe's slide (MIT)

```c
int expr(int n)
{
    int d;
    d = 4 * n * n * (n + 1) * (n + 1);
    return d;
}
```
Example (Output assembly code)* from Dr. Amarasinghe’s slide (MIT)

```
lda $30,-32($30)
stq $26,0($30)
stq $15,8($30)
bis $30,$30,$15
bis $16,$16,$2
stl $1,16($15)
st $1,16($15)
st $1,16($15)
sts $f1,24($15)
ldi $0,24($15)
bis $9,25,12
s4addq $2,0,3
ldi $4,16($15)
mull $4,33,$2
ldi $3,16($15)
addq $3,1,1,2
mull $2,4,3
ldi $3,16($15)
addq $3,1,1,2
mull $2,4,3
sti $2,20($15)
ldi $0,20($15)
br $31,33
$33:
bis $15,$15,$30
ldq $26,0($30)
ldq $15,8($30)
addq $30,32,$30
ret $31,($26),1
```

Example of a command line for Compile to assembler using cc –S

```
$33:
bis $15,$15,$30
ldq $26,0($30)
ldq $15,8($30)
addq $30,32,$30
ret $31,($26),1
```

Example of a command line for Compile to assembler using cc –S

```
$33:
bis $15,$15,$30
ldq $26,0($30)
ldq $15,8($30)
addq $30,32,$30
ret $31,($26),1
```

Example of a command line for Compile to assembler using cc –S

```
$33:
bis $15,$15,$30
ldq $26,0($30)
ldq $15,8($30)
addq $30,32,$30
ret $31,($26),1
```

Let's Optimize... *) from Dr. Amarasinghe’s slide (MIT)

```c
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*y;
    }
    return x;
}
```
int sumcalc(int a, int b, int N)
{
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*y;
    }
    return x;
}
int sumcalc(int a, int b, int N) {
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*y;
    }
    return x;
}
Constant Propagation *) from Dr. Amarasinghe's slide (MIT)

```c
int sumcalc(int a, int b, int N) {
    int i;
    int x, y;
    x = 0;
    y = 0;
    for(i = 0; i <= N; i++) {
        x = x + (4*a/b)*i + (i+1)*(i+1);
        x = x + b*0;
    }
    return x;
}
```

More sample of code optimization

• Can be found at http://web.mit.edu/6.035/www/lectures-2002/lecture_1_files/frame.htm
• from Dr. Amarasinghe’s slide (MIT)
Register Allocation

Local variable \( X \)
Local variable \( Y \)
Local variable \( I \)

\[
\begin{align*}
\text{fp} & \quad \text{fp} \\
\text{Local variable} & \quad \text{Local variable} \\
X & \quad Y \\
\end{align*}
\]

\[
\begin{align*}
\text{fp} & \quad \text{fp} \\
\text{Local variable} & \quad \text{Local variable} \\
X & \quad Y \\
\end{align*}
\]

\[
\begin{align*}
\$t9 & = X \\
\$t8 & = t \\
\$t7 & = u \\
\$t6 & = v \\
\$t5 & = i \\
\end{align*}
\]
int sumcalc(int a, int b, int N)
{
    int i;
    int x, t, u, v;
    x = 0;
    u = ((a<<2)/b);
    v = 0;
    for(i = 0; i <= N; i++) {
        t = i+1;
        x = x + v + t*t;
        v = v + u;
    }
    return x;
}

test:
subu $fp, 16
add $t9, zero, zero # x = 0
sll $t0, $a0, 2 # a<<2
div $t7, $t0, $a1 # u = (a<<2)/b
add $t6, zero, zero # v = 0
add $t5, zero, zero # i = 0
lab1: # for(i=0;i<N; i++)
    addui$t8, $t5, 1 # t = i+1
    mul $t0, $t8, $t8 # t*t
    addu $t1, $t0, $t6 # v + t*t
    addu $t9, t9, $t1 # x = x + v + t*t
    addu $6, $6, $7 # v = v + u
addui$t5, $t5, 1 # i = i+1
ble $t5, $a3, lab1
addu $v0, $t9, zero
addu $fp, 16
b $ra
### Unoptimized Code

```assembly
test:
  subu $fp, 16
  sw zero, 0($fp)
  sw zero, 4($fp)
  sw zero, 8($fp)
lab1:
  addi $t8, $t5, 1
  mul $t0, $t8, $t8
  addu $t1, $t0, $t6
  addu $t9, $t9, $t1
  addu $v0, $t9, zero
  addu $fp, 16
  b $ra
```

### Optimized Code

```assembly
test:
  subu $fp, 16
  add $t9, zero, zero
  sll $t0, $a0, 2
  div $t7, $t0, $a1
  add $t6, zero, zero
  add $t5, zero, zero
lab1:
  addui $t8, $t5, 1
  mul $t0, $a0, 4
  div $t1, $t0, $a1
  lw $t2, 8($fp)
  mul $t3, $t1, $t2
  lw $t4, 8($fp)
  addui $t4, $t4, 1
  lw $t5, 8($fp)
  addui $t5, $t5, 1
  mul $t6, $t4, $t5
  addu $t7, $t3, $t6
  lw $t8, 0($fp)
  add $t8, $t7, $t8
  sw $t8, 0($fp)
  lw $t0, 4($fp)
  mul $t1, $t0, $a1
  lw $t2, 0($fp)
  add $t2, $t2, $t1
  sw $t2, 0($fp)
  lw $t0, 8($fp)
  addui $t0, $t0, 1
  sw $t0, 8($fp)
  ble $t0, $a3, lab1
  lw $v0, 0($fp)
  addu $fp, 16
  b $ra
```

4*ld/st + 2*add/sub + br + 6*add/sub + shift + div + br + N*(9*ld/st + 6*add/sub + 4*mul + div + br) = 7 + N*21

Execution time = 43 sec

6*add/sub + shift + div + br + N*(5*add/sub + mul + br) = 9 + N*7

Execution time = 17 sec

---

**Compilers Optimize Programs for...**

- Performance/Speed
- Code Size
- Power Consumption
- Fast/Efficient Compilation
- Security/Reliability
- Debugging