Class Information

INFORMATION and REMINDERS

• Sample solution for homework 1 has been posted.

• First project has been posted. Deadline: Friday, March 24 (week after spring break) at 11:59pm.

• Homework 4 will come out on Saturday.
Project 1: Overview

### Compiler

- **Input:** tinyL program
- **Example:** `compile test1`
- **Output:** tinyL.out
- **Output always:** "tinyL.out"

### Optimizer

- **Input:** RISC machine code
- **Example:** `optimize < tinyL.out`
- **Output:** RISC machine code
- **Output to stdout**

### Virtual Machine

- **Input:** RISC machine code
- **Example:** tinyL.out
- **Output:**
  - Input and output of execution
Project 1: Dead Code Elimination

Goal: Identify instructions that do not contribute to the input/output behavior of the program. These instructions are considered “dead” and can be eliminated.

Example:

loadI 1024 => r0 // base address
...
loadI 5 => r1
loadI 7 => r2
loadI 2 => r3
add r1, r2 => r4
mult r1, r2 => r5
storeAI r5 => r0, 0 // variable ‘a’ has offset 0
outputAI r0, 0

Code after ‘dead’ instructions have been eliminated:

loadI 1024 => r0 // base address
...
loadI 5 => r1
loadI 7 => r2
mult r1, r2 => r5
storeAI r5 => r0, 0 // variable ‘a’ has offset 0
outputAI r0, 0
Imperative Programming Languages

Imperative:
Sequence of state-changing actions.

- Manipulate an abstract machine with:
  1. Variables naming memory locations
  2. Arithmetic and logical operations
  3. Reference, evaluate, assign operations
  4. Explicit control flow statements
- Key operations: Assignment and “Goto”
- Fits the von Neumann architecture closely

Von Neumann Architecture
Run-time storage organization

Typical memory layout

Logical Address Space

<table>
<thead>
<tr>
<th>Code</th>
<th>Static</th>
<th>Heap</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>high</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The classical scheme

- allows both stack and heap maximal freedom
- code and static may be separate or intermingled

Will talk about this in more detail in a later lecture!
C: An Imperative Programming Language

Expressions: include procedure and function calls and assignments, and thus can have side-effects

Control Structures:

• if statements, with and without else clauses

• loops, with break and continue exits

  while ( <expr> ) <stmt>
  do <stmt> while ( <expr> )
  for ( <expr> ; <expr> ; <expr> ) <stmt>

• switch statements

• goto with labelled branch targets
C Examples

```c
while (( c = getchar()) != EOF ) putchar(c);
```

```c
for ( i = 0 ; s[i] == ' ' ; i++ );
```

```c
for ( i = 0 ; i < n ; i++ ) {
    if ( a[i] < 0 ) continue; /*skip neg elems*/
    ...
}
```

```c
c = getchar();
switch(c) {
    case '0': case '1': case '2': case '3':
    case '4': case '5': case '6': case '7':
    case '8': case '9':
        digit[c-'0']++;
        break;
    case ' ': case '
': case 't':
        delim++;
        break;
    ...
}
```
Data Types in C

• Primitives: char, int, float, double
  no Boolean—any nonzero value is true

• Aggregates: arrays, structures

  char a[10], b[2][10];

  struct rectangle {
    struct point p1;
    struct point p2;
  }

• Enumerations: collection of sequenced values

• Pointers:
  &i    address of i
  *p    dereferenced value of p
  p+1   pointer arithmetic

  int *p, i;
p = &i;
*p = *p + 1;
<table>
<thead>
<tr>
<th>C</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic types:</td>
<td>Primitive types:</td>
</tr>
<tr>
<td>int, double, char</td>
<td>int, double, char, boolean</td>
</tr>
<tr>
<td><strong>Pointer (to a value)</strong></td>
<td><strong>Reference (to an object)</strong></td>
</tr>
<tr>
<td>Aggregates:</td>
<td>Aggregates:</td>
</tr>
<tr>
<td>array, struct</td>
<td>array, object (class)</td>
</tr>
<tr>
<td>Control flow:</td>
<td>Control flow</td>
</tr>
<tr>
<td>if-else, switch, while, break, continue, for, return, goto</td>
<td>if-else, switch, while, break, continue, for, return</td>
</tr>
<tr>
<td>Logic operators:</td>
<td>Logic operators:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical comparisons:</td>
<td>Logical comparisons:</td>
</tr>
<tr>
<td>==, !=</td>
<td>==, !=</td>
</tr>
<tr>
<td>Numeric comparisons:</td>
<td>Numeric comparisons:</td>
</tr>
<tr>
<td>&lt;&gt;, &lt;=, &gt;=</td>
<td>&lt;&gt;, &lt;=, &gt;=</td>
</tr>
<tr>
<td>string as char * array</td>
<td><strong>String</strong> as an object</td>
</tr>
</tbody>
</table>
Things to do:

Work on the project!

Read Scott: Chap. 3.1 - 3.4; 8.1 - 8.2; ALSU Chap. 7.1 - 7.3

Next time:

• Example: Singly-linked list implementation
• Procedure abstractions; run time stack; scoping.