CPSC 213
Introduction to Computer Systems
Unit 1g
Dynamic Control Flow
Polymorphism and Switch Statements

Polymorphism

› Invoking a method on an object in Java
  • variable that stores the object has a static type
  • object-reference is dynamic and so is its type
  • object's type must be implemented by the type of the receiving variable
  • but object's type may override methods of the base type

› Polymorphic Dispatch
  • target method address depends on the type of the referenced object
  • one call site can invoke different methods at different times

Polymorphic Dispatch

› Method address is determined dynamically
  • compiler cannot hardcode target address in procedure call
  • instead, compiler generates code to look up procedure address at runtime
  • address is stored in memory in the object's class jump table

› Class Jump table
  • every class is represented by class object
  • the class object stores the class's jump table
  • the jump table stores the address of every method implemented by the class
  • objects store a pointer to their class object

Static and dynamic of method invocation

• address of jump table is determined dynamically
  • method's offset into jump table is determined statically

Example of Java Dispatch

Pseudo code

in Java

• static methods are class methods
  • invoked by naming the class, not an object

in C

• specify procedure name

What is the difference between these two C snippets?

• A[2] calls foo, but (1) does not
• B[2] (1) is not valid C
• B[1] jumps to foo using a dynamic address and (2) a static address
• B[3] They both call foo using dynamic addresses
• B[4] They both call foo using static addresses

Now, implement proc() and foos() assembly code
Switch Statements

What happens when this code is compiled and run?

• [A] It does not compile
• [B] For any value of input it generates an error
• [C] If input is 1 it prints "bar 2" and it does other things for other values
• [D] If input is 1 it prints "bar 1" and it does other things for other values

Human vs Compiler

Benefits for humans:
• the syntax models a common idiom: choosing one computation from a set

But, switch statements have interesting restrictions:
• case labels must be static, cardinal values
  • a cardinal value is a number that specifies a position relative to the beginning of an ordered set
  • for example, integers are cardinal values, but strings are not
• case labels must be compared for equality to a single dynamic expression
  • some languages permit the expression to be an inequality

Do these restrictions benefit humans?
• have you ever wanted to do something like this?

Why Compilers like Switch Statements

• Notice what we have
  • switch condition evaluates to a number
  • each case has a distinct number
  • and so, the implementation has a simplified form
  • build a table with the address of every case arm, indexed by case value
  • switch by indexing into the table and jumping to matching case arm

For example:

Implementation

- done:     ld   $j, r0
- default:  ld   $j, r0
- case20:   ld   $j, r0
- break

- goto
  - jump-taken condition
  - target address stored in a register
  - we already introduced this instruction, but used it for static procedure calls

Dynamic Jumps

• Jump base+offset
  • Jump target address stored in a register
  • We already introduced this instruction, but used it for static procedure calls

Indirect jumps

• Jump target address stored in memory
  • Base plus displacement and indexed modes for memory access

Summary

- Static vs Dynamic flow control
  • static if jump target is known by compiler
  • dynamic for polymorphic dispatch, function pointers, and switch statements

Polymorphic Dispatch in Java

- invoking a method on an object in java
  • method address depends on object type, which is not known statically
  • object has pointer to class object; class object contains method jump table
  • procedure call is an indirect jump - i.e., target address in memory

Function Pointers in C

- a variable that stores the address of a procedure
  • used to implement dynamic procedure call, similar to polymorphic dispatch

Switch Statements

- syntax restricted so that they can be implemented with jump table
  • jump-table implementation running time is independent of the number of case labels
  • but, only works if case label values are reasonably dense

Question 2

- What happens when this code is compiled and run?

```c
switch (input) {
    case 1: bar (1); break;
    case 2: bar (2); break;
    case 3: bar (3); break;
    default: printf ("bar %d!");
}
```

Question 3

- Which implements proc(input) (input + 1);

```c
void foo (int i) { printf ("foo %d!"); } # i
void bar (int i) { printf ("bar %d!"); } # i

int main (int argc, char* argv[]) {
    int input;
    if (argc==2) {
        input = atoi (argv[1]);
        proc[input] (input+1);
    }
}
```