CS 164 Programming Languages and Compilers **Spring 2019**

Discussion Week of 4/22: Inheritance, Exceptions, and Code Generation

Instructor: Paul N. Hilfinger GSIs: Vivant Sakore, Nikhil Athreya

1 Inheritance

1. Recall the *stub* trick from lecture, an implementation strategy for multiple inheritance. Suppose we have the following:

```
class Base {
public:
    int foo;
    virtual int f() {
        std::cout << this << std::endl; return this->foo;
    }
};

class Base2 {
public:
    int bar;
    virtual int g() {
        std::cout << this << std::endl; return this->bar;
    }
};

class Derived : public Base, public Base2 {
public:
    int baz;
    virtual int h() {
        std::cout << this << std::endl; return this->baz;
    }
};
```

   To overcome this issue, compilers will move the `this` pointer so that each method sees what it expects. For simplicity, they actually create stub methods that will move the `this` pointer before calling the original method. For example, the compiler may generate

```
Base2::g'() { move_this_pointer(); g(); }
```

   (a) Roughly sketch the class layout of the above classes (using the same 4-byte offset convention from the inheritance lecture). For `Derived`, assume `h()` is stored inside its own version of `Base`'s vtable.

(b) What will the following print (assuming there is no padding)?

```
Derived a;
std::cout << &a << std::endl; // Suppose this prints 0x10.
a.f();
a.g();
a.h();
```

Answer:

```
a.
   0 |   (Base vtable pointer; vtable contains f())
   4 |   int foo

   0 |   (Base2 vtable pointer; vtable contains g())
   4 |   int bar

   0 |   (Base vtable pointer; vtable contains f(), h(), and g'();
         g'() is a stub that increments this by 8, then calls g())
   4 |   int foo
   8 |   (Base2 vtable pointer; vtable contains g())
  12 |   int bar
  16 |   int baz

b.
    0x10
    0x18
    0x10
```

2 Exceptions and Code Generation

In this question we'll look at how the setjmp/longjmp exception mechanism can be implemented. Consider a 32-bit computer architecture with 5 registers (`r0`, ..., `r4`): `r0-2` are used for function parameters, `r3` is used for the return address, and `r4` is used to store the return value. Implement `setjmp(jmp_buf*)` and `longjmp(jmp_buf*, int)` in assembly (AT&T syntax). Assume that the `jmp_buf` is large enough, and that the second argument to longjmp is never 0.

Answer:

```
setjmp:
    movl r0, 0(r0)   ; Hint: setjmp() cannot modify any registers until
```

```
    movl r1, 4(r0)   ; it has saved them.
    movl r2, 8(r0)   ;
    movl r3, 12(r0)  ;
    xorl r4, r4      ; Set the return value register to 0.
    retl             ; Return to the address stored in r3.

longjmp:
    movl r1, r4      ; Set the return value up front before r1 disappears.
    movl 4(r0), r1   ; Skip recovering r0 to preserve the jmp_buf pointer.
    movl 8(r0), r2   ;
    movl 12(r0), r3  ; Recover the return address passed into setjmp().
    movl 0(r0), r0   ; Finally, recover r0. We handled the others already.
    retl             ; Return "again" from setjmp() by jumping to (r3).
```

1. Is it possible to longjmp to the same `jmp_buf` multiple times?

2. Write a program which throws an exception, handles it, and returns to the exception site from the exception handler using setjmp/longjmp.

   Answer: 1. Yes
   2.

   ```
   void f() {
       jmp_buf jb1, jb2;
       if (setjmp(&jb1) == 0) {
           if (setjmp(&jb2) == 0) {
               longjmp(&jb1, 1);
           } else {
               // return to exception site
           }
       } else {
           longjmp(&jb2, 2);
       }
   }
   ```