Assignment 3

MiniJava to MicroIR

What have we done so far ...

1. Wrote a MacroJava to MiniJava translator (parsed valid programs else gave an error).
2. Took the MiniJava programs obtained in the previous step and type checked them (gave an error message if programs did not type check)

Next Steps

At this stage, we have valid MiniJava programs. We now translate/convert the codes in MiniJava to MicroIR (an intermediate representation for MiniJava).

MicroIR exposes the runtime model:

- Integer is of size 4
- Field and Method references, require accessing the address of the object.
- “new” operator is replaced by HALLOCATE

Now, let’s take a look at some of the features of MicroIR.

MicroIR

procedures:

If we have a procedure

ProcLabel [5] Stmt

then it takes five arguments, called (temp 0), (temp 1), ..., (temp 4). Other temporaries can be used without declaration and treated as locals. It executes the statement Stmt.

arrays:

The index of an array is a multiple of 4, starting at 0.

NOOP:

Does nothing.

ERROR:

Terminates the program execution with an error message.

MicroIR

CJUMP Temp Label:

If Temp evaluates to 1, then continue with the next statement, otherwise jump to Label.

HSTORE Temp_1 IntegerLiteral Temp_2:

Temp_1 evaluates to an address

IntegerLiteral is an offset

Temp_2 evaluates to the value that should be stored.

HLOAD Temp_1 Temp_2 IntegerLiteral:

Temp_1 is the temporary into which a value should be loaded

Temp_2 evaluates to an address

IntegerLiteral is an offset.

MicroIR

NE: is "!="

LE: is "<="

HALLOCATE SimpleExp:

> SimpleExp evaluates to an integer, and that corresponding amount of heap space is allocated, and the return address of that is returned as the result.

Now, let's have a look at the MicroIR grammar.

Some Questions

```
class A {
        int x;
        int y;
        int z;
}
```

Now we have these two statements
A a1 = **new A();**
A a2 = **new A();**

How much memory is required by each of these class A objects ?

Some Questions

```
class A {
        int x;
        int y;
        int z;
}
```

```
class B extends A {
        int xyz;
}
```

Now we have these two statements
A a = new A();
B b = new B();

How much memory is required by each of these class A objects ?

Some Questions

```
class A {
    public int foo() {
        return 1;
    }
}
```

```
class B extends A {
    public int foo() {
        return 2;
    }
}
```

```
A a = new A();
B b = new B();
int x = a.foo();
int y = b.foo();

//x == ?
//y == ?
```

Some Questions

```
class A {
    public int foo() {
        return 1;
    }
}
```

```
class B extends A {
    public int bar() {
        return 2;
    }
}
```

```
A a = new A();
B b = new B();
int x = a.foo();
int y = b.foo();

//x == ?
//y == ?
```

How do we now move from MiniJava to MicroIR

Let’s consider some statements in MiniJava and see how can we achieve them in MicroIR.

A a = new A();

//some space needs to be allocated in the heap - need to know about all the class fields for this particular class

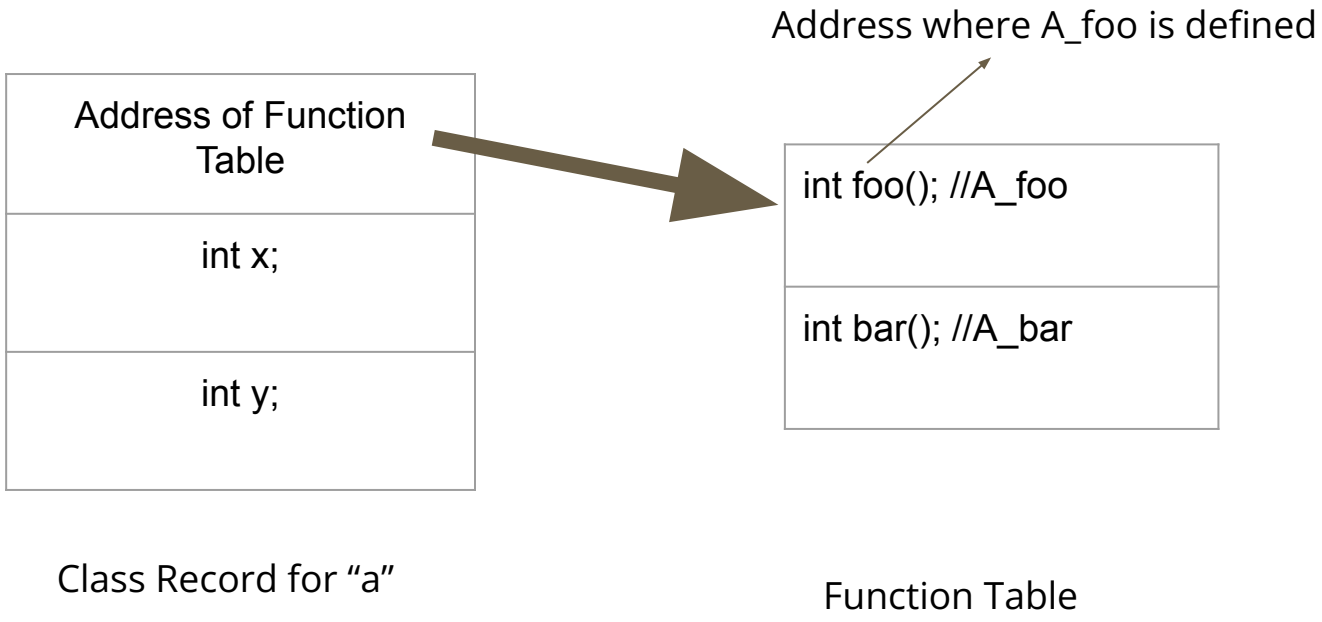
num_aux = this.computeFac(num - 1);

//we need to know which function are we referring to - need to know all the functions that are accessible to this class object

MiniJava class object in MicroIR

```
class A {
        int x;
        int y;
        public int foo() { //..... }
        public int bar() { //..... }
}
```

a = new A();


Address where A_foo is defined
Address of Function Table
int x;
int y;
int foo(); //A_foo
int bar(); //A_bar
Class Record for “a”
Function Table

Example

Let’s now look at Factorial.java (miniJava) and Factorial.microIR.

How to deal with Arrays

- **int[] number;** //if it's a class field,then allocate 4 bytes in the record table else keep a temp variable for it.
- **number = new int[5];** //we need to allocate some space on the heap (a minimum of 5 * 4 bytes is required)
- **number.length** //we somehow need to store the length of the array somewhere
  - This is something that can be done
  - For new int[5], allocate (6 * 4) bytes of memory where the first 4 bytes always store the length of the array.
  - In the test cases, you can assume that all array accesses will be within bounds.