COMP3131/9102: Programming Languages and Compilers

Jingling Xue

School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia

`http://www.cse.unsw.edu.au/~cs3131`

`http://www.cse.unsw.edu.au/~cs9102`

Assignment 4 Feedback

1. Check your marks
2. Read the feedback from our course page

Week 8 (2nd Lecture): Java Byte Code Generation

1. Assignment 5
2. Java bytecode verifier

Assignment 5

- Read the supporting code in Emitter.java
  - 70% of the code generator provided, including:
    - `The generation of field declarations and the class initialiser <clinit> for all global scalar variables in visitProgram. But you are required to modify this method to deal with all array-related declarations and initialisations.`
      - `The generation of the non-arg constructor initialiser <init>`
    - `Various visit methods`
  - You will mostly focus on implementing visitBinaryExpr() and visitUnaryExpr(), where almost all the expressions are translated.
  - Translating statements is straightforward as per their code templates introduced.
  - Translate declarations as described in the Monday lecture

The default constructor <init> Already Done for You

```
// cons.vc
int i = 1;
int main() {
  int x = i;
  return 1;
}
/*
public class cons {

static int i = 1;

cons() { }  // the default constructor:  <init>
            // with the hidden parameter this: <init> (cons this)
public static void main(String argv[]) {
  global vc$ = new cons();
    // Step 1:  vc$ = malloc() for cons
    // Step 2: vc$.<init>, i.e., <init>(vc$)

  int x = i;
  return 1;
  }
```

```
} */
.class public cons
.super java/lang/Object

.field static i I

        ; standard class static initializer
.method static <clinit>()V

        iconst_1
        putstatic cons/i I

        ; set limits used by this method
.limit locals 0
.limit stack 1
        return
.end method

        ; standard constructor initializer
.method public <init>()V
.limit stack 1
.limit locals 1
        aload_0
```

```
        invokespecial java/lang/Object/<init>()V
        return
.end method
.method public static main([Ljava/lang/String;)V
L0:
.var 0 is argv [Ljava/lang/String; from L0 to L1
.var 1 is vc$ Lcons; from L0 to L1
        new cons
        dup
        invokenonvirtual cons/<init>()V
        astore_1
.var 2 is x I from L0 to L1
        getstatic cons/i I
        istore_2
        return
L1:
        return

 ; set limits used by this method
.limit locals 3
.limit stack 2
.end method
```

Class Initialisations <clinit>

- You need to generate the field declaration and initialisation code in <clinit> for global arrays — not provided in the supporting code.
- Done for you for scalar global variables

```
// arrayclinit.vc:

int a[] = {10, 20}; // a global array

int main() {
  int i = a[1];
  return 1;
}
```

```
// Jasmin code:

.class public arrayclinit
.super java/lang/Object

.field static a [I

 ; standard class static initializer
.method static <clinit>()V

 iconst_2      Array size
 newarray int
 dup
 iconst_0  Array index
 bipush 10
```

```
 iastore
 dup
 iconst_1
 bipush 20
 iastore
 putstatic arrayclinit/a [I
```

Make a point to array

```

 ; set limits used by this method
.limit locals 0
.limit stack 4
 return
.end method

 ; standard constructor initializer
.method public <init>()V
```

```
.limit stack 1
.limit locals 1
 aload_0
 invokespecial java/lang/Object/<init>()V
 return
.end method
.method public static main([Ljava/lang/String;)V
L0:
.var 0 is argv [Ljava/lang/String; from L0 to L1
.var 1 is vc$ Larrayclinit; from L0 to L1
 new arrayclinit
 dup
 invokenonvirtual arrayclinit/<init>()V
 astore_1
.var 2 is i I from L0 to L1
```

```
 getstatic arrayclinit/a [I
 iconst_1
 iaload
 istore_2
 return
L1:
 return

 ; set limits used by this method
.limit locals 3
.limit stack 2
.end method
```

Assignment 5: Some Language Issues

- Java byte code requires that
  - all variables be initialised
  - all method be terminated by a return
- Both are not enforced in the VC language
- All test cases used for marking Assignment 5 will satisfy these two restrictions.

ByteCode Verification

- Loop

  ```
  while (true) 1;
  ```

- Bytecode:

  ```
  iconst_1
  pop
  ```

- Removing pop causes a Java.VerifyError:

  ```
  Exception in thread "main" java.lang.VerifyError: (class: x,
  method: foo signature: (V) Inconsistent stack height 1 != 0)
  ```

- JVM Spec:

  > If an instruction can be executed along several different execution paths, the operand stack must have the same depth (§2.6.2) prior to the execution of the instruction, regardless of the path taken.

  ```
  https://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#
  jvms-4.10.1.4
  ```

- This is you are asked to generate a pop, if necessary, for an expression statement in the last lecture.

Reading

- The spec of Assignmen 5

Next Class: DFAs and NFAs (Cont'd)