

CS352 Compilers: Principle and Practice

Fall 2004

Course instructor: Prof. Dennis Brylow

PSO Session on Tuesday November 9, 2004

Teaching Assistant: Mummoorthy Murugesan
mmuruges@cs.purdue.edu

Check the newsgroup for IMPORTANT messages: **purdue.class.cs352**

Before we start:

1. Questions on Project 5?

Agenda:

1. Structure of Tree Package
2. Structure of Translate Package
3. Sample output
4. I've not given "Array Access" example. If you are unclear, let me know. I'll go through it in the class.

1. Tree Package Classes:

BINOP	ESEQ	JUMP
CALL	EXP	LABEL
SEQ	CJUMP	MEM
CONST	MOVE	TEMP
NAME.		

See page 4 for a sample program to print all the nodes: (think about our logic of testing: Brinch Hansen)

What else you'll have in the Tree package?

Try to separate them into Exp & Stm's.

2. Translate Package:

1)

```
/** Accepts a TreeVisitor node v by invoking the method visit. */
public Translate.Exp accept(TreeVisitor v) { return v.visit(this); }
```

What's new?

StrAddExp

2) public Exp visit (Absyn.Program n)

```
{
  for (all the ClassDecs)
    add DataFrag to fragment list for static
    fields
    // Good way to do this: frame.record()
    // method, passed name and static field
    // number (0 this term), as discussed in lecture.
```

```
Prints out:  .data
              .align (see the output)
```

```
for (all the ClassDecs) – vtable output
  add DataFrag to fragment list for vttables
  // Good way to do this:
  // frame.vtable() method, passed name and
  // list of labels for method bodies,
  // as discussed in lecture.
```

Prints out:

```
Main_vtable:
  .data
One_vtable:
  .data
Two_vtable:
  .word      Two.twofun
```

```
for (all the ClassDecs)
  visit(classDec) to generate Expression Trees.
```

```
}
```

3)

```
public Exp visit (Absyn.MethodDec m)
```

```
{   - Allocate a new Frame.Frame for each method.
```

```

- Notice that we still need a varEnv with beginScope() and
  endScope() -- this time, to remember which Frame.Access
  we have allocated for each var.
```

```

- Build up a sequence of SEQ nodes to store tree as you
  build it up:
  - Iterate through visiting Formals
  - Iterate through visiting statement list of method.
```

```

- Add new ProcFrag to list of fragments, containing the
  frame and the expression tree for this methods body.
```

```
    Try to locate this in the output?
```

```
}
```

4) Object runtime layout:

```
class A {
  int x;
  int y;
  int z;
  int foo() {...}
  int bar() {...}
  int baz() {...}
}
```

	offset	RECORD		VTable for class A
	-1	VTable	--->	0 A.foo address
pointer -->	0	x		1 A.bar address
	1	y		2 A.baz address
	2	z		

For arrays:

	-1	length
pointer -->	0	array[0]
	1	array[1]
	2	array[2]
	...	
	len-1	array[length - 1]

5)

How do you print out the tree?

You have a collection of data frags & proc frags.

If any frag is an instance of

Datafrag print

Else

Procfrag print

6) Assignment:

return new Nx(MOVE(var.unEx(), exp.unEx()));

MOVE Accepts Tree.EXP & Tree.EXP

Learning by Example

```

/*****
A sample test case to print all the nodes in Tree.
*****/
class Main
{
    public static void main (String[] a)    {    }
}

class One
{
    int onea;
    public int objectaccess()
    {
        One o;
        return o.onea;
    }
    public int assignment()
    {
        int a;
        a=0;
        a=1;
        return a;
    }
}

class Two extends One
{
    int twoa;

```

```

public int twofun()
{
    twoa=0;
    if(twoa<0)
        twoa=0;
    else
        twoa=1;
    System.out.write(twoa);
    return 0;
}
}

```

Output Analysis:

1. Data Fragments (slice of the output)

Main_vtable:

.data

One_vtable:

.word One.objectaccess

.word One.assignment

.data

Two_vtable:

.word One.objectaccess

.word One.assignment

.word Two.twofun

2. One.objectaccess:

SEQ(

MOVE(

TEMP t34, ← Refers to object 'o'

CONST 0),

MOVE(

TEMP t3, ← t3 is the return value register

ESEQ(

SEQ(

MOVE(

TEMP t35,

TEMP t34),

CJUMP(EQ,

TEMP t35,

CONST 0,

L0, L1)),

← Check for null pointer

```

ESEQ(
  LABEL L1,
  MEM(
    BINOP(PLUS,
      TEMP t35,
      CONST 0))))))    ← Const 0 refers to field "onea"

```

One.assignment:

```

Formals(InReg(t36))
Actuals(InReg(t3))
SEQ(
  SEQ(
    SEQ(
      MOVE(
        TEMP t37, ← t37 refers to i
        CONST 0),
      MOVE(
        TEMP t37,
        CONST 0)),
    MOVE(
      TEMP t37,
      CONST 1)),
  MOVE(
    TEMP t3,
    TEMP t37))

```

System.out.write(twoa):

```

EXP(
  CALL(
    NAME _write,    ← Name write function
    ESEQ(
      SEQ(
        MOVE(
          TEMP t43,
          TEMP t38),
        CJUMP(EQ,
          TEMP t43,    ← Reason about L2
          CONST 0,
          L2, L10)),
      ESEQ(
        LABEL L10,
        MEM(
          BINOP(PLUS,
            TEMP t43,
            CONST 4))))))

```