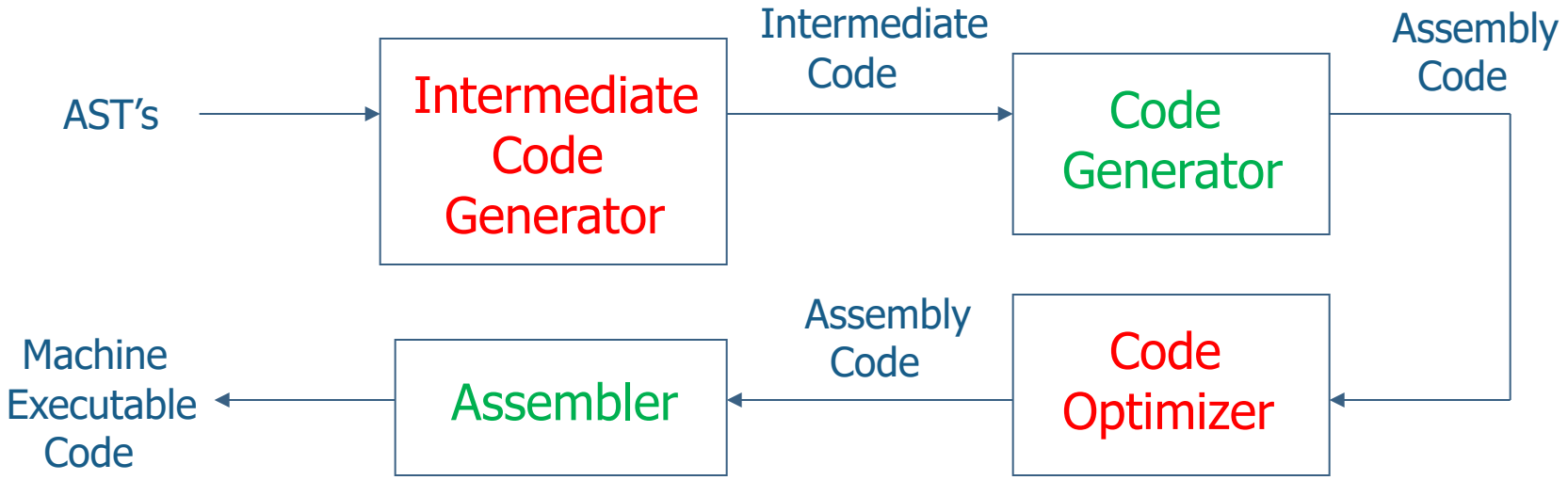
Intermediate Code Generation (I)

CIS*4650 (Winter 2020)

Review

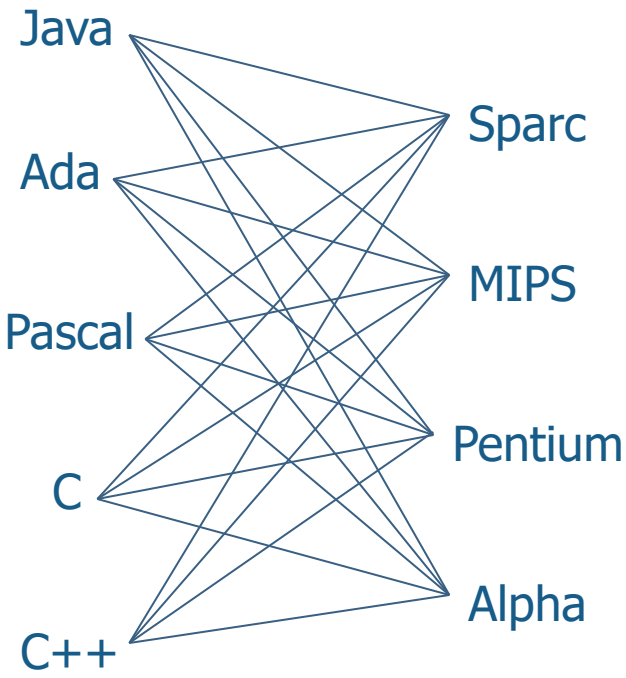
- Code generation: generate executable code for a target machine

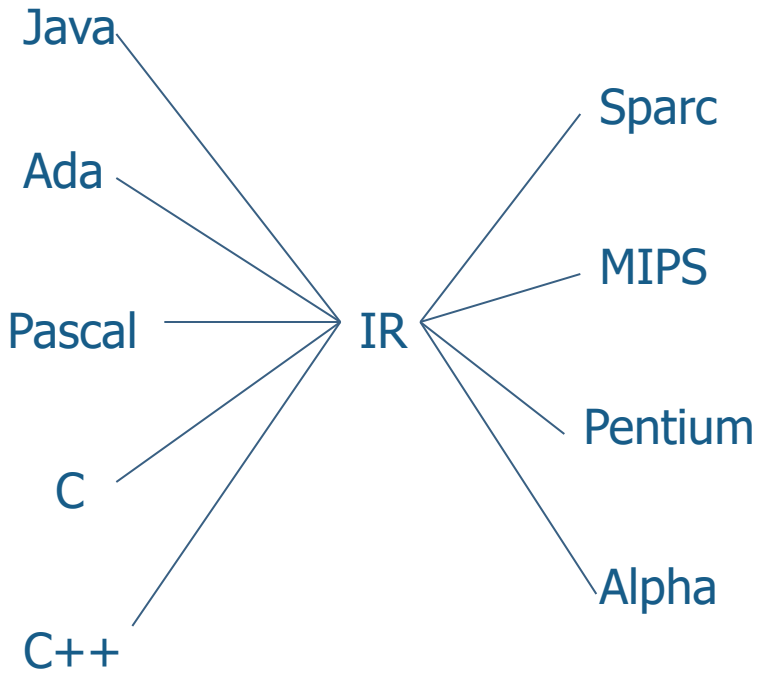


Note that for Checkpoint Three, we will only implement "Code Generator" and run the assembly code on the "TM Simulator". "Intermediate Code Generator" and "Code Optimizer" will be skipped for simplifications.

Intermediate Representation (IR)

- Can be AST's, but linear sequences with jumps are more preferred

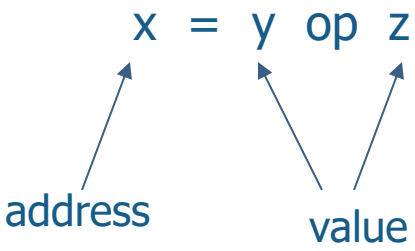




Retargetable Solution

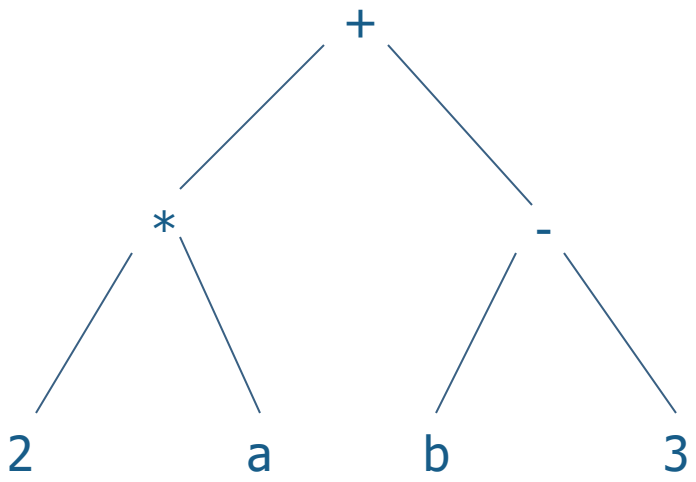
Three-Address Code

- Basic form: maximum of three addresses
  - Compiler needs to generate names for temporary variables



e.g., 2 * a + (b – 3)

t1 = 2 * a
t2 = b – 3
x = t1 + t2



Bigger Example

```
{ Sample Tiny program for
   computing factorial }

read x;
if 0 < x then
   fact := 1;
   repeat
      fact := fact * x;
      x := x – 1;
   until x = 0;
   write fact
end
```

```
read x
t1 = x > 0
if_false t1 goto L1
fact = 1
label L2
t2 = fact * x
fact = t2
t3 = x - 1
x = t3
t4 = x == 0
if_false t4 goto L2
write fact
label L1
halt
```

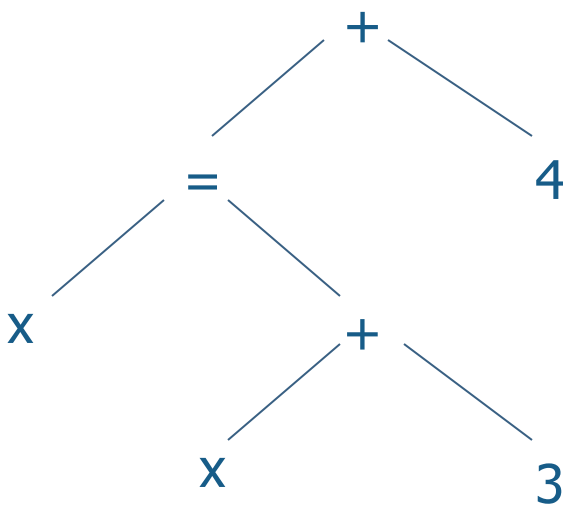
Code Generation for Expressions

Grammar: exp -> id = exp | aexp
aexp -> aexp + factor | factor
factor -> (exp) | num | id

e.g., (x = x + 3) + 4

Intermediate Code:

t1 = x + 3
x = t1
t2 = x + 4



Code Generation for Expressions

```
void genCode( Exp tree ) {        // newtemp() returns a new name such as t1, t2, etc.
    String codestr = “”;
    if( tree != null ) {
        if( tree instanceof OpExp ) {
            genCode( tree.left );
            genCode( tree.right );
            tree.temp = newtemp();      // each node is added with a “temp” string
            codestr += tree.temp + “ = ” + tree.left.temp + “ + ” + tree.right.temp;
            emitCode( codestr );
        } else if( tree instanceof AssignExp ) {
            genCode( tree.rhs );
            tree.temp = tree.lhs.temp;
            codestr += tree.lhs.temp + “ = ” + tree.rhs.temp;
            emitCode( codestr )
        } else if( tree instanceof SimpleVar ) {
            // do nothing
        } else if( tree instanceof IntExp ) {
            // do nothing
        } else
           emitCode(“Error”);
    }
}
```

Array References

- Address vs. value of a variable:

```
t1 = &x + 10
*t1 = 2
```

- Array references:
  - address of a[i+1]:

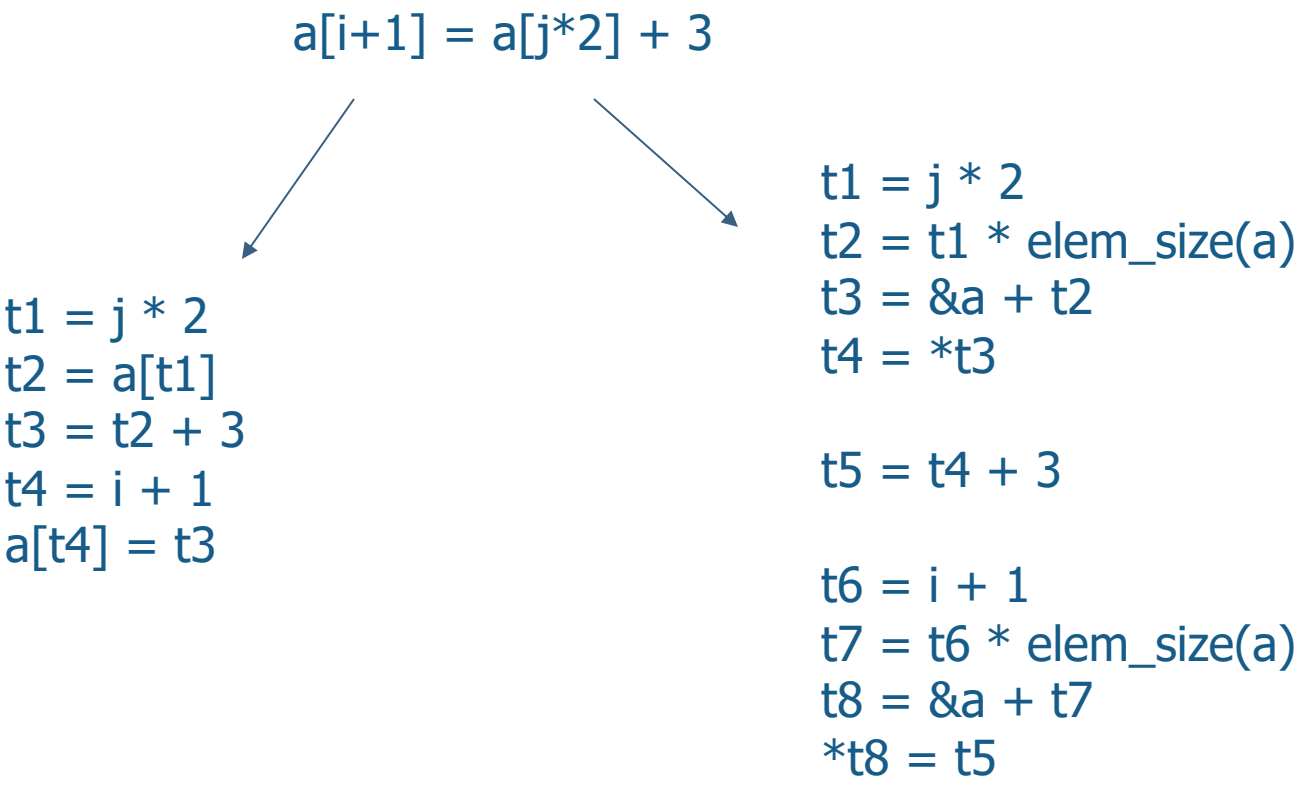
    &a + (i+1)*elem_size(a)
  - fetch the value of an element: t2 = a[t1]

    t = &a + t1*elem_size(a) t2 = *t
  - assign to the address of an element: a[t2] = t1

    t = &a + t2*elem_size(a) *t = t1

Array References

- Different levels of details:

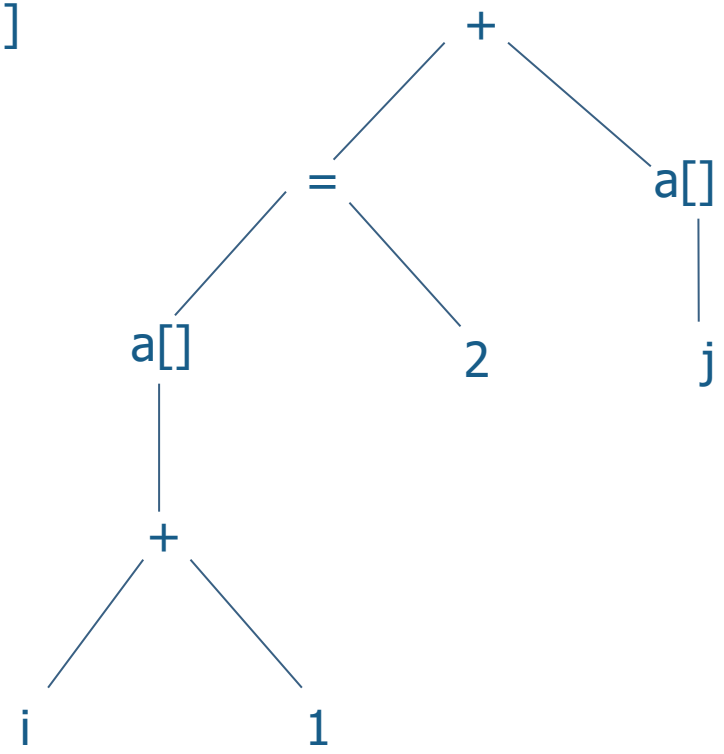


Code Generation for Arrays

Grammar: exp -> subs = exp | aexp
aexp -> aexp + factor | factor
factor -> (exp) | num | subs
subs -> id | id [exp]

e.g., (a[i+1] = 2) + a[j]

Intermediate code:
t1 = i + 1
t2 = t1 * elem_size(a)
t3 = &a + t2
*t3 = 2
t4 = j * elem_size(a)
t5 = &a + t4
t6 = *t3 + *t5



Code Generation for Arrays

```
void genCode( Exp tree ) {
    String codestr = “”;
    if( tree != null ) {
        if( tree instanceof OpExp ) {
            // refer to the related fragment
        } else if( tree instanceof AssignExp ) {
            // refer to the related fragment
        } else if( tree instanceof IndexVar ) {
            // refer to the related fragment
       } else if( tree instanceof SimpleVar ) {
            // do nothing
       } else if( tree instanceof IntExp ) {
            // do nothing
       } else
          emitCode( “Error” );
    }
}
```

```
// code fragment for IndexVar
genCode( tree.index );
String temp = newtemp();
codestr += temp + “ = ” +
    tree.index.temp + “ * elem_size(” +
    tree.name + “)”;
emitCode( codestr );
String temp2 = newtemp();
codestr += temp2 + “ = &” +
    tree.name + “ + ” + temp;
emitCode( codestr );
tree.temp = temp2;
tree.isAddr = true;
```

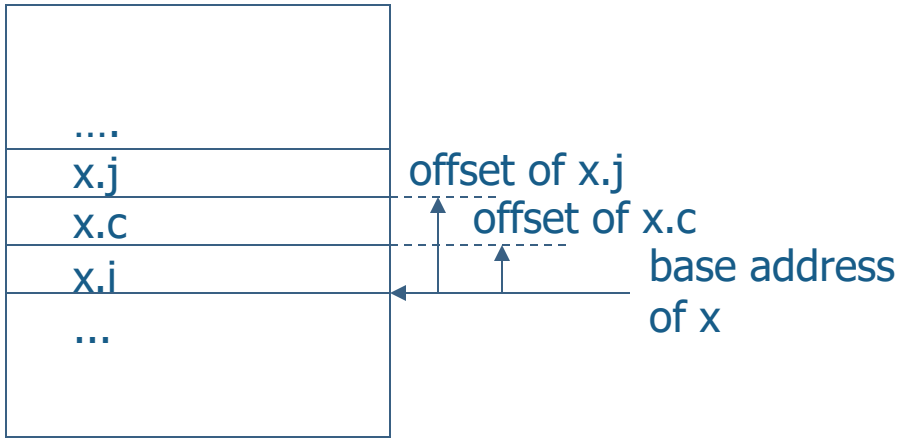
Code Generation for Arrays

```
// code fragment for OpExp
genCode( tree.left );
genCode( tree.right );
tree.temp = newtemp();
codestr += tree.temp +  " = ";
if( tree.left.isAddr )
    codestr += "*" ;
codestr += tree.left.temp + " + ";
if( tree.right.isAddr )
    codestr += "*";
codestr += tree.right.temp;
emitCode( codestr );
```

```
// code fragment for AssignExp
genCode( tree.lhs );
genCode( tree.rhs );
tree.temp = tree.lhs.temp;
tree.isAddr = tree.lhs.isAddr;
if( tree.isAddr )
    codestr += "*" ;
codestr += tree.temp + " = ";
if( tree.rhs.isAddr )
    codestr += "*";
codestr += tree.right.temp;
emitCode( codestr );
```

Record References

```
typedef struct {
   int i;
   char c;
   int j;
} Record;
…
Record x;
```

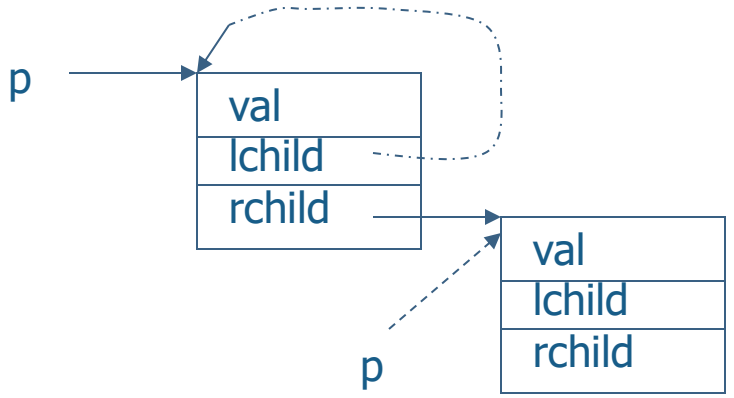


e.g., x.j = x.i

```
t1 = &x + field_offset(x, j)
t2 = &x + field_offset(x, i)
*t1 = *t2
```

Pointer References

```
typedef struct treeNode {
   int val;
   struct treeNode *lchild, *rchild;
} TreeNode;
...
TreeNode *p;
```




```
e.g.,  p->lchild = p;
       p = p->rchild;

t1 = p + field_offset(*p, lchild)
*t1 = p;
t2 = p + field_offset(*p, rchild)
p = *t2
```