Implementing Classes, Arrays, and Assignments

Logistics

- –TCE, Teacher Course Evaluation (see email "TCE Student Notification")
  - –They are anonymous and not seen until after final grades are out.
  - –Suggestions from TCEs are used to continually improve courses.
  - –If >80% of class does the TCE, then will look for score <=93 on final for curve.
- –PA4 peer reviews and HW9 grades are coming
- –PA5 is due December 5th
- –PA4 issues
  - –Revising 40 point penalty down to 25.
  - –Use regress.bash or some other script to perform regression testing.

Implementing Classes, Arrays, and Assignments

(1) (Covered Last Time) Memory model for classes and arrays

(2) Type Checking

(3) Code Generation

Some Terminology

Implicit “this” (callee)

- a reference to the object the method was called on
- Reference compiler assumes ”this” is passed as parameter in registers r25:r24

Receiver expression (caller)

- For method calls, the expression to the left of the “.”.
- Evaluates to a pointer to the object that the method is being called on.

Activation record (callee)

- Sequence of bytes on the run-time stack that contains the information for one method.
- Includes the return address for caller, old frame pointer, parameters (including the implicit “this”), locals, and whatever expressions are currently being computed.

Implementing type checking for PA5 MeggyJava

Visitor over AST will check for type errors at each AST node

Syntax *AST node*

```
id = Exp ;                           AssignStatement(id, Exp)
```

[LINENUM,POSNUM] Undeclared variable VARNAME
[LINENUM,POSNUM] Invalid expression type assigned to variable VARNAME

```
public Type name(...) {...return Exp; }
                                          MethodDecl(name, Stms, Exp)
```

[LINENUM,POSNUM] Invalid type returned from method METHODNAME

Exp . name (Args) CallExp(name, Args)

[LINENUM,POSNUM] Receiver of method call must be a class type

[LINENUM,POSNUM] Method METHODNAME does not exist

[LINENUM,POSNUM] Method METHODNAME requires exactly NUM arguments

[LINENUM,POSNUM] Invalid argument type for method METHODNAME

Error message for symbols redeclared within same scope

```
class ID …                         ClassDecl
public Type ID …                   MethodDecl
Type ID;                           VarDecl
```

[LINENUM,POSNUM] Redefined symbol ID

Code Gen for Classes and Local variables

Method activation records on run-time stack

- Parameters will still have locations in the activation record.
- Local variables will also have locations in the activation record.

Member variables will be stored in object instances

- The new expression should cause a call to malloc.
- Member variables will have offsets within an object instance.
- The "this" variable will contain a pointer to the object instance.

BuildSymTable for varDecl

```
VarDecl
    create VarSTE

    if it is a member variable
      make the base "Z" (r31:r30)
      make the offset the current class offset (ClassSTE will need this)
      increment the class offset/size with the size of the variable

    else if it is a local
      make the base "Y" (r29:r28)
      make the offset the current method offset (MethodSTE)
      increment the method offset/size with the size of the variable
```

Code Generation for IdExp and assignStmt

IdExp

1) Lookup id in symbol table to get VarSTE
2) If the VarSTE is a member variable
    2a) Look up VarSTE for "this" and generate code that loads the value of "this" into registers r31:r30.
3) load variable into a register(s) using the base+offset from VarSTE.
4) Push the variable value on the stack.

AssignStatement

1) Lookup id in symbol table to get VarSTE
2) If the VarSTE is a member variable
    2a) Look up VarSTE for "this" and generate code that loads the value of "this" into registers r31:r30.
3) store value of expression on top of run-time stack into base+offset from VarSTE

Code Generation for method call and this

CallExp

1) Use checkTypes to determine class type or receiver.
2) Look up the ClassSTE from the receiver type.
3) Then lookup the MethodSTE from the ClassSTE scope.
2) Generate code that pops parameters off the stack and
    into the appropriate registers from right to left.
    Receiver reference is the first parameter (this), last pushed.
3) Generate code that calls the mangled method name.
4) Generate code that pushes the return value back on the stack.

ThisExp

1) push the value of the "this" parameter onto the run-time stack
   load “this” into r31:30   and then push it

Arrays

An array is a collection of items of the same type

- so that the address of an element can be computed from the start address and the index (efficiency)
- index: int (or int derivative type like unsigned or byte)

Once an array is allocated, the sizes of its dimensions do not change (as opposed to ArrayLists, Lists, …)

Java arrays are one Dimensional

- higher dimensional arrays are arrays of arrays

  these are sometimes called "ragged" arrays, as the lengths sub arrays can differ
- as opposed to rectangular arrays in Fortran and C

Implementing type checking for MeggyJava (Arrays)

Syntax	*AST Node(s)*
new int [Exp] new Meggy.Color [Exp]	NewArrayExp
Exp [Exp]	ArrayExp and ArrayAssignStatement
Exp . length	LengthExp

new int [Exp] — NewArrayExp
new Meggy.Color [Exp]

[LINENUM,POSNUM] Invalid operand type for new array operator
// number of elements should be an integer or byte

Exp [Exp] — ArrayExp and ArrayAssignStatement

[LINENUM,POSNUM] Array reference to non-array type
[LINENUM,POSNUM] Invalid index expression type for array reference
// index expression should be of type integer or byte
[LINENUM,POSNUM] Invalid expression type assigned into array
// array could be an array of colors or an array of integers

Exp . length — LengthExp

[LINENUM,POSNUM] Operator length called on non-array type
// type of the length expression is integer

Dynamically Allocating Arrays

NewArrayExp

- Assume size of array in elements is on stack as an int
- Gen code to calculate number of bytes, numelem * sizeof(elem)
- Gen code to add 2 bytes for lenth int to size of array
- Gen code to call malloc
- Gen code to set the first two bytes of array to numelem
- Gen code to push array’s address onto stack

Length Expression

LengthExp

- Assume array reference/pointer is already on top of stack at runtime.
- Gen code that pops array reference off stack into two registers.
- Gen code that loads integer that array reference points to into two registers.
- Gen code to push that value/length onto the stack.

Uses of Array Elements

ArrayExp

- Assume the integer index is at the top of the stack and the array reference/pointer is directly under it.
- Generate code to pop those off the stack and into registers.
- Generate code to calculate the array element address.

- Generate code that loads the array element and pushes it onto stack.

Generate Code for Array Assignments

ArrayAssignStatement

- Assume that rhs expression, index expression, and array reference are on the stack.
- Generate code to pop those of the stack and store them into registers.
- Generate code that calculates the array element address (see previous slide).
- Generate code that stores the rhs expression into the array element memory location.

Another Memory Layout Example

```
import meggy.Meggy;

class ArrayExp {
    public static void main(String[] a){
        Meggy.setPixel((byte)(new MyClass().testing()),
                       (byte)4, Meggy.Color.GREEN);
    }
}

class MyClass {
    public int testing() {
        int [] x;
        x = new int [7];
        x[0] = 1;
        x[1] = 6;
        x[6] = 3;
        return x[0] + x[x[1]];
    }
}
```