Homework: RefLang

Instructions:

- Due Date: Nov 9th 2016 (Wed) 6:00 pm
- Export your homework solution as archive file (File->Export->General->archive).
- Attach the archive file. Please provide the complete project as submission not just the src directory.
- In this homework, we will implement the interpreter for Reflang. **Here are all the changes that are required**.
  - Extend the set of values in Value.java to add RefVal which stores the location.(Question 1)
  - Implement memory in form of an array. You need to create a file Heap.java (Question 2)
  - Implement ASTs for required expressions in AST.java.(Question 3 a)
  - Extend Formatter for these expressions.(Question 3 b)
  - Implement semantics of these expressions in Evaluator.java.(Question 4)
  - Extend the grammar for these newly added expressions.(Question 4)
  - Test your Reflang for expressions developed in Reflang.(Question 5)

Questions:

1. (5 pt) First, we will add a new kind of value to the set of values. This new kind of value will represent reference values in our language. You can do that by:
   - Adding a new Java class, RefVal, to the interface Value.
   - Internally this class will maintain an integer index, loc, and
   - RefVal class must provide methods to access this loc.
   - The string representation of a RefVal (in method tostring) will be created by prepending "loc:" to the value of loc.

   Sol

```
static class RefVal implements Value {
  private int _loc = -1;
  public RefVal(int loc) { _loc = loc; }
  public String tostring() {
    return "loc:" + this._loc;
  }
  public int loc() { return _loc; }
}
```

2. (15 pt) Design and implement Heap, a new abstraction representing area in the memory reserved for dynamic memory allocation.

(a) Implement Heap as a Java interface named Heap with four methods ref, deref, setref, and free.

- The return type of all four methods is Value.
- The method ref takes a single parameter of type Value.
- The method deref takes a single parameter of type Value.RefVal from Question 1.
- The method setref takes a two parameters of type Value. First parameter is of type RefVal while second parameter is Value.
- The method free takes a single parameter of type Value.RefVal from Question 1.

(b) Implement a 16 bit heap as a Java class Heap16Bit inside the interface Heap.

- The class Heap16Bit must implement the interface Heap, and thus provide implementation of each method ref, deref, setref, and free inside the interface.
- The class would model memory as an array named _rep of type Value[]
- The method ref
  - takes a single parameter val, of type Value
  - allocates memory and stores val in allocated memory at location l
  - returns a RefVal containing location l.
- The method deref
  - takes a single parameter loc, of type RefVal
  - returns value stored at location l, where l is stored in loc.
- The method setref
  - takes two parameters
  - first parameter loc, is of type RefVal which encapsulates location l
  - second parameter val, is of type Value
  - this method replaces the value stored at l with val
  - returns val
- The method free
  - takes a single parameter loc, of type RefVal which encapsulates location l
  - deallocates the memory location l from _rep.
  - returns loc

Sol

```
public interface Heap {

  Value ref (Value value);

  Value deref (Value.RefVal loc);

  Value setref (Value.RefVal loc, Value value);

  Value free (Value.RefVal value);

  static public class Heap16Bit implements Heap {
    static final int HEAP_SIZE = 65_536;
    Value[] _rep = new Value[HEAP_SIZE];
    int index = 0;

    public Value ref (Value value) {
      if(index >= HEAP_SIZE)
      return new Value.DynamicError("Out␣of␣memory␣error");
      Value.RefVal new_loc = new Value.RefVal(index);
      _rep[index++] = value;
      return new_loc;
    }

    public Value deref (Value.RefVal loc) {
      try {
        return _rep[loc.loc()];
      } catch (ArrayIndexOutOfBoundsException e) {
        return new Value.DynamicError("Segmentation␣fault␣at␣access␣" + loc);
      }
    }

    public Value setref (Value.RefVal loc, Value value) {
      try {
        return _rep[loc.loc()] = value;
      } catch (ArrayIndexOutOfBoundsException e) {
        return new Value.DynamicError("Segmentation␣fault␣at␣access␣" + loc);
      }
    }

    public Value free (Value.RefVal loc) {
      try {
        _rep[loc.loc()] = null;
        return loc;
      } catch (ArrayIndexOutOfBoundsException e) {
        return new Value.DynamicError("Segmentation␣fault␣at␣access␣" + loc);
      }
    }

    public Heap16Bit(){}
  }

}
```

3. (20 pt) Question 1 and Question 2 helped you creating the RefVal and Heap representation. Now we would be creating the AST node for the expressions.

   (a) Extend the AST.java and add the representation of the following nodes.
   - refexp
   - derefexp
   - assignexp
   - freeexp

   (b) Extend the Formatter for these new AST nodes in a manner consistent with existing AST nodes.

Sol

(a) AST.java

```
    public static class RefExp extends Exp {
      private Exp _value_exp;

      public RefExp(Exp value_exp) {
        _value_exp = value_exp;
      }

      public Object accept(Visitor visitor, Env env) {
        return visitor.visit(this, env);
      }

      public Exp value_exp() { return _value_exp; }

    }

    public static class DerefExp extends Exp {
      private Exp _loc_exp;

      public DerefExp(Exp loc_exp) {
        _loc_exp = loc_exp;
      }

      public Object accept(Visitor visitor, Env env) {
        return visitor.visit(this, env);
      }

      public Exp loc_exp() { return _loc_exp; }

    }

    public static class AssignExp extends Exp {
      private Exp _lhs_exp;
      private Exp _rhs_exp;

      public AssignExp(Exp lhs_exp, Exp rhs_exp) {
        _lhs_exp = lhs_exp;
        _rhs_exp = rhs_exp;
      }

      public Object accept(Visitor visitor, Env env) {
        return visitor.visit(this, env);
      }

      public Exp lhs_exp() { return _lhs_exp; }
      public Exp rhs_exp() { return _rhs_exp; }

    }

    public static class FreeExp extends Exp {
      private Exp _value_exp;

      public FreeExp(Exp value_exp) {
        _value_exp = value_exp;
      }

      public Object accept(Visitor visitor, Env env) {
        return visitor.visit(this, env);
      }

      public Exp value_exp() { return _value_exp; }

    }
```

(b) Printer.java

```
    public String visit(AST.RefExp e, Env env) {
        String result = "(ref␣";
        result += e.value_exp().accept(this, env);
        return result + ")";
    }

    public String visit(AST.DerefExp e, Env env) {
        String result = "(deref␣";
        result += e.loc_exp().accept(this, env);
        return result + ")";
    }

    public String visit(AST.AssignExp e, Env env) {
        String result = "(set!␣";
        result += e.lhs_exp().accept(this, env) + "␣";
        result += e.rhs_exp().accept(this, env);
        return result + ")";
    }

    public String visit(AST.FreeExp e, Env env) {
        String result = "(free␣";
        result += e.value_exp().accept(this, env);
        return result + ")";
    }
```

4. (60 pt) Goal of this question is to understand and implement the semantics of the expressions created for Reflang.

   (a) (10 pt) In Question 2, we have designed an Interface called Heap, which supports 4 different methods. Add a global heap of type Heap16Bit to the interpreter. This object will be the heap used by all expressions in the evaluator.

   (b) (10 pt) Implement visit method for refexp in Evaluator.java according to the semantics of ref expression specified in prologue.

(c) (10 pt) Implement visit method for derefexp in Evaluator.java according to the semantics of ref expression specified in prologue.

(d) (10 pt) Implement visit method for assignexp in Evaluator.java according to the semantics of ref expression specified in prologue.

(e) (10 pt) Implement visit method for freeexp in Evaluator.java according to the semantics of ref expression specified in prologue.

(f) (10 points) In order to get your Reflang working, you would be required to extend the grammar file. Extend the grammar file for supporting four new expressions of Reflang.

Sol

(a)
```
public class Evaluator implements Visitor<Value> {
  Heap heap = new Heap16Bit();
}
```

(b)
```
public Value visit(RefExp e, Env env) { // New for reflang.
  Exp value_exp = e.value_exp();
  Value value = (Value) value_exp.accept(this, env);
  return heap.ref(value);
}
```

(c)
```
public Value visit(DerefExp e, Env env) { // New for reflang.
  Exp loc_exp = e.loc_exp();
  Value.RefVal loc = (Value.RefVal) loc_exp.accept(this, env);
  return heap.deref(loc);
}
```

(d)
```
public Value visit(AssignExp e, Env env) { // New for reflang.
  Exp rhs = e.rhs_exp();
  Exp lhs = e.lhs_exp();
  //Note the order of evaluation below.
  Value rhs_val = (Value) rhs.accept(this, env);
  Value.RefVal loc = (Value.RefVal) lhs.accept(this, env);
  Value assign_val = heap.setref(loc, rhs_val);
  return assign_val;
}
```

(e)
```
public Value visit(FreeExp e, Env env) { // New for reflang.
  Exp value_exp = e.value_exp();
  Value.RefVal loc = (Value.RefVal) value_exp.accept(this, env);
  heap.free(loc);
  return new Value.UnitVal();
}
```

(f) grammar file

```
...
| ref=refexp { $ast = $ref.ast; }
| deref=derefexp { $ast = $deref.ast; }
| assign=assignexp { $ast = $assign.ast; }
| free=freeexp { $ast = $free.ast; }
;

refexp returns [RefExp ast] :
'(' Ref
e=exp
')' { $ast = new RefExp($e.ast); }
;

derefexp returns [DerefExp ast] :
'(' Deref
e=exp
')' { $ast = new DerefExp($e.ast); }
;

assignexp returns [AssignExp ast] :
'(' Assign
e1=exp
e2=exp
')' { $ast = new AssignExp($e1.ast, $e2.ast); }
;

freeexp returns [FreeExp ast] :
'(' Free
e=exp
')' { $ast = new FreeExp($e.ast); }
;
```

5. (15 pt) Goal of this question is to test our implementation and understand the semantics of the Reflang interpreter.

   (a) (2 pt) Perform the following operations on your implementation of Reflang and provide transcript in HW7.scm file.
   (deref (ref 1))
   (free (ref 1))
   (let ((loc (ref 1))) (set! loc 2))
   (let ((loc (ref 3))) (set! loc (deref loc)))

   (b) (3 pt) Write 3 Reflang programs which use aliases and also provide the transcript of running those programs.

   (c) In this question you will be implementing a linked list. In a linked list one element of the node is reference to another node. In our implementation of linked list, each node will have two fields. First field of the node is a number while second element will be reference to other node.

   - (10 pt) write a lambda method 'add', which

i. takes two parameters
ii. first parameter 'head' is head of linked list
iii. second parameter 'ele' is a node
iv. add function adds ele at the end of linked list

- (5 pt) write a lambda method 'print', which
  i. takes node as parameter (representing head of linked list)
  ii. returns a list of numbers present in linked list.

Following transcripts will help you understand the functions more:

```
$ (define head (node 1))
$ (add head (node 2))
(lambda ( op ) (if op fst snd))
$ (add head (node 3))
(lambda ( op ) (if op fst snd))
$ (print head)
(1 2 3)
$ (add head (node 0))
(lambda ( op ) (if op fst snd))
$ (add head (node 6))
(lambda ( op ) (if op fst snd))
$ (print head)
(1 2 3 0 6)
```

Sol

1.
```
$ (deref (ref 1))
1
$ (free (ref 1))
$ (let ((loc (ref 1))) (set! loc 2))
2
$ $(let ((loc (ref 3))) (set! loc (deref loc)))
3
```

2. (a)
```
$ (define a (ref 0))
loc:0
$ (let ( (x a)) x)
loc:0
```

(b)
```
$ (define alias (free a))
```

(c)
```
$ (let ((loc3 (ref 4))) (let ((loc4 loc3)) (deref loc4)))
4
```

3. (a)

```
(define pairNode (lambda (fst snd) (lambda (op) (if op fst snd))))
(define node (lambda (x) (pairNode x (ref (list)))))
(define getFst (lambda (p) (p #t)))
(define getSnd (lambda (p) (p #f)))

(define add
(lambda (head ele)
(if (null? (deref (getSnd head)))
(set! (getSnd head) ele)
(add (deref (getSnd head)) ele))))
```

(b)

```
(define printlinkedlist
(lambda (head)
(if (null? (deref (getSnd head)))
(deref (getFst head))
(cons (deref (getFst head))
(printlinkedlist (deref (getSnd head)))))))
```

Extra Credit:

1. (25 points) Goal of this question is to understand the contiguous memory allocation and indexing.

   Extend the Funclang language to add array of numeric values as a new kind of value to the programming language. This would require adding three new kinds of expressions arrayexp, indexexp, and assignarrayexp. You will also need to generalize the array allocation and access operations for heap from the previous question.

   To create an array with three rows, one can use the arrayexp as follows.

```
$ (array 3)
[0
0
0]
```

   In the output we have adjusted spacing for clarity, but you are simply required to produce output that is equal to these. To create an array with three rows and four columns, one can use the arrayexp as follows.

```
$ (array 3 4)
[[0000]
[0000]
[0000]]
```

   To create a three-dimensional array with three rows, four columns, and height two, one can use the arrayexp as follows.

```
$ (array 3 4 2)
[[[0000]
[0000]
[0000]]
[[0000]
[0000]
[0000]]]
```

To access the second element in an array with three rows, one can use the indexexp as follows.

```
$ (index (array 3) 2)
0
```

To access the element in second row and first column in an array with three rows and four columns, one can use the indexexp as follows.

```
$ (index (array 3 4) 2 1)
0
```

To assign the second element in an array with three rows, one can use the assignarrayexp as follows.

```
$ (assign (array 3) 2 342)
[0
342
0]
```

To assign the element in second row and first column in an array with three rows and four columns, one can use the indexexp as follows.

```
$ (assign (array 3 4) 2 1 342)
[[0000]
[342000]
[0000]]
```

Sol
grammar file:

```
...
| arr=arrayexp { $ast = $arr.ast; }
| ind=indexexp { $ast = $ind.ast; }
| arrassign=arrayassignexp { $ast = $arrassign.ast; }
...
```

```
arrayexp returns [ArrayExp ast]
locals [ArrayList<Exp> dimensions = new ArrayList<Exp>(); ]:
'(' 'array'
( e=exp { $dimensions.add($e.ast); } ) *
')' { $ast = new ArrayExp($dimensions); }
;

indexexp returns [IndexExp ast]
locals [ArrayList<Exp> indices = new ArrayList<Exp>(); ]:
'(' 'index'
arr=exp
( e=exp { $indices.add($e.ast); } ) +
')' { $ast = new IndexExp($arr.ast, $indices); }
;

arrayassignexp returns [ArrAssignExp ast]
locals [ArrayList<Exp> indices = new ArrayList<Exp>(); ]:
'(' 'assign'
arr=exp
( e=exp { $indices.add($e.ast); } ) +
val=exp
')' { $ast = new ArrAssignExp($arr.ast, $indices, $val.ast); }
;
```

Value.java:

```
static class ArrayVal implements Value {

  private List<Integer> _dimensions;
  private List<RefVal> _vals;
  private int toStringIndex;
  private Heap _heap; //I need the heap itself for printing out the actual array.
  public ArrayVal(List<Integer> dimensions, List<RefVal> vals, Heap heap) {
    _dimensions = dimensions;
    _vals = vals;
    _heap = heap;
  }

  public List<Integer> getDimentions(){
    return _dimensions;
  }

  public List<RefVal> getVals(){
    return _vals;
  }

  @Override
  public String tostring() {
    StringBuilder result = new StringBuilder();
    if(_dimensions.size() == 0){
      return "[]";
    }
    if(_dimensions.size() == 1){
      result.append("[");
        for(int i = 0; i < _vals.size() - 1; i++){
          RefVal val = _vals.get(i);
          result.append(_heap.deref(val).tostring());
          result.append("\n");
        }
        result.append(_heap.deref(_vals.get(_vals.size() - 1)).tostring());
        result.append("]");
      return result.toString();
    }
    toStringIndex = 0;
    result = stringHelper(_dimensions.size() - 1, result);
    return result.toString();
  }

  private StringBuilder stringHelper(int dimIndex, StringBuilder curString){
    //Base case, construct the 2D n x m matrix.
    if(dimIndex == 1){
      curString.append("[");
        for(int i = 0; i < _dimensions.get(0); i++){
          curString.append("[");
            for(int j = 0; j < _dimensions.get(1); j++){
              RefVal val = _vals.get(toStringIndex);
              curString.append(_heap.deref(val).tostring());
              curString.append(" ");
              toStringIndex++;
            }
            curString.delete(curString.length() - 1, curString.length());
            curString.append("]\n");
        }
        curString.delete(curString.length() - 2, curString.length());
        curString.append("]");
      curString.append("]");
    return curString;
  }
  //Bigger cases
  curString.append("[");
    for(int i = 0; i < _dimensions.get(dimIndex); i++){
      stringHelper(dimIndex - 1, curString);
      curString.append("\n");
    }
    //Remove the last new line
    curString.delete(curString.length() - 1, curString.length());
    curString.append("]");
  return curString;
}

}
```

Evaluator.java

```
@Override
public Value visit(ArrayExp e, Env env) {
    List<Exp> dimention = e.dimensions();

    List<Integer> actualDimention = new ArrayList<>();
    int totalNumVals = 1;
    if(dimention.size() == 0){
      totalNumVals = 0;
    }
    for(Exp exp: dimention){
      Value val = (Value)exp.accept(this, env);
      // if any of the dimention is dynamicErrorthen return the error
      if(val instanceof DynamicError){
        return val;
      }
      // check if the dimentions are integer
      if(!(val instanceof NumVal) || ((NumVal)val).v() != Math.floor(((NumVal)val).v())){
        return new DynamicError("Error: Array sizes should be integers.");
      }
      // get the dimentions as integer
      int dimension = (int) ((NumVal)val).v();
      if(dimension <= 0){
        return new DynamicError("Error: Array sizes should be positive integers.");
      }
      actualDimention.add(dimension);
      //update the number of elements in the array
      totalNumVals *= dimension;
    }

    List<RefVal> refs = new ArrayList<>();
    for(int i = 0; i < totalNumVals; i++){
      Value res = heap.ref(new NumVal(0));
      // if heap runs out of memory
      if(res instanceof DynamicError){
        return res;
      }
      refs.add((RefVal)res);
    }
    // return all locations of array
    return new ArrayVal(actualDimention, refs ,heap);
  }

  @Override
  public Value visit(IndexExp e, Env env) {
    List<Exp> indices = e.indices();
    Exp array = e.array();
    Value arr = (Value)array.accept(this, env);

    if(! (arr instanceof ArrayVal)){
      return new DynamicError("Error: First argument must be an array.");
    }
    if(((ArrayVal)arr).getDimentions().size() == 0){
      return new DynamicError("Error: An empty array has no elements to index.");
    }
    List<Integer> requestedIndex = new ArrayList<>();
    int i = 0;
    for(Exp exp: indices){
      Value val = (Value)exp.accept(this, env);
      if(val instanceof DynamicError){
        return val;
      }
      if(!(val instanceof NumVal) || ((NumVal)val).v() != Math.floor(((NumVal)val).v()) || ((int) ((NumVal)val
          ).v() <= 0)){
        return new DynamicError("Error:␣Array␣indices␣should␣be␣positive␣integers.");
      }
      int index = (int) ((NumVal)val).v();
      if(index > ((ArrayVal)arr).getDimentions().get(i)){
        return new DynamicError("Error:␣Indices␣should␣not␣exceed␣their␣relevant␣array␣size.");
      }
      requestedIndex.add(index);
      i++;
    }
    i = 0;
    //Now, using indicesEval, compute the index we want to access!
    if(requestedIndex.size() != ((ArrayVal)arr).getDimentions().size()){
      return new DynamicError("Error:␣Too␣many␣or␣too␣few␣indices␣for␣this␣array.");
    }
    int index = getIndex(requestedIndex, ((ArrayVal)arr).getDimentions());
    return heap.deref(((ArrayVal)arr).getVals().get(index));
  }

  @Override
  public Value visit(ArrAssignExp e, Env env) {
    List<Exp> indices = e.indices();
    Exp array = e.getArr();
    Value arr = (Value)array.accept(this, env);
    if(arr instanceof DynamicError){
      return arr;
    }
    if(! (arr instanceof ArrayVal)){
      return new DynamicError("Error:␣First␣argument␣must␣be␣an␣array.");
    }
    if(((ArrayVal)arr).getDimentions().size() == 0){
      return new DynamicError("Error:␣An␣empty␣array␣has␣no␣elements␣to␣index.");
    }
    List<Integer> indicesEval = new ArrayList<>();
    int i = 0;
    for(Exp exp: indices){
      Value val = (Value)exp.accept(this, env);
      if(val instanceof DynamicError){
        return val;
      }
      if(!(val instanceof NumVal) || ((NumVal)val).v() != Math.floor(((NumVal)val).v())){
        return new DynamicError("Error:␣Array␣indices␣should␣be␣integers.");
      }
      int index = ((Double)((NumVal)val).v()).intValue();
      if(index <= 0){
        return new DynamicError("Error: Indices must be positive.");
      }
      if(index > ((ArrayVal)arr).getDimentions().get(i)){
        return new DynamicError("Error: Indices should not exceed their relevant array size.");
      }
      indicesEval.add(index);
      i++;
    }
    //Now, using indicesEval, compute the index we want to access!
    if(indicesEval.size() != ((ArrayVal)arr).getDimentions().size()){
      return new DynamicError("Error: Too many or too few indices for this array.");
    }
    List<Integer> dims = ((ArrayVal)arr).getDimentions();
    int index = getIndex(indicesEval, dims);

    //Check the new value
    Value toChange = (Value)e.val().accept(this, env);
    if(toChange instanceof DynamicError){
      return toChange;
    }
    if(!(toChange instanceof NumVal)){
      return new DynamicError("Error: Current functionality allows only numeric arrays.");
    }
    //Update the reference and return the array.
    heap.setref(((ArrayVal)arr).getVals().get(index),toChange);
    return arr;
  }

  //private helper to transform a list of indices into the index of the 1D RefVal array.
  private int getIndex(List<Integer> indicesEval, List<Integer> dims){
    //First, for a one dimensional array, the index is just the index we found (-1 because 0 indexing).
    if(indicesEval.size() == 1){
      return indicesEval.get(0) - 1;
    }
    //Due to the way this homework is set up, we always have to add the first coordinate and the colNum*
        secondCoordinate.
    int index = (indicesEval.get(1) - 1) + (indicesEval.get(0) - 1) * (dims.get(1));
    //The remainder of the coordinates are multiplied by a factor based on the arr dimensions.
    for(int i = 2; i < indicesEval.size(); i++){
      int multFactor = 1;
      for(int j = 0; j < i; j++){
        multFactor *= dims.get(j);
      }
      index += (indicesEval.get(i) - 1) * multFactor;
    }
    return index;
  }
```

AST.java

```
  public static class ArrayExp extends Exp {
    private List<Exp> _dimensions;
    public ArrayExp(List<Exp> dimensions){
      _dimensions = dimensions;
    }
    public List<Exp> dimensions(){
      return _dimensions;
    }
    @Override
    public Object accept(Visitor visitor, Env env) {
      return visitor.visit(this, env);
    }
  }

  public static class IndexExp extends Exp {
    private Exp _arrExp;
    private List<Exp> _indices;
    public IndexExp(Exp arrExp, List<Exp> indices){
      _arrExp = arrExp;
      _indices = indices;
    }
    public Exp array(){
      return _arrExp;
    }
    public List<Exp> indices(){
      return _indices;
    }
    @Override
    public Object accept(Visitor visitor, Env env) {
      return visitor.visit(this, env);
    }
  }

  public static class ArrAssignExp extends Exp {
    private Exp _arrExp;
    private List<Exp> _indices;
    private Exp _val;
    public ArrAssignExp(Exp arrExp, List<Exp> indices, Exp val){
      _arrExp = arrExp;
      _indices = indices;
      _val = val;
    }
    public Exp getArr(){
      return _arrExp;
    }
    public List<Exp> indices(){
      return _indices;
    }
    public Exp val(){
      return _val;
    }
    @Override
    public Object accept(Visitor visitor, Env env) {
      return visitor.visit(this, env);
  }
}
```

Printer.java

```
@Override
public String visit(ArrayExp e, Env env) {
  String result = "(array␣";
  List<Exp> dim = e.dimensions();
  for(Exp exp: dim){
    result += exp.accept(this, env) + "␣";
  }
  return result + ")";
}

@Override
public String visit(IndexExp e, Env env) {
  String result = "(index␣";
  result += e.array().accept(this, env) + "␣";
  List<Exp> ind = e.indices();
  for(Exp exp: ind){
    result += exp.accept(this, env) + "␣";
  }
  return result + ")";
}

@Override
public String visit(ArrAssignExp e, Env env) {
  String result = "(assign␣";
  result += e.getArr().accept(this, env) + "␣";
  List<Exp> ind = e.indices();
  for(Exp exp: ind){
    result += exp.accept(this, env) + "␣";
  }
  result += e.val().accept(this, env);
  return result + ")";
}
}
```