Compiler Practical 2013
Storage Administration: Implementation

Berthold Hoffmann (B. Gersdorf, T. Röfer)
hof@informatik.uni-bremen.de
Cartesium 2.48
Structure

1. Add Garbage Collection
2. Administration of the Root Set
3. Simple Storage Adjustment
4. Changing the Semantics of Assignments
5. Bonus Tasks
Adding garbage collection

• Changing the administration of the heap
  – Store heap pointer in storage
  – This frees register R4
  – *NEW* must set attributes to *NULL*
  – Local variables must be initialized to *NULL*

• Administration of the root set
  – One stack contains all references to objects, a second stack contains the others
  – R4 points to one stack, R2 points to the other one
Adding garbage collection(2)

- Handle lack of storage
  - *NEW* compares with upper bound of Heap
  - If heap is full, start garbage collection
  - If heap is still full, an error occurs

- Change semantics of assignments
  - Evaluate right-hand side before left-hand side
    (Why?)

*Hint*: left-hand side can reference to parts of objects on the heap – this is bad for most garbage collection algorithms
METHOD m(a, b : Integer) : Integer IS
    c, d : Integer;
BEGIN
    RETURN 42;
END METHOD

Call frame so far, without separated references:

<table>
<thead>
<tr>
<th>Address</th>
<th>Caller frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3-4</td>
<td>SELF</td>
</tr>
<tr>
<td>R3-3</td>
<td>a</td>
</tr>
<tr>
<td>R3-2</td>
<td>b</td>
</tr>
<tr>
<td>R3-1</td>
<td>Return address</td>
</tr>
<tr>
<td>R3</td>
<td>Address of call frame</td>
</tr>
<tr>
<td>R3+1</td>
<td>c</td>
</tr>
<tr>
<td>R2+2</td>
<td>d</td>
</tr>
<tr>
<td>R3+3</td>
<td>Return value</td>
</tr>
</tbody>
</table>
... and with separation of references to the heap:

<table>
<thead>
<tr>
<th>Address</th>
<th>Caller frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>Return address</td>
</tr>
<tr>
<td>:</td>
<td>Address of call frame</td>
</tr>
<tr>
<td>:</td>
<td>Intermediate values (L-values, values)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Caller frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3 – 3</td>
<td>SELF</td>
</tr>
<tr>
<td>R3-2</td>
<td>a</td>
</tr>
<tr>
<td>R3-1</td>
<td>b</td>
</tr>
<tr>
<td>R3</td>
<td>c</td>
</tr>
<tr>
<td>R3+1</td>
<td>d</td>
</tr>
<tr>
<td>R3+2</td>
<td>Return value</td>
</tr>
</tbody>
</table>
Simple Garbage Collection (1)

Approach:

• *Copy Collector*, using stack space

• Every object copies itself, and all objects it points to

• Most of the implementation is done in generated LOOP methods
Implementation:

• Every class can generate a new object of its type, and can clone all its attributes

• *Object* contains a method that can clone the object, using the methods defined in derived classes

• A type-less („_Null“) attribute points to the copy of the object

• Every Object is copied only once
28.06.2012

Simple Copying Collector (1)

FOR EACH b root setDO
  b := b.lookupNewAddr;
END FOR

METHOD lookupNewAddr IS BEGIN
  IF newAddr = NULL THEN
    newAddr := NEW ThisClass;
    FOR EACH Reference r of SELF DO
      IF r # NULL THEN
        newAddr.r := r.lookupNewAddr;
      END IF
    END FOR
  END IF
  RETURN newAddr;
END METHOD

For every class
Simple Copying Collector (2)

• Assumptions
  – The heap pointer is set to the new heap at the beginning so that NEW reserves space on the new heap
  – NEW initializes the attribute newAddr in the new object with NULL

• Hints
  – Do not forget attributes of base classes
  – Integer and Boolean contain an attribute that is not a reference
Changing the Semantics of “:=“

- **Problem**: During garbage collection, addresses of attributes can lie on the R2-stack.
- **Solution**: evaluate right-hand side before left-hand side.

CLASS Main IS
   a : Main;
   METHOD b IS
      a := NEW Main;
   END METHOD
END CLASS

<table>
<thead>
<tr>
<th>Address</th>
<th>Caller frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2 – 2</td>
<td>Return address</td>
</tr>
<tr>
<td>R2 – 1</td>
<td>Address of caller frame</td>
</tr>
<tr>
<td>R2</td>
<td>Address of Self.a</td>
</tr>
</tbody>
</table>

R4-Stapel

<table>
<thead>
<tr>
<th>Address</th>
<th>Caller frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3 - 1</td>
<td>SELF</td>
</tr>
</tbody>
</table>
Garbage Collection without using stack space:

• True Copy Collector
  – No stack space consumption
    (depending on the structure of the heap)

• Mark and Compact Collector
  – Only one heap
Dispense of Variable Initialization (Idea):

• Space on the stack for local variables is reserved only when they are updated.
• An initialization corresponds to a push onto the $R4$-stack.
• The relative address of local variables is determined by the order of their uses, not by the order of their declarations.
• It is forbidden to use a local variable before it has been initialized.
Dispense of Variable Initialization (Implementation):

• *VarDeclaration* contains a label indicating whether the variable has been initialized

• Setting *VarDeclaration.offset* is delayed

5%
Much Success ... (du courage!) ... and nice vacations