LW5

DA53: Compilers and Language Theory

stephane.galland@utbm.fr

Virtual Machine

00 | Goal

The goal of this on-computer tutorial session is to write a small virtual machine.

The virtual machine reads a kind of byte-code (each instruction is represented by 4 values of type long).

The format of the byte-code, and its parser, is provided in the skeleton.

The development must be done in Java.

The execution is in two phases : compilation for obtaining the byte code, and execution of the byte in the virtual machine.

The first phase takes as parameter the TinyBasic program and outputs the binary file that contains the byte code.

The second phase takes as parameter the binary file that contains the byte-code and run the instructions from it.

Compilation :

```
$> java "fr.utbm.gi.lo46.tp5.TinyBasicCompiler" "test6.tb" "test6.bc"
```

Running the Virtual Machine :

```
$> java "fr.utbm.gi.lo46.tp5.TinyBasicVM" "test6.bc"
```

01 | Working Steps

01.1 Stack

- The stack is a central structure in a virtual machine.
- The stack contains call contexts. The uppest call context is used to run the program prior to the lower ones.
- Each call context contains :
    - A mapping table from local-variable addresses (the keys) to the values.
    - The list of the formal parameters that are passed when the call context is created (usually, they are the values passed as parameter of a function)
    - The address of the variable that contains the value, which should be returned when the call context is popped from the stack.
    - The ordinal counter : the address of the next instruction to run.
    - A pointer to the parent context.
- Because the TinyBasic contains only global variables, only the lowest call context in the stask may contain a mapping table of local variables that is not empty.
- Note that all the values are Number, because they are numerical constants or addresses. The string literals are represented by the address of the string in the memory.
- **Write the Java class named StackContext.**
- Add the following functions :
    - **void setValueAt(int address, Number value)** where address is the address of the variable in the stack and value is the value of the variable. This function puts the value of a variable in the stack. The address is positive or null and is the index of the variable in the mapping table.
    - **Number getValueAt(int address)** replies the value at the specified address.
    - The getters and the setters for the other attributes of StackContext.

01.2 Heap Manager

- The HeapManager is in charge of managing all the values that are dynamically created in the memory, named the heap, the part of the memory dedicated to the dynamic allocations.
- The heap is divided into regions, each of them have an address. The regions cannot overlap in the heap.
- Each region may contains one or more memory units.
- A memory unit always contains one of :
    - a long integer value, or
    - a floating-point value, or

- an address, or
- a string.

- **Write the class MemoryUnit with the functions :**
  - Object getValue()
  - void setValue(Object value)
  - int size() : replies the number of bytes of the stored data.
  - Class<?> type() : replies the type of the data inside the memory unit.
- **Write the class HeapRegion with the functions :**
  - Object getValue() : replies the first value in the region
  - Object getValueAt(int index) : replies the value at the specified index in the region
  - void setValueAt(int index, Object value)
  - Class<?> typeOf(int index) : replies the type of the memory unit at the specified index.
  - int size() : replies the number of memory units in the region.
  - long address() : replies the address of the region, ie. the address of the first value in the region.
- **Write the class HeapManager with the functions :**
  - HeapRegion getRegion(long address) : replies the region with the specified address.
  - HeapRegion newRegion() : create a new region in the heap.
  - void reset() : clear the heap.

01.3 Virtual Machine

- Update the class TinyVirtualMachine according to the previous exercices.