Script generated by TTT

Title: Seidl: Virtual Machines (16.06.2014)

Date: Mon Jun 16 10:15:17 CEST 2014

Duration: 27:57 min

Pages: 10

Classes and Objects

Example:

```
int count = 0;
class list {
        int info;
        class list * next;
        list (int x) {
                info = x;  count++;  next = null;
        }
        virtual int last () {
                if (next == null) return info;
                else return next → last ();
        }
}
```


Discussion:

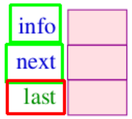
- We adopt the C++ perspective on classes and objects.
- We extend our implementation of C. In particular ...
- Classes are considered as extensions of structs. They may comprise:
  - ⇒ attributes, i.e., data fields;
  - ⇒ constructors;
  - ⇒ member functions which either are virtual, i.e., are called depending on the run-time type or non-virtual, i.e., called according to the static type of an object :-)
  - ⇒ static member functions which are like ordinary functions :-))
- We ignore visibility restrictions such as **public**, **protected** or **private** but simply assume general visibility.
- We ignore multiple inheritance :-)

40 Object Layout

Idea:

- Only attributes and virtual member functions are stored inside the class !!
- The addresses of non-virtual or static member functions as well as of constructors can be resolved at compile-time :-)
- The fields of a sub-class are appended to the corresponding fields of the super-class ...

... in our Example:



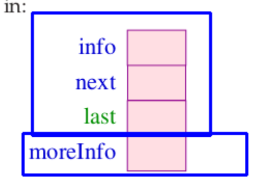
Idea (cont.):

- The fields of a sub-class are appended to the corresponding fields of the super-class :-)

Example:

```
class mylist : list {
    int moreInfo;
}
```

... results in:



For every class C we assume that we are given an adress environment ρ_C . ρ_C maps every identifier x visible inside C to its decorated relative address a . We distingish:

global variable	(G, a)
local variable	(L, a)
attribute	(A, a)
virtual function	(V, b)
non-virtual function	(N, a)
static function	(S, a)

For virtual functions x , we do not store the starting address of the code — but the relative address b of the field of x inside the object :-)

For the various of variables, we obtain for the L-values:

$$\text{code}_L\; x\; \rho \;=\; \begin{cases} \text{loadr } -3 & \text{if } x = \textbf{this} \\ \text{loadc } a & \text{if } \rho\, x = (G, a) \\ \text{loadr } a & \text{if } \rho\, x = (L, a) \\ \text{loadr } -3 \\ \text{loadc } a \\ \text{add} & \text{if } \rho\, x = (A, a) \end{cases}$$

In particular, the pointer to the current object has relative address -3 :-)

For every class C we assume that we are given an adress environment ρ_C. ρ_C maps every identifier x visible inside C to its decorated relative address a. We distingish:

global variable	(G, a)
local variable	(L, a)
attribute	(A, a)
virtual function	(V, b)
non-virtual function	(N, a)
static function	(S, a)

For virtual functions x, we do not store the starting address of the code — but the relative address b of the field of x inside the object :-)

For the various of variables, we obtain for the L-values:

$$\text{code}_L\; x\; \rho \;=\; \begin{cases} \text{loadr } -3 & \text{if } x = \textbf{this} \\ \text{loadc } a & \text{if } \rho\, x = (G, a) \\ \text{loadr } a & \text{if } \rho\, x = (L, a) \\ \text{loadr } -3 \\ \text{loadc } a \\ \text{add} & \text{if } \rho\, x = (A, a) \end{cases}$$

In particular, the pointer to the current object has relative address -3 :-)

Accordingly, we introduce the abbreviated operations:

```
loadm q   =  loadr −3
             loadc q
             add
             load

storem q  =  loadr −3
             loadc q
             add
             store
```