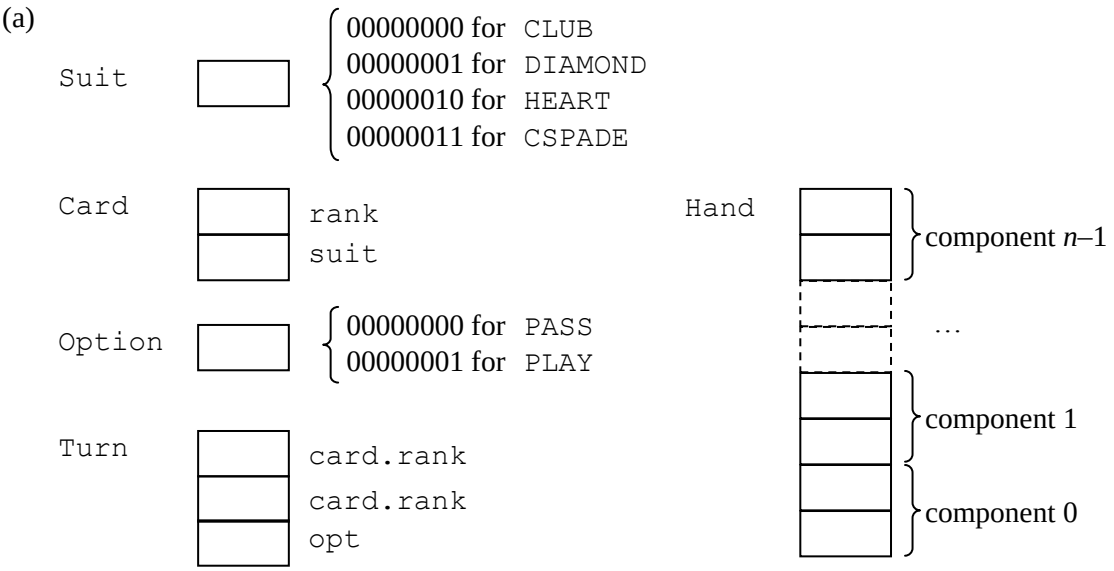
Exercises 14 (Run-time organization) – Solutions

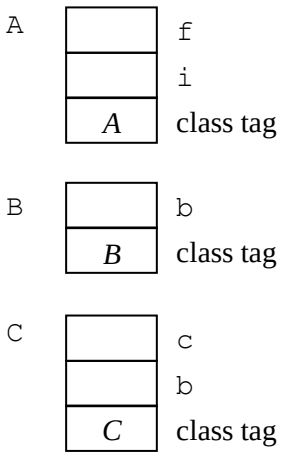
14A *(Data representation)*

The types of Exercise 2B might be represented as follows:

(a)



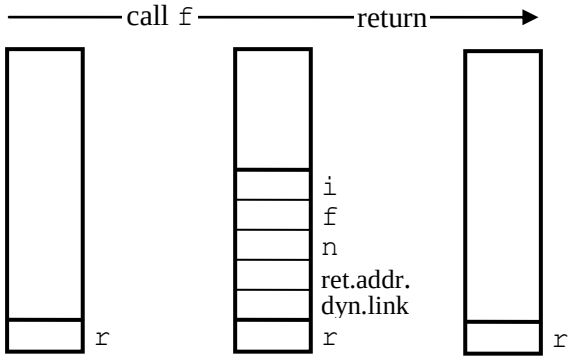
(b)



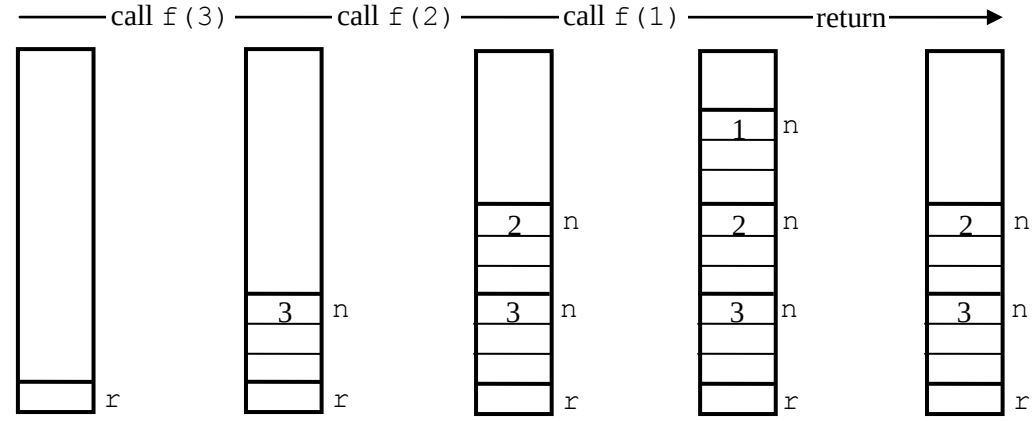
14B *(Global and local storage allocation)*

Allocation of storage to the local variables of the C programs in Exercise 9A:

(a)

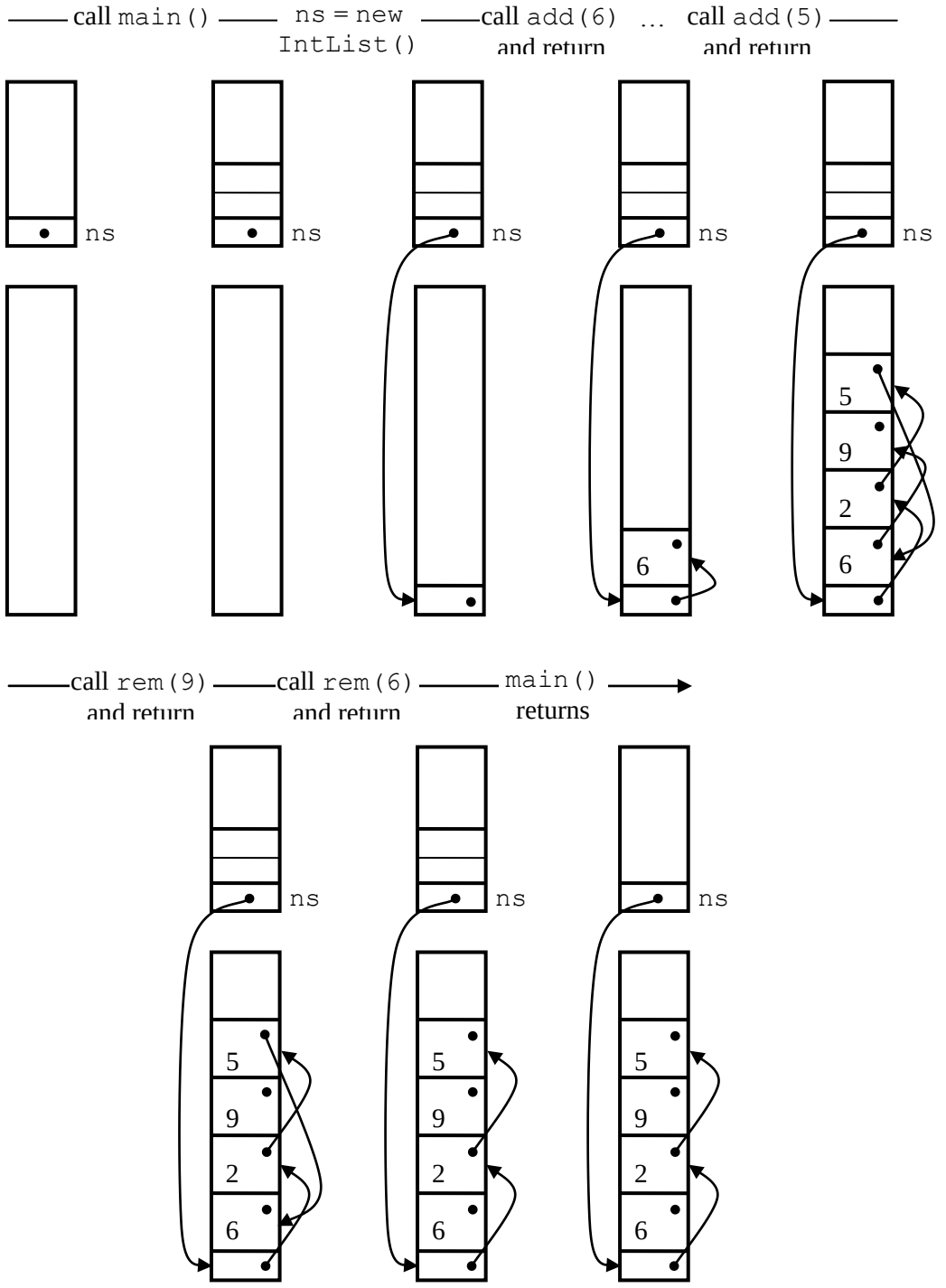


(b)



14C *(Global, local, and heap storage allocation)*

Allocation of storage to the local and heap variables of the Java program in Exercise 9B:



14D *(Garbage collection)*

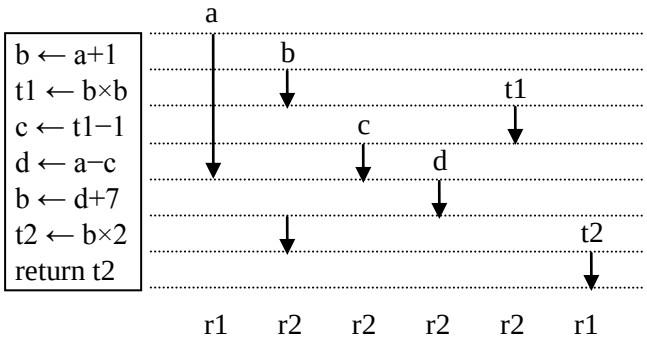
In a copying or generational garbage collector, pointers to moved heap variables could be redirected as follows.

Immediately after a heap variable *v* is copied to another space, mark the original copy of *v* and store a "forwarding address" in it. Whenever a marked heap variable *v* is accessed through a pointer *p*, simply replace *p* by the forwarding address.

Exercises 15 (Native code generation) – Solutions

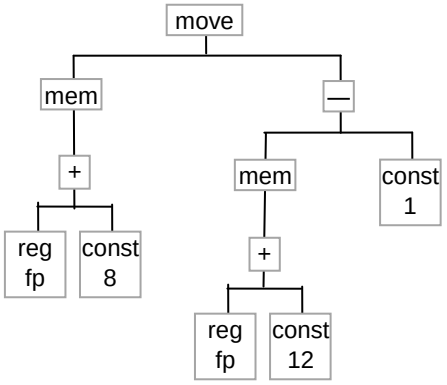
15A *(Register allocation)*

Basic block, live variables, and possible register allocation:

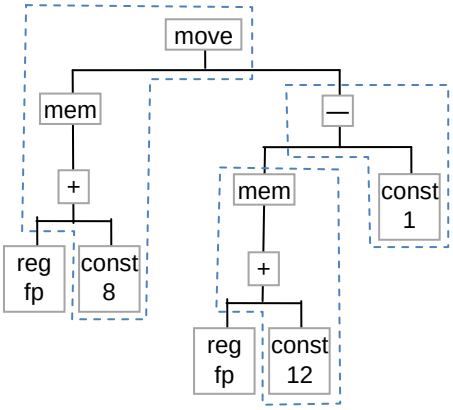


15B *(Code selection)*

(a) IR tree for C statement "`d = i - 1;`":



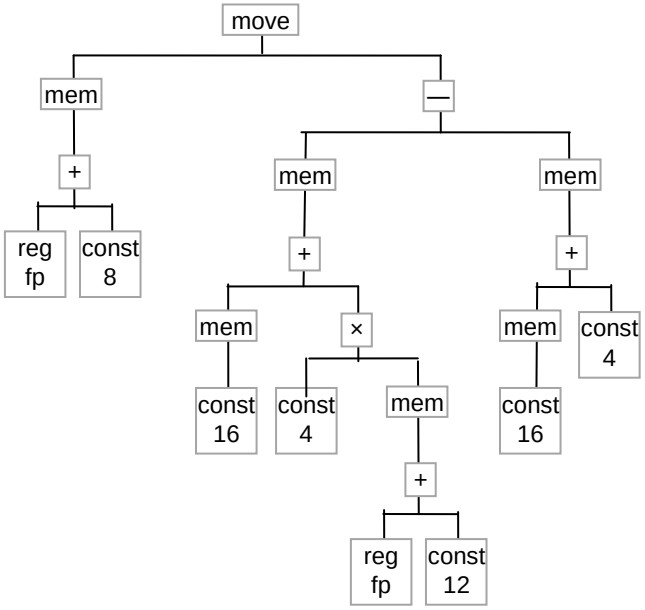
One way to cover the IR tree:



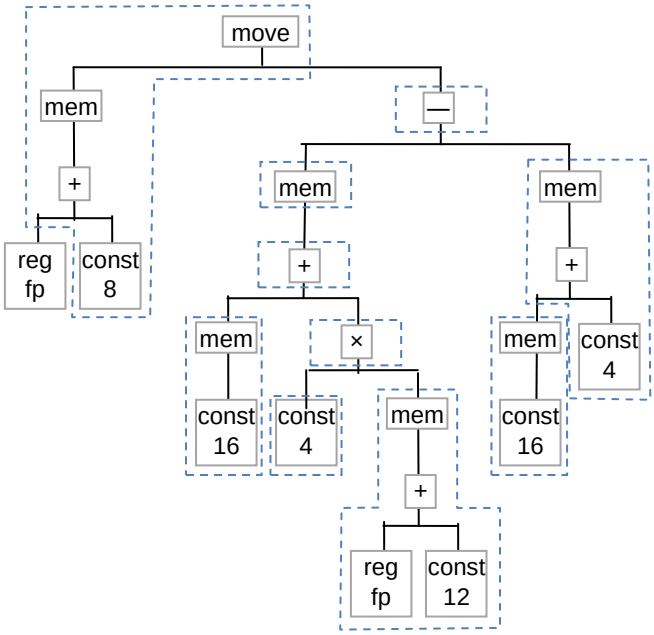
Corresponding Jouette object code:

```
LOAD  r1,12(fp)
SUBI  r1,r1,1
STORE r1,8(fp)
```

(b) IR tree for C statement "`d = a[i] - a[1];`":



One way to cover the IR tree:



Corresponding Jouette object code:

```
LOAD  r1,16(r0)
ADDI  r2,r0,4
LOAD  r3,12(fp)
MUL   r2,r2,r3
ADD   r1,r1,r2
LOAD  r1,0(r1)
LOAD  r2,16(r0)
LOAD  r2,4(r2)
SUB   r1,r1,r2
STORE r2,8(fp)
```