if ——— then

else

endif

while ——— execute

end while

for i ← vi, of [par] execute

. . . .

end for

subalg. ——— ()

return

end subalg

Function name fu ()

namefu()

return

end function

integer, string, char

[]

↑ , []

---

Implement a sorted MultiMap over a singly linked list representation:

Map : (K, V) key and value

Multimap:

Sorted Multimap :

——→ keys are seen as sorted

ex: book — carte, a rezerva, publicatia

red — roşu

blood — sânge, neam

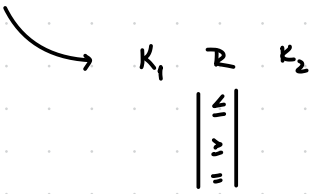
MultiMap

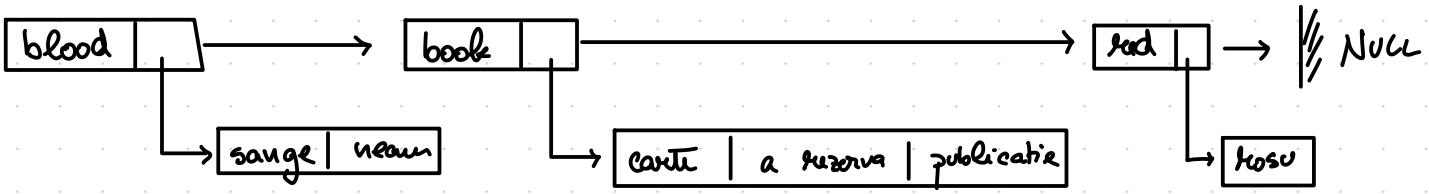
(book, carte) (book, a rezerva)

Sorted MultiMap → represented over a singly linked list


SLNode K : TKey
V : TValue
next: ↑ SLNode

SMMap:
head: ↑SLNode
R: Relation




SMMap
head: ↑SLNode
R: Relation

SLNode:
info: TElem
next: ↑SLNode

TElem:
K: TKey
VL: List

## Sorted MultiMap

```
subalg. init (smm, R)
    smm.head ← NULL
    smm.R ← R
end subalg.
```

$K_1$ R $K_2$ $\overset{def}{=}$ { true if $K_1 \leq K_2$ ; false otherwise

```
subalg destroy (smm)
    temp ← smm.head
    while temp != NIL execute
        smm.head = [smm.head].next
        destroy ([temp].info.VL)
        free (temp)
        temp ← smm.head
end subalg
```

```
Subalg findNode (smm, K, prevN, currN)
    prevN ← NIL
    currN ← smm.head
    while (currN != NIL) AND smm.R([currN].info.K, K) AND ([currN].info.K != K) execute
        prevN ← currN
        currN ← [currN].next
    end while
    if (currN != NIL) AND ([currN].info.K ≠ K) then
        currN ← NIL
    endif
End Subalg
```

```
subalg   search (sum, k, l)
    find Node (sum, k, prevN, currN)
    if currN != NIL then:
        l ← [currN]. info. VL
    else
        init (l)        // initialize the empty list
    endif
end subalg.
```

```
subalg   add (sum, k, v)
    find Node (sum, k, prevN, currN)
    if (currN == NIL)
        aux = [prevN]. next
        allocate (p)
        [prevN]. next = p
        [p] info . k = k
        init ([p]. info. VL)
        addEnd ([p]. info. VL, v)
        [p]. next = aux
    else add ([currN. LV, v)
    end if
```

```
if prevN != Nil then
else aux = sum. head
if prevN = NIL then
    sum. head ← p
```

```
subalg   remove (sum, k, v)
    find Node (sum, k, prevN, currN)
    if currN == Nil then
        return false
    else
        if [currN]. info. K ≠ K
            return false
        else.
            if isEmpty ([currN]. info. VL) ≠ true then
                found ← search ([currN]. info. VC, v)
                if found = true then
                    p ← position ([currN], info. VL, v)
                    remove ([currN]. info. VL, p, v)
                    return True
                else
                    return false
            else
                return false
```

# iterator

```
subalg init (it, smm)
    it.smm ← smm
    currL ← smm.head
    if currL ≠ NIL then
        itL ← iterator(smm[currL].info.VL)
    endif
end subalg
```

```
function valid (smm)
    if (currL ≠ NIL) then
        return true
    else
        return false
end funct.
```

```
subalg getCurrent (smm, k, v)
    if valid (smm) then
        k ← smm[currL].info.K
        v ← getCurrent (smm, itL)
    else
        @ throw Exception()
end subalg.
```

```
subalg next (it)
    if (currL ≠ NIL)
    if valid (itL) == true then
        next (itL)
    endif
    if valid (itL) == false then
        currL ← [currL].next
        if currL ≠ NIL then
            itL ← Iterator([currL].info.VL)
        endif
    endif
end subalg
```