tipus
Vector
ftipus

{Prec: $n > 0$}
funció *CrearVector*(**ent** n : **enter**) **retorna** *Vector*
{Post: *CrearVector*(n) és un vector amb interval $[0..n-1]$}

{Prec: **cert**}
funció *DimVector*(**ent** v : *Vector*) **retorna enter**
{Post: *DimVector*(v) és la dimensió del vector v}

$\{$Prec: $0 \leq i < DimVector(v)\}$
acció $AssigVector($**entsor** v : $Vector,$ **ent** i : **enter**, **ent** r : **real**$)$
$\{$Post: $v[i] = r\}$

$\{$Prec: $0 \leq i < DimVector(v)\}$
funció $ConsVector($**ent** v : $Vector,$ **ent** i : **enter**$)$ **retorna real**
$\{$Post: $ConsVector(v,i) = v[i]\}$

$\{$Prec: **cert**$\}$
acció $DestruirVector($**entsor** v : $Vector$$)$
$\{$Post: v destruït$\}$

```
const
    N: enter = 30
fconst


tipus
    Vector = taula [0..N-1] de real
ftipus
```

```
funció CrearVector(ent n: enter) retorna Vector
  var
    v: Vector
  fvar
  { n = N }
  retorna v
ffunció

funció DimVector(ent v: Vector) retorna enter
  retorna N
ffunció
```

```
acció AssigVector(entsor v: Vector,
                  ent i: enter,
                  ent r: real)
  v[i] := r
facció


funció ConsVector(ent v: Vector, ent i: enter)
                  retorna real
  retorna v[i]
ffunció
```

```
const
    MaxDim: enter = 50
fconst


tipus
    TauReal = taula [0..MaxDim-1] de real
    Vector = tupla
                dim: enter
               telem: TauReal
            ftupla
ftipus
```

```
funció CrearVector(ent n: enter) retorna Vector
  var
    v: Vector
  fvar
  { n > 0 ∧ n ≤ MaxDim }
  v.dim := n
  retorna v
ffunció

funció DimVector(ent v: Vector) retorna enter
  retorna v.dim
ffunció
```

```
acció AssigVector(entsor v: Vector,
                  ent i: enter,
                  ent r: real)
  v.telem[i] := r
facció


funció ConsVector(ent v: Vector, ent i: enter)
                  retorna real
  retorna v.telem[i]
ffunció
```

```
const
  MaxNoNul: enter = 100
fconst

tipus
  IndVal = tupla
             index: enter
             valor: real
           ftupla
  TauIndVal = taula [0..MaxNoNul−1] de IndVal
  Vector = tupla
             dim: enter
             niv: enter
             tiv: TauIndVal
           ftupla
ftipus
```

```
funció CrearVector(ent n: enter) retorna Vector
  var
    v: Vector
  fvar
  { n > 0 }
  v.dim := n
  v.niv := 0
  retorna v
ffunció

funció DimVector(ent v: Vector) retorna enter
  retorna v.dim
ffunció
```

```
acció AssigVector(entsor v: Vector,
                  ent i: enter,
                  ent r: real)
  var
    p: enter
    trobat: booleà
  fvar
  CercarTauIndVal(v, i, trobat, p)

  ...
```

```
  ...

  si
     trobat ∧ r ≠ 0.0 ⟶
       v.tiv[p].valor := r
     trobat ∧ r = 0.0 ⟶
       v.tiv[p] := v.tiv[v.niv−1]
       v.niv := v.niv−1
    ¬trobat ∧ r ≠ 0.0 ⟶{ v.niv < MaxNoNul }
       v.tiv[v.niv].index := i
       v.tiv[v.niv].valor := r
       v.niv := v.niv+1
    ¬trobat ∧ r = 0.0 ⟶
  fsi
facció
```

```
funció ConsVector(ent v: Vector, ent i: enter)
                    retorna real
  var
    p: enter
    r: real
  fvar
  CercarTauIndVal(v, i, trobat, p)
  si
     trobat⟶r := v.tiv[p].valor
    ¬trobat⟶r := 0
  fsi
  retorna r
ffunció
```

```
acció CercarTauIndVal(ent v: Vector, ent i: enter,
                      sor t: booleà, sor p: enter)
  t := fals
  p := 0
  mentre ¬(p ≥ v.niv) ∧ ¬t fer
    t := v.tiv[p].index = i
    si
        t ⟶
      ¬t ⟶ p := p + 1
    fsi
  fmentre
facció
```

```
typedef struct {
    float *telem; /* Taula d'elements */
    int dim;      /* Dimensio */
} Vector;


int
DimVector(Vector v)
{
    return v.dim;
}
```

```
Vector
CrearVector(int n)
{
    Vector v;

    assert(n > 0);

    v.dim = n;
    v.telem = malloc(n * sizeof (float));
    assert(v.telem != NULL);
    return v;
}
```

```
void
AssigVector(Vector * const v, int i, float f)
{
    assert(0 <= i && i < (*v).dim);

    (*v).telem[i] = f;
}

float
ConsVector(Vector v, int i)
{
    assert(0 <= i && i <= v.dim);

    return v.telem[i];
}
```

```
void
DestruirVector(Vector * const v)
{
    free((*v).telem);
}
```

tipus
$Matriu$
ftipus

$\{$Prec: $nf > 0 \ \wedge \ nc > 0\}$
funció $CrearMatriu($**ent** nf, nc : **enter**$)$ **retorna** $Matriu$
$\{$Post: $CrearMatriu(nf, nc)$ és una matriu $[0..nf-1, 0..nc-1]\}$

$\{$Prec: $cert\}$
acció $DestruirMatriu($**entsor** m : $Matriu)$
$\{$Post: m destruïda$\}$

$\{$Prec: *cert*$\}$
funció *FilesMatriu*$($**ent** *m* : *Matriu*$)$ **retorna enter**
$\{$Post: *FilesMatriu*(m) és el nombre de files de la matriu *m*$\}$

$\{$Prec: *cert*$\}$
funció *ColsMatriu*$($**ent** *m* : *Matriu*$)$ **retorna enter**
$\{$Post: *ColsMatriu*(m) és el nombre de columnes de la matriu *m*$\}$

$\{$Prec: $0 \leq i < \mathit{FilesMatriu}(m) \ \wedge \ 0 \leq j < \mathit{ColsMatriu}(m)\}$
acció $\mathit{AssigMatriu}($**entsor** m : Matriu, **ent** i, j : **enter**, **ent** r : **real**$)$
$\{$Post: $m[i, j] = r\}$

$\{$Prec: $0 \leq i < \mathit{FilesMatriu}(m) \ \wedge \ 0 \leq j < \mathit{ColsMatriu}(m)\}$
funció $\mathit{ConsMatriu}($**ent** m : Matriu, **ent** i, j : **enter**$)$ **retorna real**
$\{$Post: $\mathit{ConsMatriu}(m, i, j) = m[i, j]\}$

$\{$Prec: $0 \leq i, j < \mathit{FilesMatriu}(m) \ \wedge \ m = M\}$
acció $\mathit{IntFilesMatriu}($**entsor** $m : \mathit{Matriu}$, **ent** $i, j :$ **enter**$)$
$\{$Post: $(\forall \alpha, \beta : 0 \leq \alpha < \mathit{FilesMatriu}(m) \ \wedge \ \alpha \neq i \ \wedge \ \alpha \neq j \ \wedge$
$0 \leq \beta < \mathit{ColsMatriu}(m) : m[\alpha, \beta] = M[\alpha, \beta])$
$(\forall \beta : 0 \leq \beta < \mathit{ColsMatriu}(m) : m[i, \beta] = M[j, \beta] \ \wedge \ m[j, \beta] = M[i, \beta])\}$

$\{$Prec: $0 \leq i, j < \mathit{ColsMatriu}(m) \ \wedge \ m = M\}$
acció $\mathit{IntColsMatriu}($**entsor** $m : \mathit{Matriu}$, **ent** $i, j :$ **enter**$)$
$\{$Post: $(\forall \alpha, \beta : 0 \leq \alpha < \mathit{ColsMatriu}(m) \ \wedge \ \alpha \neq i \ \wedge \ \alpha \neq j \ \wedge$
$0 \leq \beta < \mathit{FilesMatriu}(m) : m[\alpha, \beta] = M[\alpha, \beta])$
$(\forall \beta : 0 \leq \beta < \mathit{FilesMatriu}(m) : m[\beta, i] = M[\beta, j] \ \wedge \ m[\beta, j] = M[\beta, i])\}$

```
const
  N: enter = 30
  M: enter = 50
fconst


tipus
  Matriu = taula [0..N-1, 0..M-1] de real
ftipus
```

```
funció CrearMatriu(ent f, c: enter) retorna Matriu
  var
    m: Matriu
  fvar
  { f = N }
  { c = M }
  retorna m
ffunció

funció FilesMatriu(ent m: Matriu) retorna enter
  retorna N
ffunció

funció ColsMatriu(ent m: Matriu) retorna enter
  retorna M
ffunció
```

```
acció AssigMatriu(entsor m: Matriu,
                  ent i, j: enter,
                  ent r: real)
  { 0 ≤ i ∧ i < N }
  { 0 ≤ j ∧ j < M }
  m[i, j] := r
facció

funció ConsMatriu(ent m: Matriu, ent i, j: enter)
                  retorna real
  { 0 ≤ i ∧ i < N }
  { 0 ≤ j ∧ j < M }
  retorna m[i, j]
ffunció
```

```
const
   MaxFiles: enter = 50
   MaxCols: enter = 70
fconst

tipus
   TauReal = taula [0..MaxFiles−1, 0..MaxCols−1]
                de real
   Matriu = tupla
               files, cols: enter
               telem: TauReal
            ftupla
ftipus
```

```
funció CrearMatriu(ent f, c: enter) retorna Matriu
  var
    m: Matriu
  fvar
  { f > 0 ∧ n ≤ MaxFiles }
  { c > 0 ∧ c ≤ MaxCols }
  m.files := f
  m.cols := c
  retorna m
ffunció
```

```
funció FilesMatriu(ent m: Matriu) retorna enter
  retorna m.files
ffunció
```

```
funció ColsMatriu(ent m: Matriu) retorna enter
  retorna m.cols
ffunció
```

```
acció AssigMatriu(entsor m: Matriu,
                  ent i, j: enter,
                  ent r: real)
  { 0 ≤ i ∧ i < m.files }
  { 0 ≤ j ∧ j < m.cols }

  m.telem[i, j] := r
facció

funció ConsMatriu(ent m: Matriu, ent i: enter)
                  retorna real
  { 0 ≤ i ∧ i < m.files }
  { 0 ≤ j ∧ j < m.cols }

  retorna m.telem[i, j]
ffunció
```

```
const
    MaxElems: enter = 350
fconst


tipus
    TauReal = taula [0..MaxElems-1] de real
    Matriu = tupla
                files, cols: enter
                telem: TauReal
             ftupla
ftipus
```

```
funció CrearMatriu(ent f, c: enter) retorna Matriu
  var
    m: Matriu
  fvar
  { f > 0 ∧ n ≤ MaxFiles }
  { c > 0 ∧ c ≤ MaxCols }
  m.files := f
  m.cols := c
  retorna m
ffunció
```

```
funció FilesMatriu(ent m: Matriu) retorna enter
  retorna m.files
ffunció
```

```
funció ColsMatriu(ent m: Matriu) retorna enter
  retorna m.cols
ffunció
```

```
acció AssigMatriu(entsor m: Matriu,
                  ent i, j: enter,
                  ent r: real)
  { 0 ≤ i ∧ i < m.files }
  { 0 ≤ j ∧ j < m.cols }

  m.telem[i*m.cols+j] := r
facció

funció ConsMatriu(ent m: Matriu, ent i: enter)
                  retorna real
  { 0 ≤ i ∧ i < m.files }
  { 0 ≤ j ∧ j < m.cols }

  retorna m.telem[i*m.cols+j]
ffunció
```

```
const
  MaxNoNul: enter = 100
fconst

tipus
  IndVal = tupla
             index: enter
             valor: real
           ftupla
  TauIndVal = taula [0..MaxNoNul−1] de IndVal
  Matriu = tupla
             files, cols: enter
             niv: enter
             tiv: TauIndVal
           ftupla
ftipus
```

```
funció CrearMatriu(ent f, c: enter) retorna Matriu
  var
    m: Matriu
  fvar
  { f > 0 }
  { c > 0 }
  m.files := f
  m.cols := c
  m.niv := 0
  retorna m
ffunció
```

```
funció FilesMatriu(ent m: Matriu) retorna enter
  retorna m.files
ffunció
```

```
funció ColsMatriu(ent m: Matriu) retorna enter
  retorna m.cols
ffunció
```

```
acció AssigMatriu(entsor m: Matriu,
                  ent i, j: enter,
                  ent r: real)
  var
    p: enter
    trobat: booleà
  fvar
  CercarTauIndVal(m, i*m.cols+j, trobat, p)

  ...
```

```
...

si
    trobat ∧ r ≠ 0.0 ⟶
      m.tiv[p].valor := r
    trobat ∧ r = 0.0 ⟶
      m.tiv[p] := m.tiv[m.niv−1]
      m.niv := m.niv−1
  ¬trobat ∧ r ≠ 0.0 ⟶{ m.niv < MaxNoNul }
      m.tiv[m.niv].index := i∗m.cols+j
      m.tiv[m.niv].valor := r
      m.niv := m.niv+1
  ¬trobat ∧ r = 0.0 ⟶
fsi
facció
```

```
funció ConsMatriu(ent m: Matriu, ent i: enter)
                    retorna real
  var
    p: enter
    r: real
  fvar
  CercarTauIndVal(m, i*m.cols+j, trobat, p)
  si
     trobat⟶r := m.tiv[p].valor
    ¬trobat⟶r := 0
  fsi
  retorna r
ffunció
```

```
acció CercarTauIndVal(ent m: Matriu, ent i: enter,
                      sor t: booleà, sor p: enter)
  t := fals
  p := 0
  mentre ¬(p ≥ m.niv) ∧ ¬t fer
    t := m.tiv[p].index = i
    si
        t ⟶
      ¬t ⟶ p := p + 1
    fsi
  fmentre
facció
```

```
typedef struct {
    float *telem; /* Taula d'elements */
    int nf;       /* Nombre de files */
    int nc;       /* Nombre de columnes */
} Matriu;


void
DestruirMatriu(Matriu * const m)
{
    free((*m).telem);
}
```

```
int
FilesMatriu(Matriu m)
{
    return m.nf;
}

int
ColsMatriu(Matriu m)
{
    return m.nc;
}
```

```
Matriu
CrearMatriu(int nf, int nc)
{
    Matriu m;

    assert(nf > 0 && nc > 0);

    m.nf = nf;
    m.nc = nc;
    m.telem = malloc(nf * nc * sizeof(float));
    assert(m.telem != NULL);
    return m;
}
```

```
void
AssigMatriu(Matriu * const m, int i, int j,
            float f)
{
    assert(0 <= i && i < (*m).nf);
    assert(0 <= j && j < (*m).nc);

    (*m).telem[i * (*m).nc + j] = f;
}
```

```
float
ConsMatriu(Matriu m, int i, int j)
{
    assert(0 <= i && i < m.nf);
    assert(0 <= j && j < m.nc);

    return m.telem[i * m.nc + j];
}
```

```
void
IntFilesMatriu(Matriu * const m, int i, int j)
{
  int k;

  assert(0 <= i && i < (*m).nf);
  assert(0 <= j && j < (*m).nf);

  k = 0;
  while (k != (*m).nc) {
    InterReal(&(*m).telem[i * (*m).nc + k],
              &(*m).telem[j * (*m).nc + k]);
    k = k + 1;
  }
}
```

```
void
IntColsMatriu(Matriu * const m, int i, int j)
{
  int k;

  assert(0 <= i && i < (*m).nf);
  assert(0 <= j && j < (*m).nf);

  k = 0;
  while (k != (*m).nf) {
    InterReal(&(*m).telem[k * (*m).nc + i],
              &(*m).telem[k * (*m).nc + j]);
    k = k + 1;
  }
}
```

```
typedef struct {
    float **telem; /* Taula d'apuntadors a
                      taules d'elements */
    int nf;        /* Nombre de files */
    int nc;        /* Nombre de columnes */
} Matriu;
```

```
int
FilesMatriu(Matriu m)
{
    return m.nf;
}

int
ColsMatriu(Matriu m)
{
    return m.nc;
}
```

```
Matriu
CrearMatriu(int nf, int nc)
{
    Matriu m;
    int i;

    assert(nf > 0 && nc > 0);

    m.nf = nf;
    m.nc = nc;
    m.telem = malloc(nf * sizeof (float *));
    assert(m.telem != NULL);

    ...
}
```

```
Matriu
CrearMatriu(int nf, int nc)
{
    ...

    i = 0;
    while (i != nf) {
      m.telem[i] = malloc(nc * sizeof (float));
       assert(m.telem[i] != NULL);
       i = i + 1;
    }
    return m;
}
```

```
void
DestruirMatriu(Matriu * const m)
{
    int i;

    i = 0;
    while (i != (*m).nf) {
      free((*m).telem[i]);
      i = i + 1;
    }
    free((*m).telem);
}
```

```
void
AssigMatriu(Matriu *const m, int i, int j,
            float f)
{
    assert(0 <= i && i < (*m).nf);
    assert(0 <= j && j < (*m).nc);

    (*m).telem[i][j] = f;
}
```

```
float
ConsMatriu(Matriu m, int i, int j)
{
    assert(0 <= i && i < m.nf);
    assert(0 <= j && j < m.nc);

    return m.telem[i][j];
}
```

```
void
IntFilesMatriu(Matriu * const m, int i, int j)
{
  assert(0 <= i && i < (*m).nf);
  assert(0 <= j && j < (*m).nc);

  InterApReal(&(*m).telem[i], &(*m).telem[j]);
}
```

```
void
IntColsMatriu(Matriu * const m, int i, int j)
{
  int k;

  assert(0 <= i && i < (*m).nf);
  assert(0 <= j && j < (*m).nc);

  k = 0;
  while (k != (*m).nf) {
    InterReal(&(*m).telem[k][i],
              &(*m).telem[k][j]);
    k = k + 1;
  }
}
```

tipus $TauFreq$ **ftipus**

$\{$Prec: $cert\}$
acció $Inicialitzar($**sor** $t : TauFreq)$
$\{$Post: t és una taula de freqüències buida$\}$

$\{$Prec: $Freq(t,x) = f\}$
acció $Afegir($**entsor** $t : TauFreq,$ **ent** $x : Element)$
$\{$Post: $Freq(t,x) = f+1\}$

$\{$Prec: $cert\}$
funció $Freq($**ent** $t : TauFreq,$ **ent** $x : Element)$ **retorna enter**
$\{$Post: $\exists i : (element_i, frequencia_i) \in t \ \wedge \ x = element_i$
$\wedge \ Freq(t,x) = frequencia_i \ \vee \ Freq(t,x) = 0\}$