Tipos Abstratos de Dados

Essa rotina pode ser descrita como:

1. Seja a o maior entre o numerador e o denominador e b o menor.
2. Divida a por b, encontrando um quociente q e um resto r (isto é, $a = q * b + r$).
3. Defina $a = b$ e $b = r$.
4. Repita os passos 2 e 3 até que b seja igual a 0.
5. Divida tanto o numerador quanto o denominador pelo valor de a.

Implemente esta rotina em C.

UNIVASF

```
void reduzir_racional (RACIONAL *inrat,
RACIONAL *outrat)
{
   int a, b, rem;
   if (inrat->num > inrat->den)
   {
      a = inrat->num;
      b = inrat->den;
   }
   else
   {
      a = inrat->den;
      b = inrat->num;
   }
```

```
  while (b)
  {
    rem = a%b;
    a = b;
    b = rem;
  }
  outrat->num = inrat->num/a;
  outrat->den = inrat->den/a;
}
```

Listas – Caracterização e Alocação Sequencial

UNIVASF

Caracterização

Uma lista (linear) é uma sequência de zero ou mais elementos, cada um deles sendo um valor primitivo ou composto, e, para $k \in [1..n]$,

- x_1 é o primeiro elemento;
- x_k é o antecessor de x_{k+1};
- x_k é o sucessor de x_{k-1};
- $x_2, x_3, ..., x_n$ compõem o resto da lista;
- x_n é o último elemento.

Caracterização

A estrutura de lista representa a ordem linear entre os elementos.

Diz-se que o elemento x_k ocupa a posição k da lista e n é o tamanho da lista.

Se n=0, a lista é vazia.

A lista é um objeto particularmente flexível, o seu número de elementos pode variar.

As listas podem ser homogêneas ou heterogenias.

UNIVASF

Caracterização

As listas podem ser lineares ou generalizadas.

As listas têm um extenso universo de aplicações:

- armazenamento e recuperação de informações;
- processamento de linguagens;
- simulação de processos;
- sistemas operacionais;
- inteligência artificial;
- e muito outros campos.

UNIVASF

Caracterização

Em geral, as aplicações necessitam de um conjunto de operações primitivas que permitem criar uma nova lista, verificar se a lista é vazia, acessar o k-ésimo elemento, inserir um elemento como k-ésimo da lista, remover o k-ésimo elemento, etc.

A partir dessas operações básicas pode-se definir outras como: verificar se um elemento está na lista ou localizar sua posição, dividir ou combinar listas, invertê-las, aplicar sobre elas algum operador elemento a elemento, compará-las, etc.

UNIVASF

Alocação Sequencial

Como os elementos de uma lista se dispõem conceitualmente de forma consecutiva, a disposição física dos mesmos no modo sequencial é intuitiva.

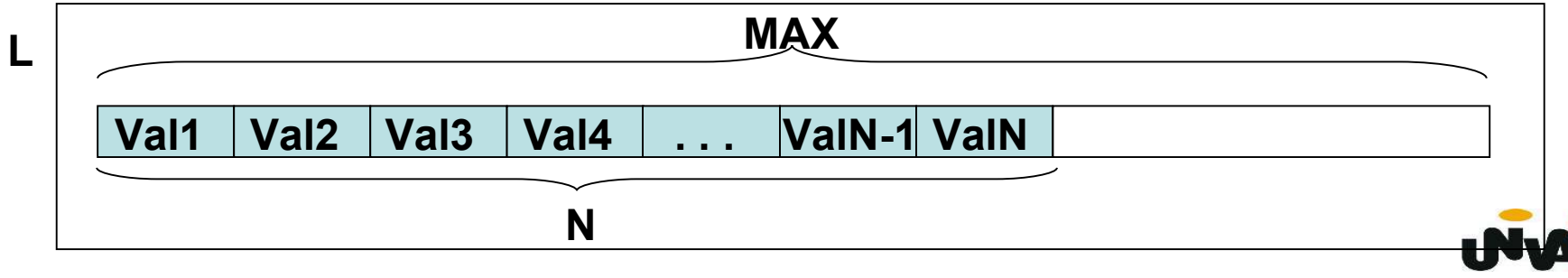
Como no caso dos arranjos unidimensionais, o endereço de um nodo poderá ser calculado com base no endereço do primeiro nodo, se todos os nodos ocuparem blocos de memória de mesmo tamanho.

Assim a contiguidade física é particularmente adequada para o conjunto de operações, em que referências ao k-ésimo elemento são comuns.

UNIVASF

Alocação Sequencial

Logo, pode-se basear a representação da lista em um vetor para conter os elementos da lista, esta começando no primeiro elemento do vetor, associado a um contador que indica o número de elementos efetivamente utilizados do vetor, ou seja, o número de elementos na lista. Observe o esquema



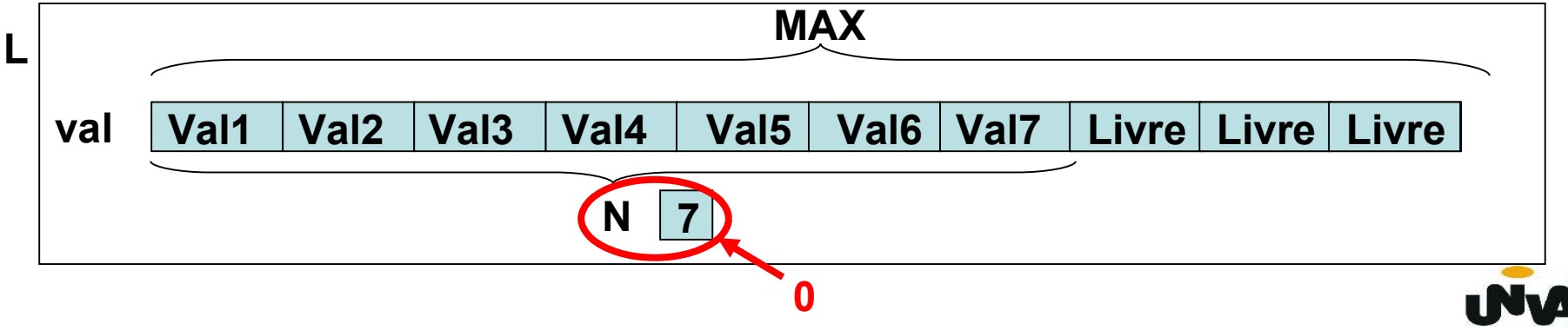
Desta forma, a definição para o TAD lista (de inteiros) seria:

```
typedef struct
{
        int N;          /*numero de elementos*/
        int val[max];   /*vetor de elementos.*/
}LISTA;
void cria_lista (LISTA *);
int eh_vazia (LISTA *);
int tam (LISTA *);
void ins (LISTA *, int, int);
int recup (LISTA *, int);
void ret (LISTA *, int);
```

Alocação Sequencial

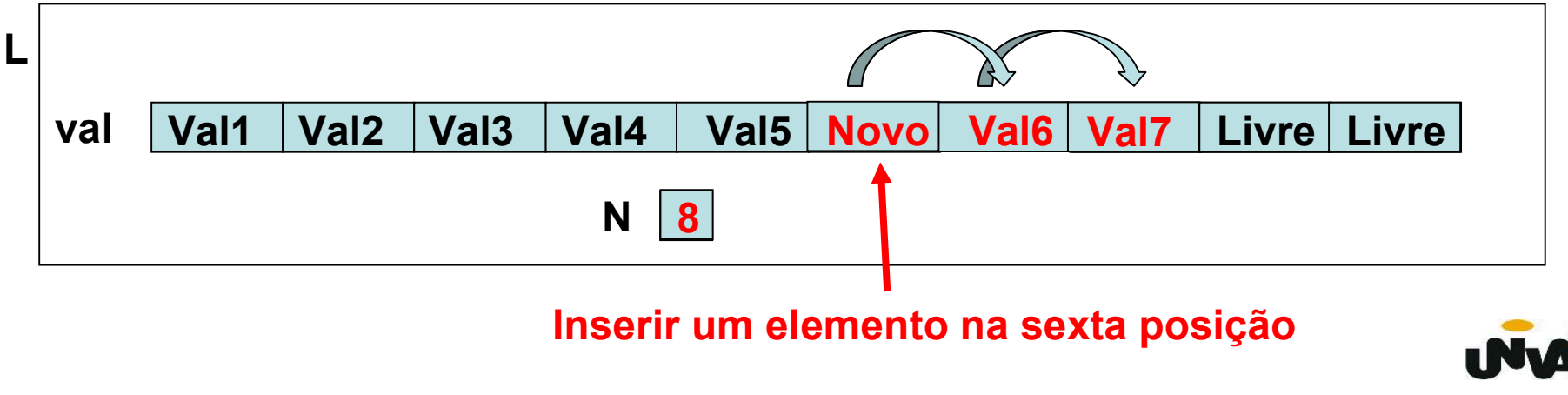
Antes de efetuarmos a implementação das operações, algumas observações se fazem importantes:

- criar a lista corresponde a zerar o contador;
- o contador é testado para verificar se a lista é vazia, sendo o tamanho da mesma o valor do contador;



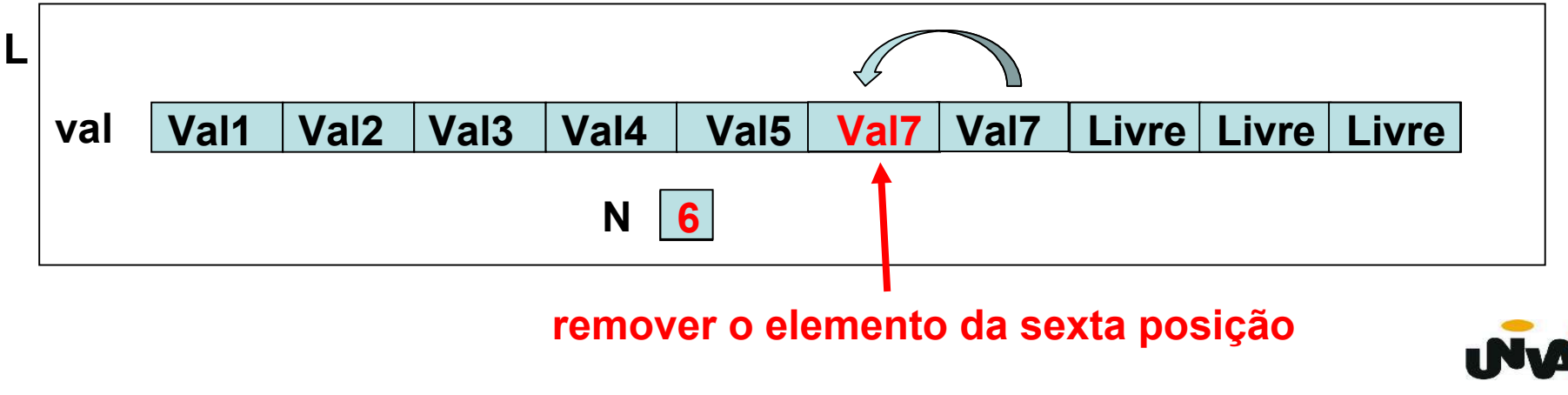
Alocação Sequencial

- inserir um elemento na k-ésima posição exigirá abrir espaço, deslocando todos os elementos posteriores uma casa em direção ao fim do vetor, armazenando-se então o novo elemento da lista na lacuna obtida, incrementando-se o contador;



Alocação Sequencial

- a retirada, inversamente, exigirá uma compactação, pelo recuo dos elementos posteriores ao que tiver sido retirado, e o decremento do contador.



Alocação Sequencial

Operações de inserção só podem ser efetuadas para $k \in$ [1..tam(L)+1], desde que tam (L) < max. Retiradas só são válidas para $k \in$ [1..tam(L)].

Com base nestas observações implemente as operações do TAD LISTA.

UNIVASF

```
typedef struct
{
        int N;      /*numero de elementos*/
        int val[max]; /*vetor de elementos*/
}LISTA;
void cria_lista (LISTA *);
int eh_vazia (LISTA *);
int tam (LISTA *);
void ins (LISTA *, int, int);
int recup (LISTA *, int);
void ret (LISTA *, int);
```

```
void cria_lista (LISTA *l)
{
   l->N = 0;
}
int eh_vazia (LISTA *l)
{
   return (l->N == 0);
}
int tam (LISTA *l)
{
   return (l->N);
}
```

```
void ins (LISTA *l, int v, int k) {
  int i;
  if (l->N == max)
  {
    printf ("\nERRO! Estouro na lista.\n");
    exit (1);
  }
  else
    if (k < 1 || k > l->N+1)
    {
      printf ("\nERRO! Posição invalida para insercao.\n");
      exit (2);
    }
```

```
    for (i=l->N; i>=k; i--)
        l->val[i]=l->val[i-1];
    l->val[k-1]=v;
    l->N++;
}
```

```
int recup (LISTA *l, int k)
{
   if (k < 1 || k > l->N)
   {
      printf ("\nERRO! Consulta invalida.\n");
      exit (3);
   }
   else
      return (l->val[k-1]);
}
```

```
void ret (LISTA *l, int k)
{
   int i;
   if (k < 1 || k > l->N)
   {
      printf ("\nERRO! Posicao invalida para retirada.\n");
      exit (4);
   }
   l->N--;
   for (i=k-1; i<l->N; i++)
      l->val[i]=l->val[i+1];
}
```

Alocação Sequencial - Exercício

Implemente, no TAD LISTA, a seguinte operação:

```
int pertence (LISTA *l, int v);
```

a qual retorna 1 (um) se **v** pertence a lista apontada por **l** e 0 (zero) caso contrário.

Alocação Sequencial - Exercício

Implemente, no TAD LISTA, a seguinte operação:

int eh_ord (LISTA *l);

a qual retorna 1 (um) se a lista apontada por **l** está em ordem crescente e 0 (zero) caso contrário.

Alocação Sequencial - Exercício

Implemente, no TAD LISTA, utilizando recursividade, a seguinte operação:

void gera_lista (LISTA *l, int m, int n);

a qual utilizando-se das operações do TAD LISTA produz uma lista de inteiros correspondente a [m..n].