Listas

Caracterização

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

- x_1 é o primeiro elemento ou cabeça da lista;
- x_k é o antecessor ou predecessor 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

Em outras palavras, 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, dado que o seu número de elementos pode variar, ou seja: ela pode aumentar ou diminuir de tamanho. Os elementos podem ser todos do mesmo tipo, quando se têm listas homogêneas, ou podem ser de tipos diferentes, nas listas heterogêneas.

Caracterização

As listas podem ser lineares, se só podem conter átomos (objetos de tipo qualquer, simples ou composto, exceto listas), ou genéricos, se podem conter como elementos outras listas. Têm um extenso universo de aplicações, notadamente em armazenamento e recuperação de informações, processamento de linguagens, simulação de processos, sistemas operacionais, inteligência artificial e muito outros campos.

Caracterização

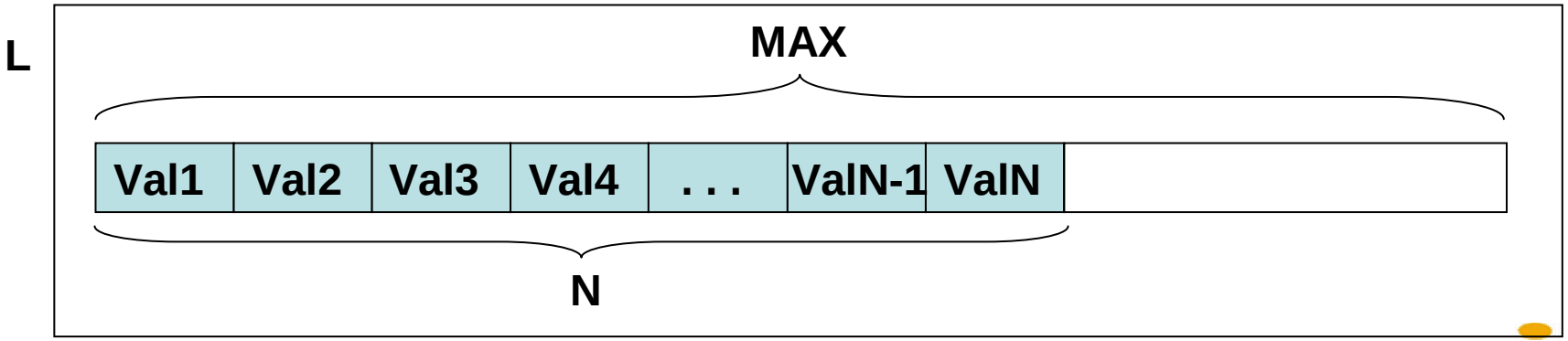
Em geral, as aplicações necessitam de um conjunto de operações primitivas que permitem criar uma nova lista, verificar se a lista é vazia, acessar o k-ésimo elemento, inserir um elemento como k-ésimo da lista, remover o k-ésimo elemento, etc. A partir dessas operações básicas pode-se definir outras como: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

Alocação Seqüencial

Como os elementos de uma lista se dispõem conceitualmente de forma consecutiva, a disposição física dos mesmos no modo seqüencial 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 tiverem num único bloco de memória. Assim a contigüidade física é particularmente adequada para o conjunto de operações, em que referências ao k-ésimo elemento são comuns.

Alocação Seqüencial

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 Seqüencial

Antes de efetuarmos a implementação das operações, algumas observações sobre elas se fazem importantes: criar a lista corresponde a zerar o contador; também o contador é testado para verificar se a lista é vazia, sendo o tamanho da mesma o valor do contador; 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 no final o contador;

Alocação Seqüencial

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

Implementaremos agora as operações do TAD LISTA.

```
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! Posição invalida para
       retirada.\n");
      exit (4);
   }
   l->N--;
   for (i=k-1; i<l->N; i++)
      l->val[i]=l->val[i+1];
}
```

Alocação Seqüencial - 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 Seqüencial - 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 Seqüencial - 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].