Algoritmos e Estruturas de dados

Listas Encadeadas

Prof. Dr. Fábio Rodrigues de la Rocha

UTFPR

Definição:

Anteriormente estudamos listas encadeadas que foram implementadas como vetores na memória. Aquele tipo de lista assume que os elementos estão sequencialmente na memória do computador. Ou seja, se o elemento 0 do vetor de caracteres está no endereço 1000 o elemento 1 estará no endereço 1001.
Existem alguns problemas na implementação que faz uso de vetores:

- Precisa-se conhecer de antemão a quantidade máxima de elementos; ex: `Tipo_Lista elementos [100];`
- A operação de eliminação não elimina realmente os elementos, ou seja, não libera a memória que estava em uso pelo tal elemento para o SO.

UTFPR

Listas como vetores

Quando utilizamos vetores, a memória é alocada estaticamente. Ou seja, quando o programa é **iniciado** toda a memória do vetor é alocada, antes mesmo de as instruções do programa serem executadas. A memória permanece alocada até o programa terminar.

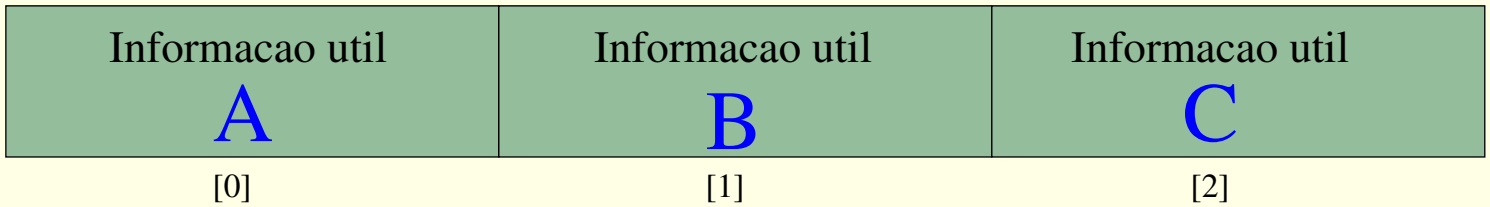
```
1  typedef struct {
2      int     x;
3      float   y;
4      double  z;
5  }Meu_Tipo;
6
7  int main (void) {
8      Meu_Tipo vetor[10000];
9      .
10     .
11 }
```

Listas como vetores

Vetores quando utilizados para implementar listas possuem a característica de que os elementos estão sempre numa sequencia.

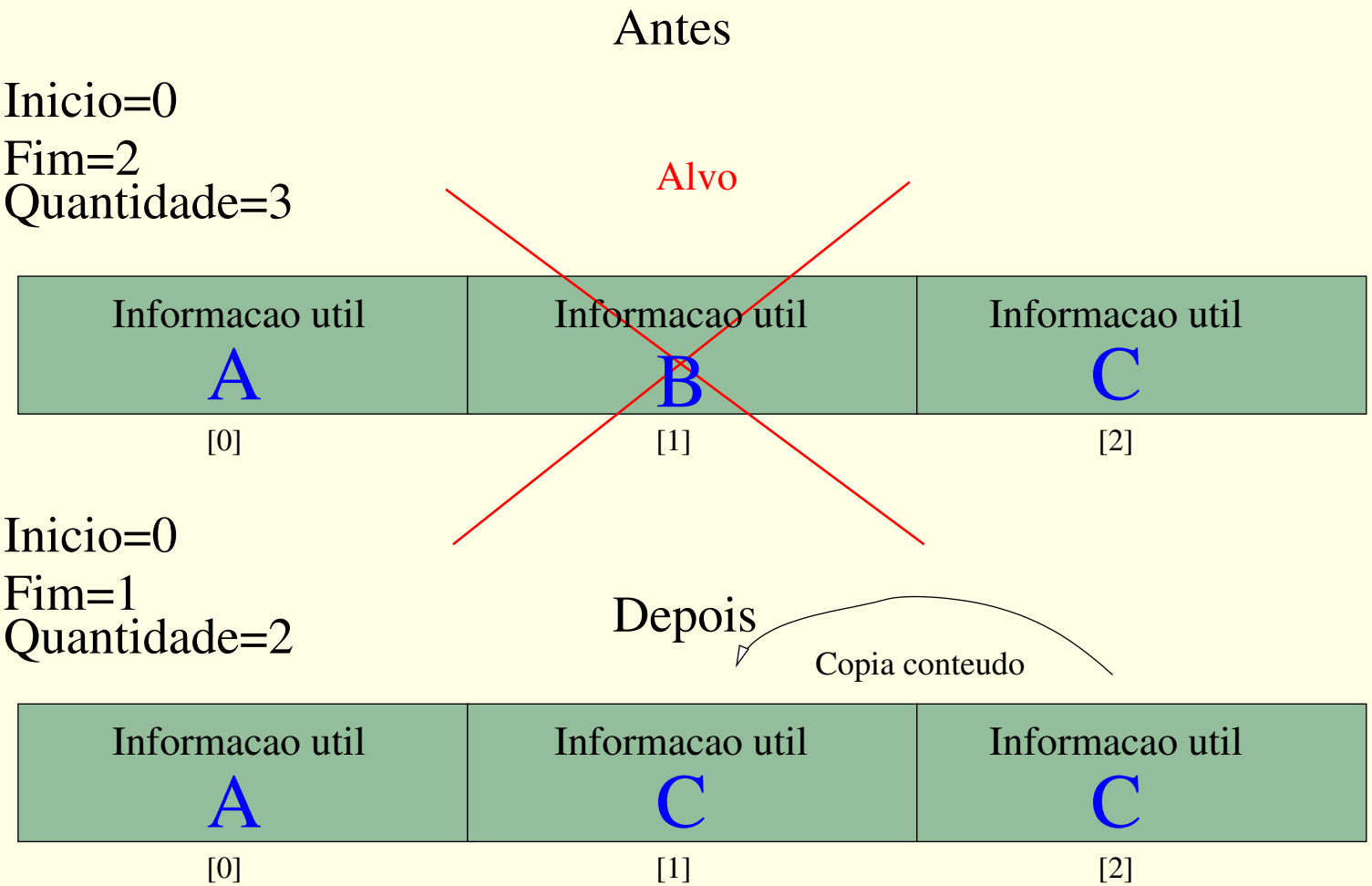
Alocacao sequencial ou estatica

Inicio=0
Fim=2
Quantidade=3



Listas como vetores

Mas e se desejarmos eliminar um elemento ?



Eliminacao logica do elemento alvo

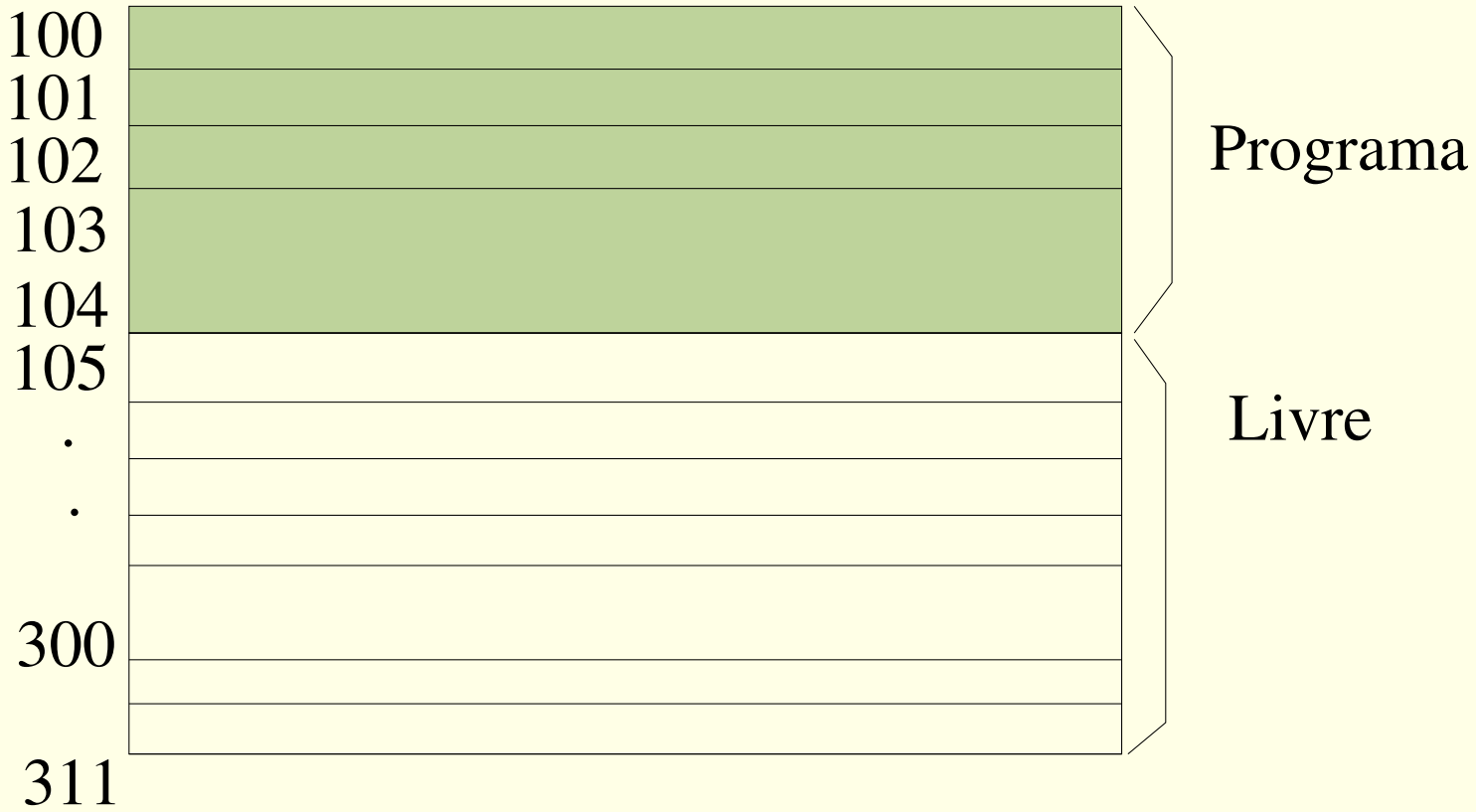
Alocação dinâmica de memória

Definição:

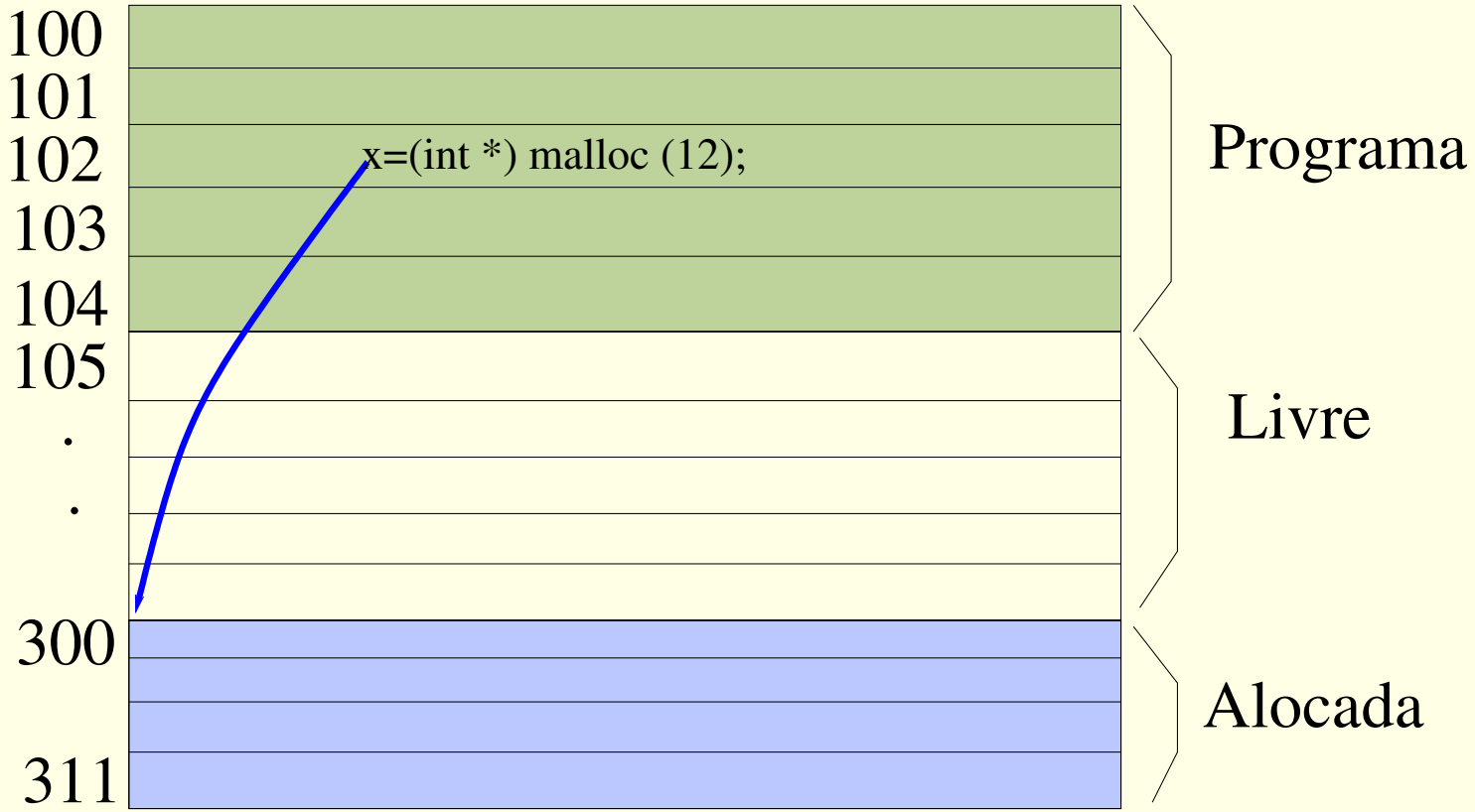
Chamamos de alocação dinâmica de memória quando o programador deve invocar uma função para obter memória do sistema operacional.
`malloc` - **m***emory* **alloc***ate*.

```
1 int *x;
2       x = (int *) malloc (12);
3       // Solicitou-se 12 bytes para o SO
4       // Se existe memoria disponivel,
5       // o SO retornar  o endereco de inicio da
            memoria
6       // cada inteiro ocupa 4 bytes, logo temos 3
            inteiros.
7       x[0]=234;
8       x[1]=123;
9       x[2]=450;
```

Memória internamente



Memória internamente



Alocação dinâmica de memória

No código do slide anterior, digamos que o SO tenha encontrado 12 bytes de memória livre iniciando no endereço 300. Ou seja, seriam os endereços 300,...,311. Nesta situação a função `malloc` retornará o endereço 300. Precisamos de uma variável ponteiro para armazenar este endereço. No programa em questão, temos `int *x` que é um ponteiro para inteiro.

A função `malloc` é feita para retornar um endereço qualquer, não somente para variáveis inteiras, mas para floats, double, ou variavéis que o programador tenha definido (`typedef` e `struct`). Para transformar o retorno da função `malloc()` para ser compatível com a variável `x` usamos o operador **cast** ou **operador de mudança de tipo**. `x = (int *) malloc (12);`

Alocação dinâmica de memória

Desalocando memória

Memória que foi alocada por `malloc()` e não é mais utilizada pode ser liberada pelo programador C para permitir que o SO faça algum uso desta. Internamente, a cada chamada `malloc()` o sistema contabiliza o requisito de memória e insere numa tabela de acesso restrito.

```
1 int *x, *z, *k;
2
3 x = (int *) malloc (12);
4 // vamos assumir que x
    =20
5 z = (int *) malloc (120)
    ;
6 // vamos assumir que y
    =328
```

End. Inicial	qtd bytes
20	12
328	120
4000	4000

Desalocando memória

Para desalocar a memória, utilizamos `free(endereco_inicial)`.

```
1 int *x, *z, *k;
2
3 x = (int *) malloc (12);
4 // vamos assumir que x=20
5 z = (int *) malloc (120);
6 // vamos assumir que y=328
7 k = (int *) malloc (234);
8 //vamos assumir que k=4000
9 free(z); //libera 120 bytes
```

Alocação dinâmica de memória

Descobrindo o tamanho de alguma variável

Podemos descobrir o tamanho de alguma variável utilizando a função `sizeof()`. Assim sendo:

```
1 typedef struct {
2     int CPF;       // 32 bits ou  4 bytes
3     char nome[20]; // 20 bytes
4 }Tipo_Pilha;       // total: 24 bytes
5  int x;
6  x=sizeof (int);     // 4 bytes
7  x=sizeof (int)*12; // 48 bytes
8  x=sizeof (Tipo_Pilha); //24 bytes
```

Note que `sizeof()` deve ser utilizando com os tipos de variáveis e não com o nome delas. Ou seja, posso obter o `sizeof(int)` mas

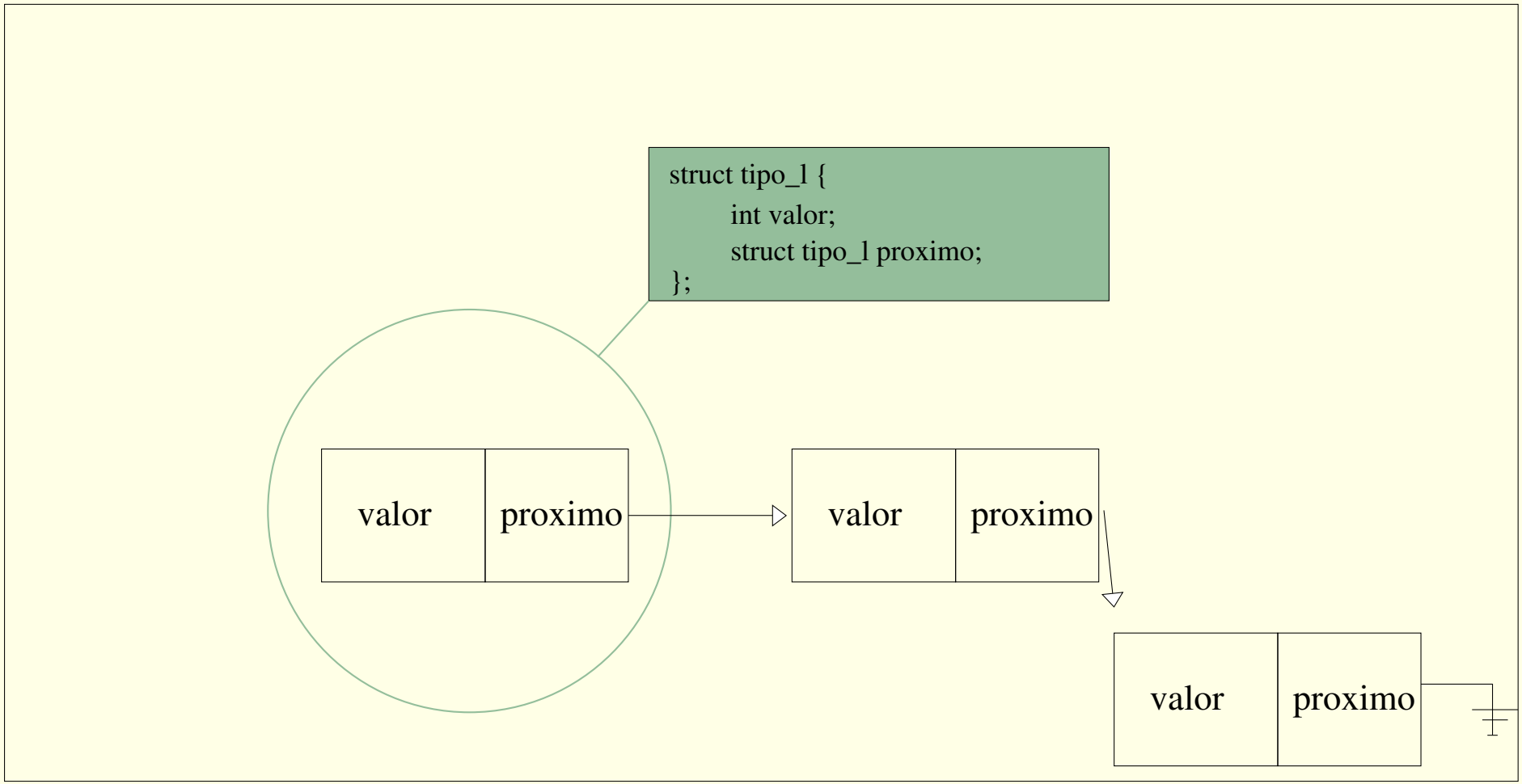
Alocação dinâmica de memória

Descobrindo o tamanho de alguma variável

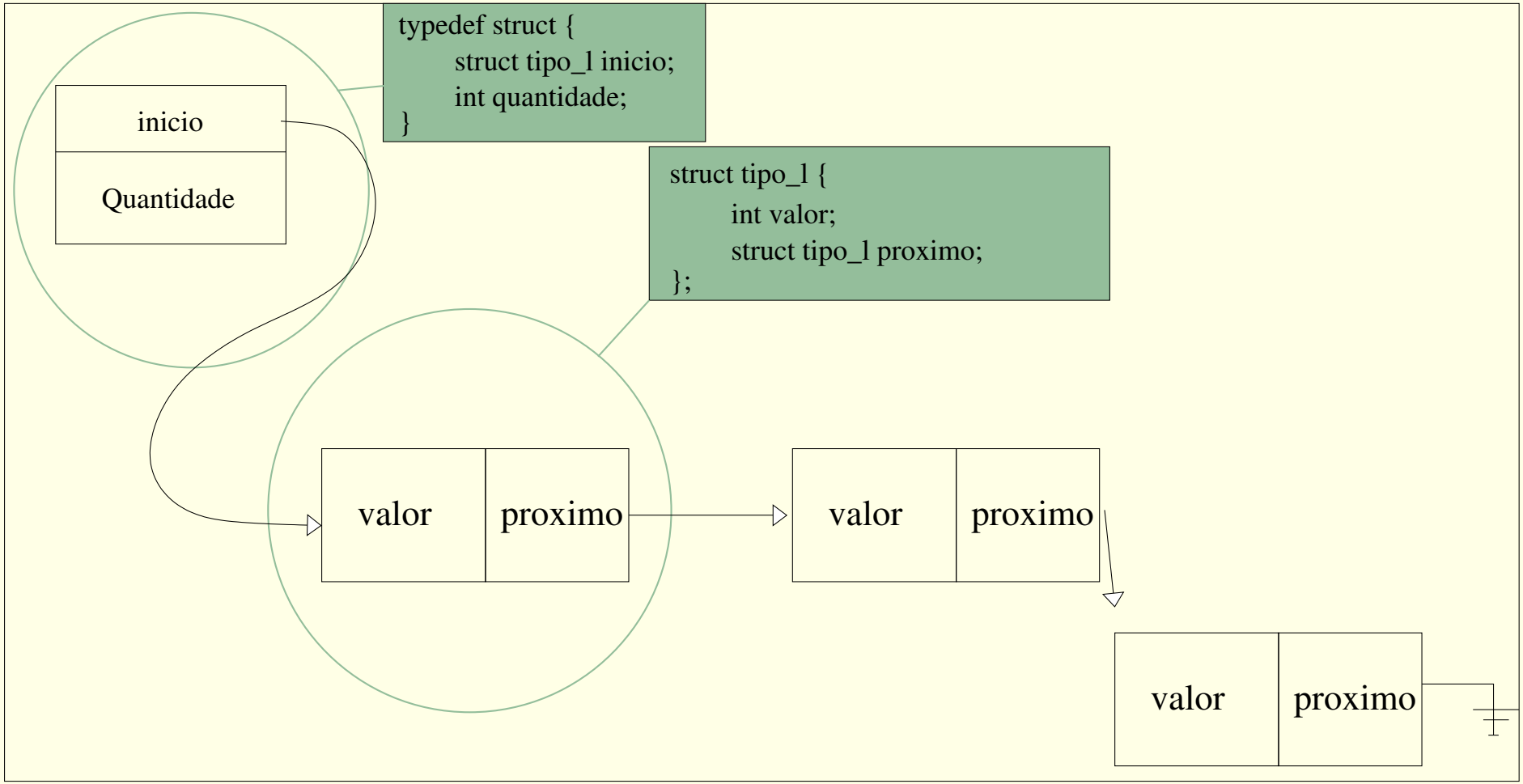
Veja o exemplo

```
1  typedef struct {
2      int CPF;          // 32 bits ou  4 bytes
3      int idade;        // 4 bytes
4  }Tipo_Pilha;          // total: 8 bytes
5
6  Tipo_Pilha * vetor;
7  vetor=(Tipo_Pilha *) malloc (100*sizeof(
       Tipo_Pilha));
8  if (vetor != NULL) { //Obteve a memoria
9
10     vetor[0].CPF  =123;
11     vetor[0].idade=23;
12
13     vetor[1].CPF=456;
```

Criando uma lista



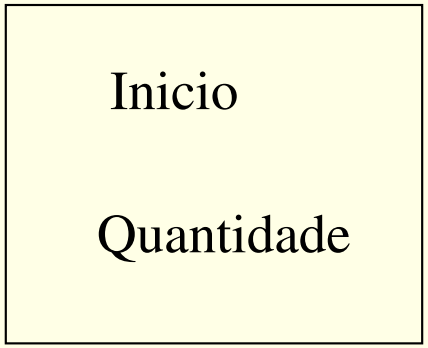
Criando uma lista



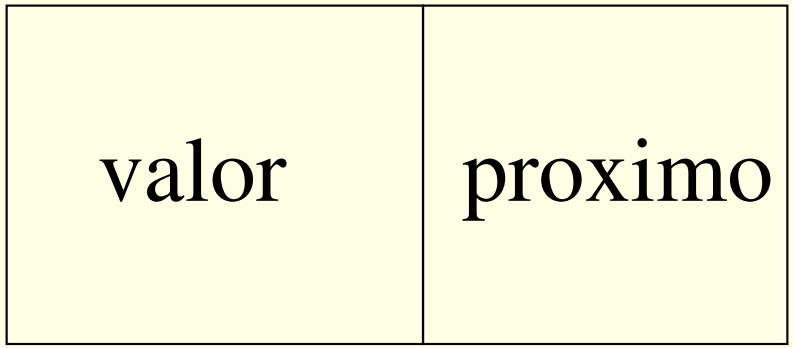
Criando uma lista

```
1 struct tipo_l {
2         int valor;
3         struct tipo_l *
             proximo;
4 };
5
6 typedef struct {
7         struct tipo_l *
             inicio;
8         int quantidade;
9 }Tipo_Lista;
```

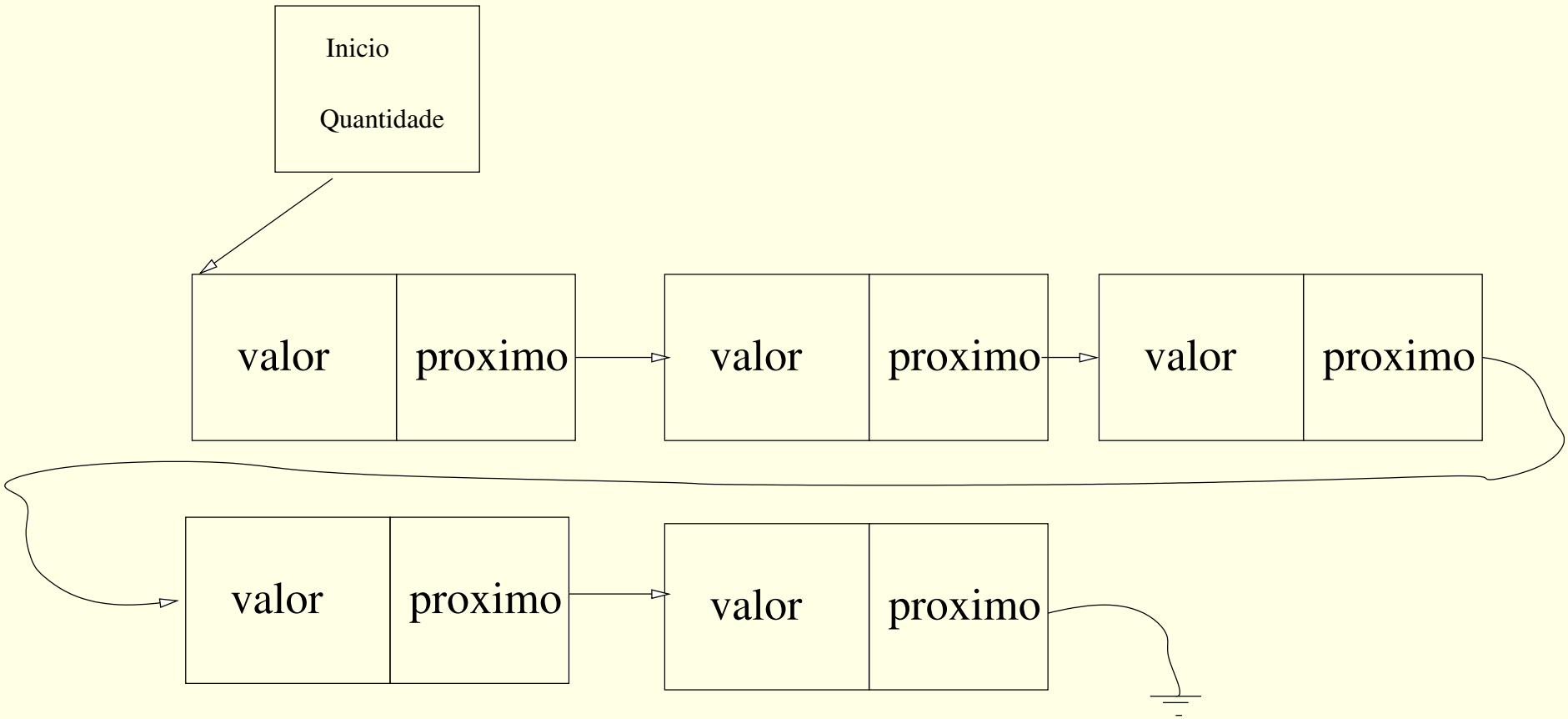
Tipo_Lista



struct tipo_elemento



Lista completa



Criando uma lista

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct tipo_elemento {
5      int valor;
6      struct tipo_elemento *proximo;
7  };
8
9  typedef struct {
10     struct tipo_elemento *inicio;
11     int quantidade;
12 }Tipo_Lista;
13
14 void inicializa (Tipo_Lista *a) {
15     a->quantidade=0;
16     a->inicio=NULL;
```

Criando uma lista

```
1  void insere (Tipo_Lista *a, int numero) {
2      struct tipo_elemento *tmp;
3      tmp = (struct tipo_elemento *) malloc (sizeof(
           struct tipo_elemento));
4      tmp->valor = numero;
5
6      if (a->quantidade == 0)
7          tmp->proximo=NULL;
8      else
9          tmp->proximo = a->inicio;
10
11     a->inicio=tmp;
12     a->quantidade++;
13 }
```

Criando uma lista

```
1  void mostra( Tipo_Lista m)
2  {
3      struct tipo_elemento *tmp;
4      tmp = m.inicio;
5
6      while (tmp != NULL )
7      {
8          printf("Valor=%d\n",tmp->valor);
9          tmp = tmp->proximo;
10     }
11 }
```

Criando uma lista

```
1  void elimina (Tipo_Lista *a, int alvo) {
2      struct tipo_elemento *tmp, *ant;
3      tmp=a->inicio;
4
5      while ( tmp != NULL) {
6          if (tmp->valor==alvo) {
7              // O elemento a ser eliminado     o
                  primeiro
8              if ( a->inicio==tmp )
9                  a->inicio = tmp->proximo;
10             else     ant->proximo   =   tmp->proximo;
11             free(tmp);
12             a->quantidade--;
13             return;
14         }
15         ant = tmp;
```