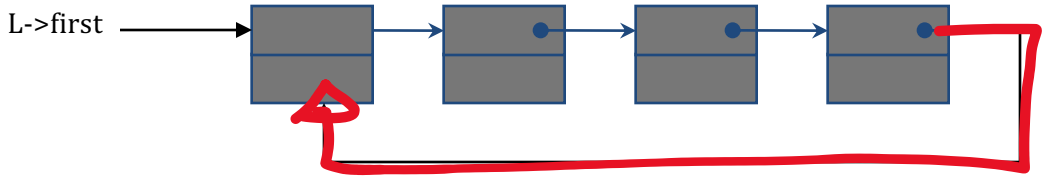
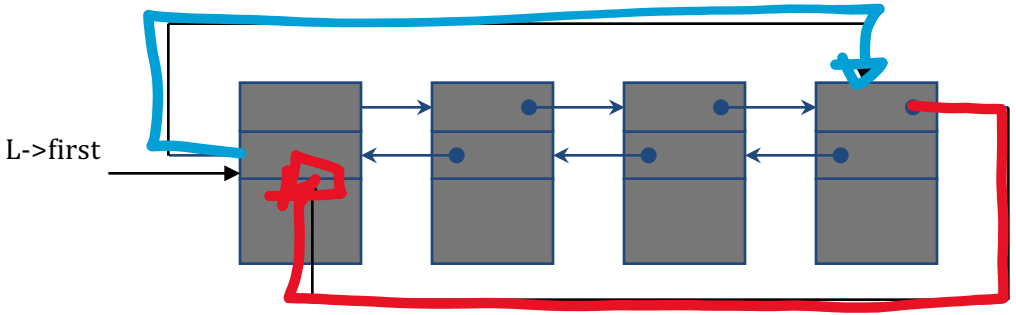
Listas Encadeadas Circulares

Estrutura de Dados
Prof. Anselmo C. de Paiva
Departamento de Informática – Núcleo de Computação Aplicada NCA-UFMA

Listas Encadeadas Circulares

- O último elemento recebe o endereço do primeiro (Lista simplesmente encadeada)
- O campo next do último elemento aponta para o primeiro e o campo previous do primeiro aponta para o último

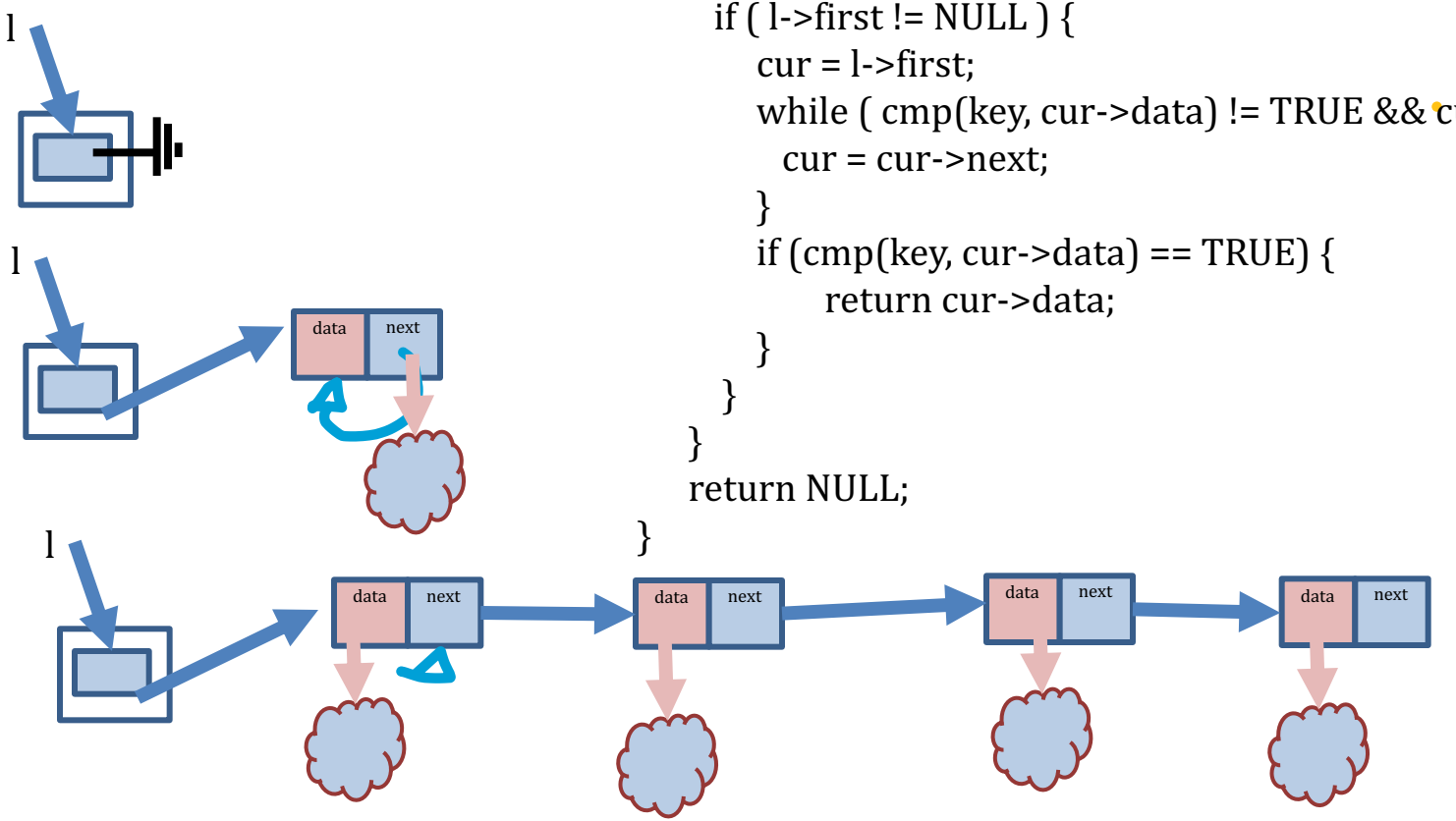




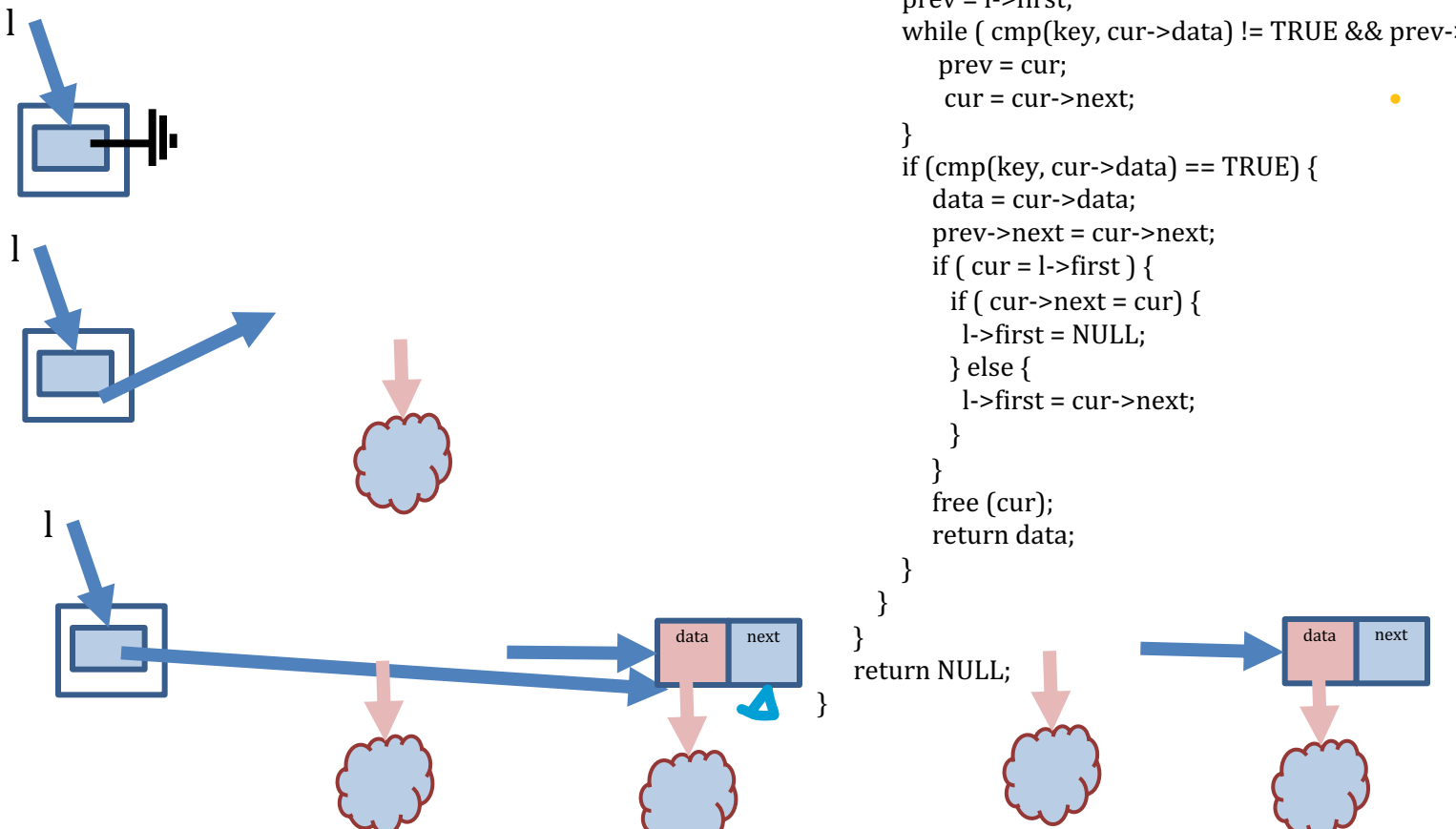
Listas Duplamente Encadeadas

- O que muda??
  - A forma de identificar o último elemento da lista
    - agora é o elemento cujo campo next é igual a l->first.
  - Não precisa entrar na lista somente pelo l->first, agora basta saber o endereço de um Nó e pode percorrer a lista.
  - Para qdo o campo next for igual ao Nó de entrada
- Exercícios
  - Escreva os algoritmos para as seguintes operações em listas circulares simplesmente encadeadas
    - Busca de elemento
    - Remoção de um elemento
    - Inserção após um elemento.

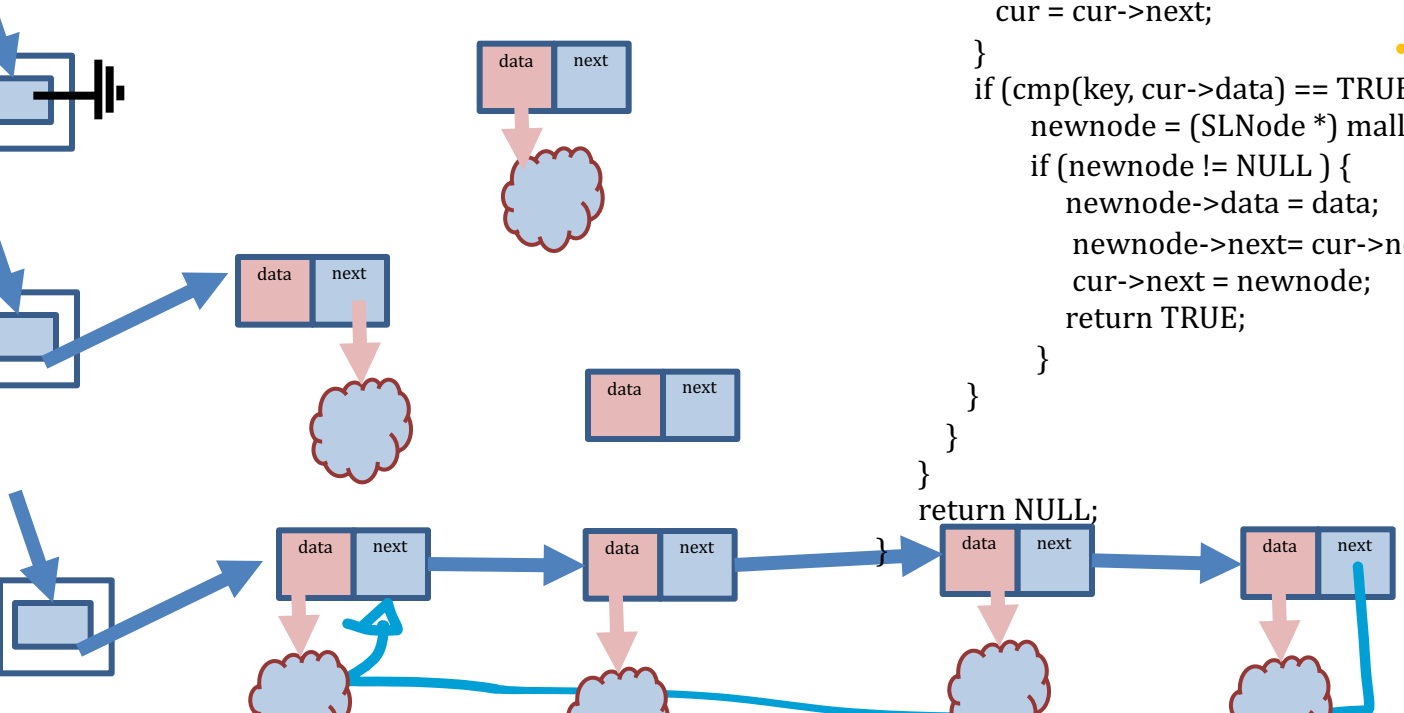
```
void *csllQuerySpec ( SLList *l, void *key, int (*cmp)(void *, void *))
{
    SLNode *cur;
    if ( l!= NULL ) {
       if ( l->first != NULL ) {
           cur = l->first;
           while ( cmp(key, cur->data) != TRUE && cur->next != l->first) {
             cur = cur->next;
           }
           if (cmp(key, cur->data) == TRUE) {
                 return cur->data;
           }
       }
    }
    return NULL;
}
```

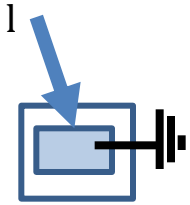


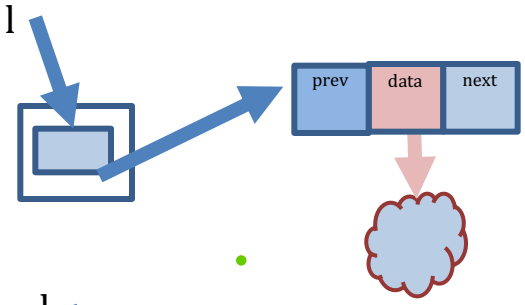
```
void *csllRemoveSpec ( SLList *l, void *key, int (*cmp)(void *, void *))
{
    if ( l!= NULL ) {
      if ( l->first != NULL ) {
          cur = l->first->next;
          prev = l->first;
          while ( cmp(key, cur->data) != TRUE && prev->next != l->first) {
               prev = cur;
                cur = cur->next;
          }
          if (cmp(key, cur->data) == TRUE) {
              data = cur->data;
              prev->next = cur->next;
              if ( cur = l->first ) {
                if ( cur->next = cur) {
                  l->first = NULL;
                } else {
                  l->first = cur->next;
                }
              }
              free (cur);
              return data;
          }
      }
    }
    return NULL;
}
```

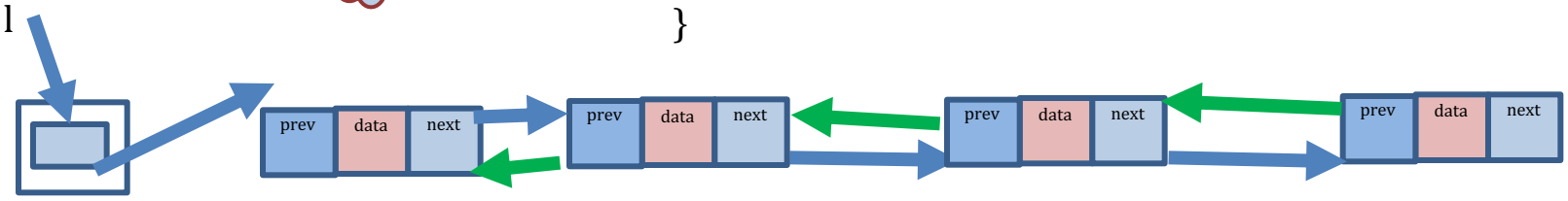


```
int csllInsertAfterSpec ( SLList *l, void *key,
                          int    (*cmp)(void *, void *), void *data)
{
    SLNode *cur;
    if ( l!= NULL ) {
       if ( l->first != NULL ) {
          cur = l->first;
          while ( cmp(key, cur->data) != TRUE &&
                     cur->next != l->first) {
             cur = cur->next;
          }
          if (cmp(key, cur->data) == TRUE) {
                newnode = (SLNode *) malloc (sizeof(SLNode));
                if (newnode != NULL ) {
                   newnode->data = data;
                    newnode->next= cur->next;
                    cur->next = newnode;
                   return TRUE;
                }
          }
       }
    }
    return NULL;
}
```










```
void *cdllQuerySpec ( DLList *l, void *key, int (*cmp)(void *, void *))
{
    DLNode *cur;
    if ( l != NULL ) {
       if ( l->first != NULL ) {
           cur = l->first;
           while ( cmp(key, cur->data) != TRUE && cur->next != l->first)
             cur = cur->next;
           }
           if (cmp(key, cur->data) == TRUE) {
                 return cur->data;
           }
       }
    }
    return NULL;
}
```

Cur->next = cur;

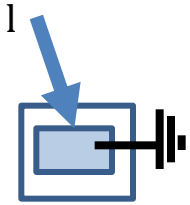
Cur->prev = cur;

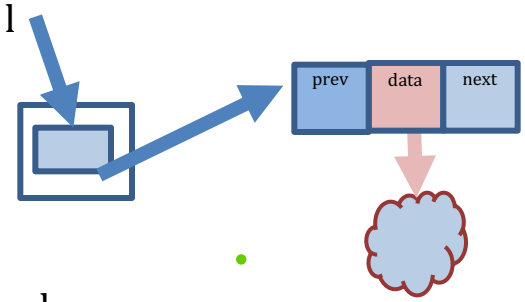
L->first->next = l->first

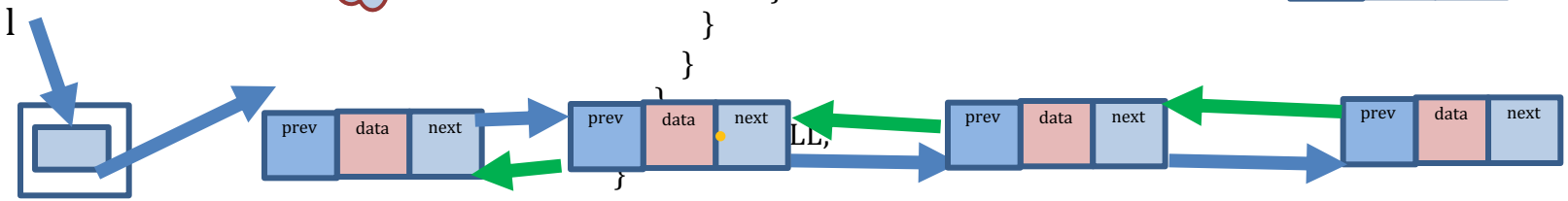
cur->next == cur->prev

```
void *cdllRemoveSpec ( DLList *l, void *key, int (*cmp)(void *, void *))
{
   DLNode *spec;
    if ( l!= NULL ) {
       if ( l->first != NULL ) {
           spec = l->first;
           while ( cmp(key, cur->data) != TRUE && spec->next != l->first) {
                 spec = spec>next;
           }
           if (cmp(key, spec->data) == TRUE) {
               data = spec ->data;
               prev = spec ->prev;
               next = spec ->next;
               if (spec ->next == cur) {
                     l->first = NULL;
               } else {
                    prev->next = next;
                   next->prev = prev;
               }
               free (spec);
               return data;
           }
        }
    }
     return NULL;
}
```


```
int cdllInsertAfterSpec ( DLList *l, void *key, int    (*cmp)(void *, void *), void *data)
{
   DLNode *spec;
   if ( l!= NULL ) {
      if ( l->first != NULL ) {
         spec = l->first;
         while ( cmp(key, spec>data) != TRUE &&  spec>next != l->first) {
           spec = spec->next;
         }
         if (cmp(key, cur->data) == TRUE) {
              newnode = (DLNode *) malloc (sizeof(DLNode));
              if (newnode != NULL ) {
                 newnode->data = data;
                  newnode->next= spec>next;
                   newnode->prev = spec;
                  spec->next->prev = newnode;
                  spec->next = newnode;
                   return TRUE;
               }
          }
       }
    }
}
```







Escreva um algoritmo que calcule o comprimento de uma lista L1:

```
int cdllNumElms( DLList *l)
{
    DLNode.  *cur;  int nelms=0;
    if ( l != NULL) {
      if ( l->first != NULL ) {
          cur = l->first;
          while ( cur-> next != l->first) {
              nelms++;
              cur = cur->next;
           }
           nelms ++;
           return nelms;
        }
        return 0;
    }
    return -1;
}
```

5. Escreva um algoritmo NumComuns (L1,L2) , que deve retornar um valor inteiro igual ao número de valores comuns às duas listas ordenadas L1 e L2;

```
int sllNumComuns ( SLList *l1, SLList *l2, int (*cmp) ( void *, void *))    // OPCAO COM O DO WHILE
{.
   SLLNode *cur1, *cur2;  int nelms=0;
   if( l1 != NULL && l2 != NULL ) {
      if ( l1->first != NULL &&  l2->first != NULL ) {
=Cur1 = l1->first
          do {
              cur2 = l2->first;
              do  {
                   if ( cmp ( cur1->data, cur2->data) ==  TRUE) {
                       nelms++;
                   }
                  cur2=cur2->next;
              } while (cur2->next != NULL )
              cur1 = cur1->next;
           } while (cur1->next != NULL);
          return nelm;
       }
       return 0;
    }
    return -1;
}
```

5. Escreva um algoritmo NumComuns (L1,L2) , que deve retornar um valor inteiro igual ao número de valores comuns às duas listas ordenadas L1 e L2;

```
int sllNumComuns ( SLList *l1, SLList *l2, int (*cmp) ( void *, void *))
{
   SLLNode *cur1, *cur2;  int nelms=0;
   if( l1 != NULL && l2 != NULL ) {
      if ( l1->first != NULL &&  l2->first != NULL ) {
            Cur1 = l1->first
           while (cur1 != NULL) {
              cur2 = l2->first;
              while (cur2  != NULL ) {
                   if ( cmp ( cur1->data, cur2->data) ==  TRUE) {
                       nelms++;
                   }
                  cur2=cur2->next;
              }
              cur1 = cur1->next;
           }
           return nelm;
       }
       return 0;
    }
    return -1;
}
```

```
int sllIntercala ( SLList *l1, SLList *l2)
{
   if ( l1 != NULL && l2 != NULL ) {
     if ( l1-> first != NULL && l2->first != NULL ) {
        cur1 = l1->first;
        cur2 = l2->first;
        next1 = cur1->next;
        next2 = cur2->nextl;
        While (cur1 != NULL &&  cur2 != NULL) {
             cur1->next = cur2;
             cur2->prev = cur1;
             if ( next1 != NULL){
               cur2->next = next1;
               next1-> prev = cur2;
             }
            cur1= next1;
            cur2 = next2;
            if (next1 != NULL ) {
                next1 = next1->next;
             }
             if ( next2 != NULL) {
                next2=next2->next;
              }
         }
         return TRUE;
     }
   } return  FALSE;
}
```

11. Escreva um algoritmo ÉInversa (L1, L2) que retorna 1 se a lista L1 tem os mesmos elementos de L2 na ordem inversa, -1 se L1 tem menos elementos que L2 e 0 se L1 tem mais elementos que L2

```
int sllEInversa ( SLList *l1, SLList *l2, int (*cmp) ( void *, void *))
{
```