SDA - Liste

Daniel Chiș - 2022, UPB, ACS, An I, Seria AC

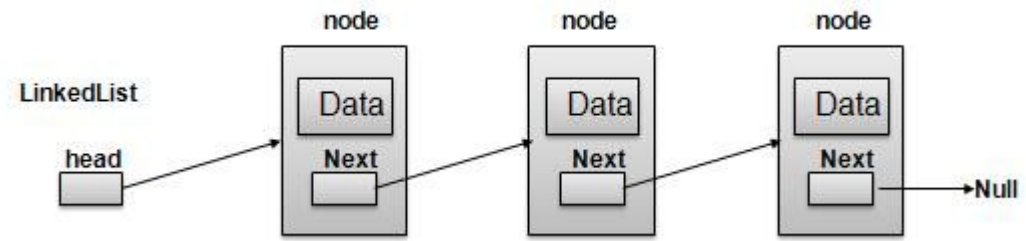
Liste Înlănțuite
(Linked Lists)

Liste Simplu Înlănțuite

Liste Simplu Înlănțuite

O listă înlănțuită reprezintă o secvență de lanțuri (links) care conțin elemente. Fiecare link conține o conexiune către alt link.

- Link - fiecare link conține date numite elemente
- Next - fiecare link conține un link către următorul link numit Next
- LinkedList - o listă înlănțuită conține un link de conexiune către primul link numit First



```c
struct node {
   int data;
   int key;
   struct node *next;
};

struct node *head = NULL;
struct node *current = NULL;

//display the list
void printList() {
   struct node *ptr = head;
   printf("\n[ ");

   //start from the beginning
   while(ptr != NULL) {
      printf("(%d,%d) ",ptr->key,ptr->data);
      ptr = ptr->next;
   }

   printf(" ]");
}
```

Definire + afișare listă

Insert

```
void insertFirst(int key, int data) {
   //create a link
   struct node *link = (struct node*) malloc(sizeof(struct node));

   link->key = key;
   link->data = data;

   //point it to old first node
   link->next = head;

   //point first to new first node
   head = link;
}
```

Delete

```
struct node* delete(int key) {

   //start from the first link
   struct node* current = head;
   struct node* previous = NULL;

   //if list is empty
   if(head == NULL) {
      return NULL;
   }

   //navigate through list
   while(current->key != key) {

      //if it is last node
      if(current->next == NULL) {
         return NULL;
      } else {
         //store reference to current link
         previous = current;
         //move to next link
         current = current->next;
      }
   }

   //found a match, update the link
   if(current == head) {
      //change first to point to next link
      head = head->next;
   } else {
      //bypass the current link
      previous->next = current->next;
   }

   return current;
}
```

Reverse

```
void reverse(struct node** head_ref) {
   struct node* prev   = NULL;
   struct node* current = *head_ref;
   struct node* next;

   while (current != NULL) {
      next  = current->next;
      current->next = prev;
      prev = current;
      current = next;
   }

   *head_ref = prev;
}
```

Search

```
struct node* find(int key) {

   //start from the first link
   struct node* current = head;

   //if list is empty
   if(head == NULL) {
      return NULL;
   }

   //navigate through list
   while(current->key != key) {

      //if it is last node
      if(current->next == NULL) {
         return NULL;
      } else {
         //go to next link
         current = current->next;
      }
   }

   //if data found, return the current Link
   return current;
}
```

Sort

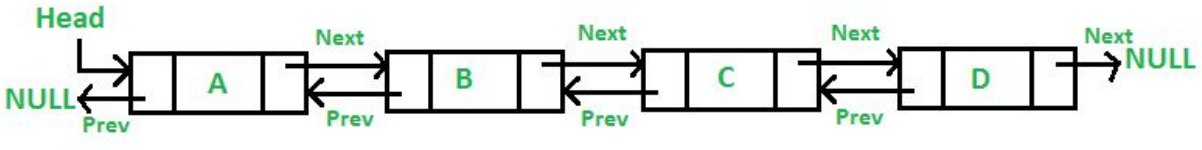
```
void sort() {

   int i, j, k, tempKey, tempData;
   struct node *current;
   struct node *next;

   int size = length();
   k = size ;

   for ( i = 0 ; i < size - 1 ; i++, k-- ) {
      current = head;
      next = head->next;

      for ( j = 1 ; j < k ; j++ ) {

         if ( current->data > next->data ) {
            tempData = current->data;
            current->data = next->data;
            next->data = tempData;

            tempKey = current->key;
            current->key = next->key;
            next->key = tempKey;
         }

         current = current->next;
         next = next->next;
      }
   }
}
```

Liste dublu înlănțuite

Liste Dublu Înlănțuite

O listă dublu înlănțuită este o variație a listei simplu înlănțuită în care navigația este posibilă în ambele sensuri.

- Link - fiecare link conține date numite elemente
- Next - fiecare link conține un link către următorul link numit Next
- Prev - fiecare link conține un link către elementul anterior numit Prev
- LinkedList - o listă înlănțuită conține un link de conexiune către primul link numit First dar și un link Last



```c
struct node {
   int data;
   int key;

   struct node *next;
   struct node *prev;
};

//this link always point to first Link
struct node *head = NULL;

//this link always point to last Link
struct node *last = NULL;

struct node *current = NULL;
```

Definire

Delete

```
struct node* delete(int key) {

   //start from the first link
   struct node* current = head;
   struct node* previous = NULL;

   //if list is empty
   if(head == NULL) {
      return NULL;
   }

   //navigate through list
   while(current->key != key) {
      //if it is last node

      if(current->next == NULL) {
         return NULL;
      } else {
         //store reference to current link
         previous = current;

         //move to next link
         current = current->next;
      }
   }

   //found a match, update the link
   if(current == head) {
      //change first to point to next link
      head = head->next;
   } else {
      //bypass the current link
      current->prev->next = current->next;
   }

   if(current == last) {
      //change last to point to prev link
      last = current->prev;
   } else {
      current->next->prev = current->prev;
   }

   return current;
}
```

Insert la primul nod

```
void insertFirst(int key, int data) {

   //create a link
   struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;

   if(isEmpty()) {
      //make it the last link
      last = link;
   } else {
      //update first prev link
      head->prev = link;
   }

   //point it to old first link
   link->next = head;

   //point first to new first link
   head = link;
}
```

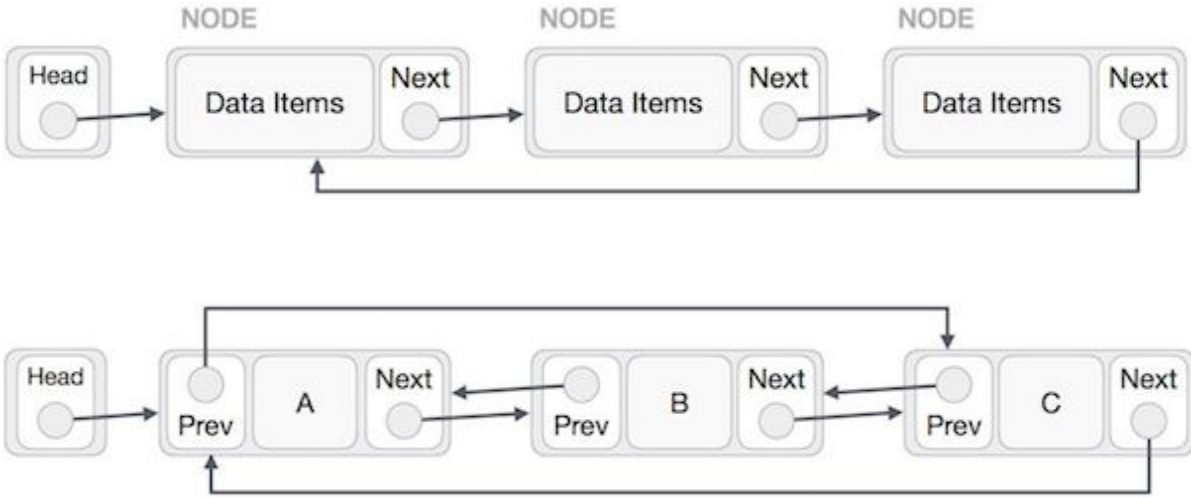
Insert la ultimul nod

```
void insertLast(int key, int data) {

   //create a link
   struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;

   if(isEmpty()) {
      //make it the last link
      last = link;
   } else {
      //make link a new last link
      last->next = link;

      //mark old last node as prev of new link
      link->prev = last;
   }

   //point last to new last node
   last = link;
}
```

Liste circulare

Liste Circulare

Listele circulare sunt o variație de liste înlănțuite în care primul element este legat de ultimul element iar ultimul de primul. Listele circulare pot fi atât simplu cât și dublu înlănțuite.



Insert

```
void insertFirst(int key, int data) {

   //create a link
   struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;

   if (isEmpty()) {
      head = link;
      head->next = head;
   } else {
      //point it to old first node
      link->next = head;

      //point first to new first node
      head = link;
   }
}
```

Delete

```
struct node * deleteFirst() {

   //save reference to first link
   struct node *tempLink = head;

   if(head->next == head) {
      head = NULL;
      return tempLink;
   }

   //mark next to first link as first
   head = head->next;

   //return the deleted link
   return tempLink;
}
```

Display

```
void printList() {

   struct node *ptr = head;
   printf("\n[ ");

   //start from the beginning
   if(head != NULL) {

      while(ptr->next != ptr) {
         printf("(%d,%d) ",ptr->key,ptr->data);
         ptr = ptr->next;
      }
   }

   printf(" ]");
}
```

Summary

De reținut

Listele pot fi simplu și dublu înlănțuite.

Ambele tipuri pot deveni circulare.

La fiecare listă avem link-uri care au date si chei către Next (și Prev în caz de dublu înlănțuită).

Exerciții

Exerciții

1. Realizați un program care să șteargă duplicatele dintr-o listă simplu înlănțuită nesortată.**3p**
2. Realizați un algoritm care să verifice că o listă simplu înlănțuită este palindrom. **3p**
3. Realizați o listă dublu înlănțuită în care să căutați un element și să îl ștergeți. **3p**

Exercitii FIIR

Creați o listă simplu înlănțuită cu 10 elemente la care să realizați următoarele operații.

- Creare listă
- Inserare elemente
- Ștergere elemente
- Căutare elemente
- Sortate listă
- Reverse listă