Liste

1 Liste înlănțuite

O **listă înlănțuită** este o structură liniară de date în care alocarea se realizează în mod dinamic. Spre deosebire de vectori, ordinea este determinată prin pointeri către următorul obiect.

Elementele listei se numesc **noduri**, fiecare nod fiind format din:

1. **câmp cheie** - un element al mulțimii reprezentate și
2. **pointer** către nodul următor.

```
struct Node {
  int Data;
  Node *Next;
};
```

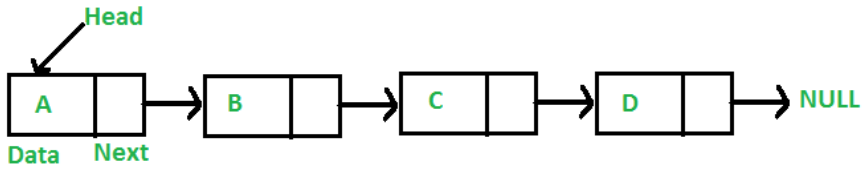

Figura 1: Reprezentarea unui nod dintr-o listă simplu înlănțuită[1]

În continuare vom trece prin operațiile de bază ce se pot aplica peste o listă înlănțuită oarecare:

1.1 Inserarea unui element

Presupune adăugarea unui element nou în cadrul listei (la început, în interior sau la final). Pentru cazul general vom considera că dorim să introducem un nou element între un nod ***p*** și predecesorul său ***oldp***.

[1]Linked List Node, https://www.geeksforgeeks.org/linked-list-set-1-introduction/

```
Node *q = (Node *) malloc(sizeeof(Node)) ;
// prelucrare q -> Data;
q -> Next = p;
if (oldp != NULL)
  oldp -> Next = q;
else
  Head = q;
```

Exercițiul 1.1. (1.5p) Să se creeze o listă ordonată crescător folosind operații de inserție (nodul Head va retine elementul minim).

1.2 Ștergerea unui element

Presupune ștergerea unui element din cadrul listei (de la început, din interior sau de la final). Pentru cazul general vom considera că dorim să ștergem un element ***p***, succesor al nodului ***oldp***.

```
Node *temp = p;
if (oldp != NULL)
  oldp -> Next = p -> Next;
else
  Head = p -> Next;
// prelucrare temp sau temp -> Data
free(temp);
```

Exercițiul 1.2. (1.5p) Implementați operația de ștergere a întregii liste.

1.3 Căutarea unui element

Constă în traversarea (eventual incompletă) a listei și compararea cu elementul căutat. Pentru cazul general vom considera că dorim să căutăm o valoare oarecare ***Val***.

```
Node *p = Head;
while (p != NULL  &&  Val != p -> Data)
  p = p -> Next;
if (p == NULL) // cautare fara succes;
else // gasit in p, Val == p -> Data;
```

Exercițiul 1.3. (1.5p) Implementați operația de ștergere a unui element dat la tastatură dintr-o listă înlănțuită oarecare (dacă el există).

Observație! Operațiile de bază sunt prezentate în limbajul C. Pentru utilizarea C++, alocarea și eliberarea de memorie se vor realiza astfel:

- alocarea de memorie

```
Node *p = new Node;
```

- eliberarea de memorie

```
delete p;
```

2 Liste dublu înlănțuite

Listele dublu înlănțuite sunt similare celor simple, nodurile având în plus însă un **pointer** către nodul anterior.

```
struct Node {
  int Data;
  Node *Next;
  Node *Prev;
};
```

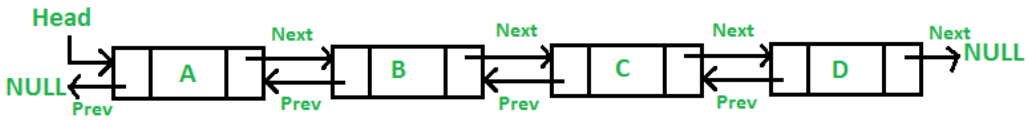

Figura 2: Reprezentarea unui nod dintr-o listă dublu înlănțuită[2]

2.1 Inserarea unui element

Pentru cazul general vom considera că dorim să introducem un nou element între un nod ***p*** și predecesorul său ***oldp***.

[2]Double Linked List Node, https://www.geeksforgeeks.org/doubly-linked-list/?ref=leftbar-rightbar

```
Node *q = (Node *) malloc(sizeof(Node)) ;
// prelucrare q -> Data;
q -> Next = p;
p -> Prev = q;
q -> Prev = oldp
if (oldp != NULL)
  oldp -> Next = q;
else
  Head = q;
```

2.2 Ștergerea unui element

Pentru cazul general vom considera că dorim să ștergem un element ***p***, succesor al nodului ***oldp***.

```
Node *temp = p;
if (oldp != NULL) {
  p -> Next -> Prev = oldp;
  oldp -> Next = p -> Next;
}
else {
  p -> Next -> Prev = NULL;
  Head = p -> Next;
}
// prelucrare temp sau temp -> Data
free(temp);
```

2.3 Căutarea unui element

Căutarea este absolut similară celei prezente în subsecțiunea 1.3. În schimb, există unele situații în care nodul final este și el cunoscut (să îl numim **Tail**) și atunci traversarea se poate realiza de data aceasta și de la sfârșit spre început. Pentru cazul general vom considera că dorim să căutăm o valoare oarecare ***Val***.

```
Node *p = End;
while (p != NULL  &&  Val != p -> Data)
  p = p -> Prev;
if (p == NULL) // cautare fara succes;
else // gasit in p, Val == p -> Data;
```

3 Liste circulare

Listele circulare sunt un tip de liste în care ultimul nod are drept succesor nodul inițial (Head). În cazul în care sunt dublu înlănțuite, nodul inițial are de asemenea ca predecesor ultimul nod. Operațiile implementate anterior sunt menținute. Singura diferență apare în

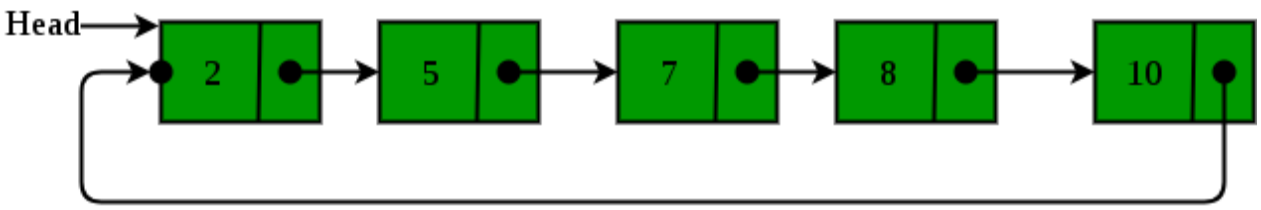

Figura 3: Exemplu listă circulară[3]

procedura de căutare sau la traversarea listei. În cazul listelor circulare parcurgerea listei se încheie în momentul în care se ajunge din nou în nodul considerat **Head**.

4 Aplicații

Exercițiul 2 - Reprezentarea numerelor mari. (2.5p) Se consideră două numere reprezentate într-o listă înlănțuită în care fiecare nod reprezintă o cifră. Cifrele sunt reținute în ordine inversă astfel încât cifra unităților să fie stocată la începutul listei . Să se scrie un program C/C++ care să calculeze suma acestor numere.
Exemplu:
Input: (1 -> 5 -> 3), (2 -> 6 -> 4)
Output: 813

[3]Circular Linked List, https://www.geeksforgeeks.org/circular-linked-list/?ref=lbp

Exercițiul 3 - Reprezentarea polinoamelor. (3p) Să se implementeze cu ajutorul unei liste liniare simplu înlănţuită alocată dinamic un polinom de grad n. Fiecare nod se va considera că reţine gradul fiecărui monom, precum şi coeficientul său. Structura poate fi definită astfel:

```
struct Node {
  int Grade, Coef;
  Node *Next;
};
```

Fiind date polinoamele P și Q de grad n, respectiv m, reprezentate ca mai sus, să se scrie câte un program care calculează și afișează:
3.1) Polinomul rezultat prin înmulțirea polinomului P cu scalarul a (număr real nenul);
3.2) $P(x_0)$ (evaluarea polinomului P într-un punct dat x_0);
3.3) Suma polinoamelor P și Q.

Exercițiul 4. (2.5p) Să se implementeze un program C/C++ care să împartă o listă circulară în 2 liste circulare separate (Vezi și The Tortoise and the Hare Algotithm).
Exemple:
Lista originală: 12 24 15 21 -> Prima listă: 12 24; A doua listă: 15 21.
Lista originală: 2 5 7 8 10 -> Prima listă: 2 5; A doua listă: 7 8 10.

Exercițiul 5. (2.5p) Să se implementeze un program care realizează ștergerea unui nod din interiorul unei liste simplu înlănțuite, oferindu-se acces numai la nodul respectiv.

5 Observații

1. Punctajul necesar pentru Laboratorul 3 se poate obține prin realizarea exercițiilor 1 - 3, exercițiile 4 și 5 fiind bonus.

2. Prezentarea problemei (problemelor) se poate realiza în timpul laboratoarelor L și $L+1$ unde L este considerat laboratorul curent (în cazul de față laboratoarele 3 și 4).

3. Prezentarea se poate desfășura atât fizic în timpul laboratorului, cât și online printr-o programare stabilită anterior cu laborantul (pentru cazuri speciale: simptome COVID, probleme personale etc.)

4. Transmiterea laboratorului se realizează pe adresa de e-mail ruxandra.balucea@unibuc.ro pana in ziua laboratorului. Denumirea exercițiilor va fi de tipul x_grupa_Nume_Prenume (exemplu: 2_141_Pop_Ion). **Rezolvarea exercițiilor 1_1, 1_2, 1_3 se va realiza în 3 funcții apelate dintr-un același fișier. Similar pentru exercițiul 3**. Toate aceste fișiere vor fi transmise într-o arhivă **grupa_Nume_Prenume.zip**.