Structuri de date

Liste: o listă este o structură abstractă de date care e formată dintr-o succesiune de elemente pentru care fiecare element conține o informație și o legătură către următorul element, cu excepția ultimului, care are o marcă specială.

Exemplu: jocul "Treasure Hunt".

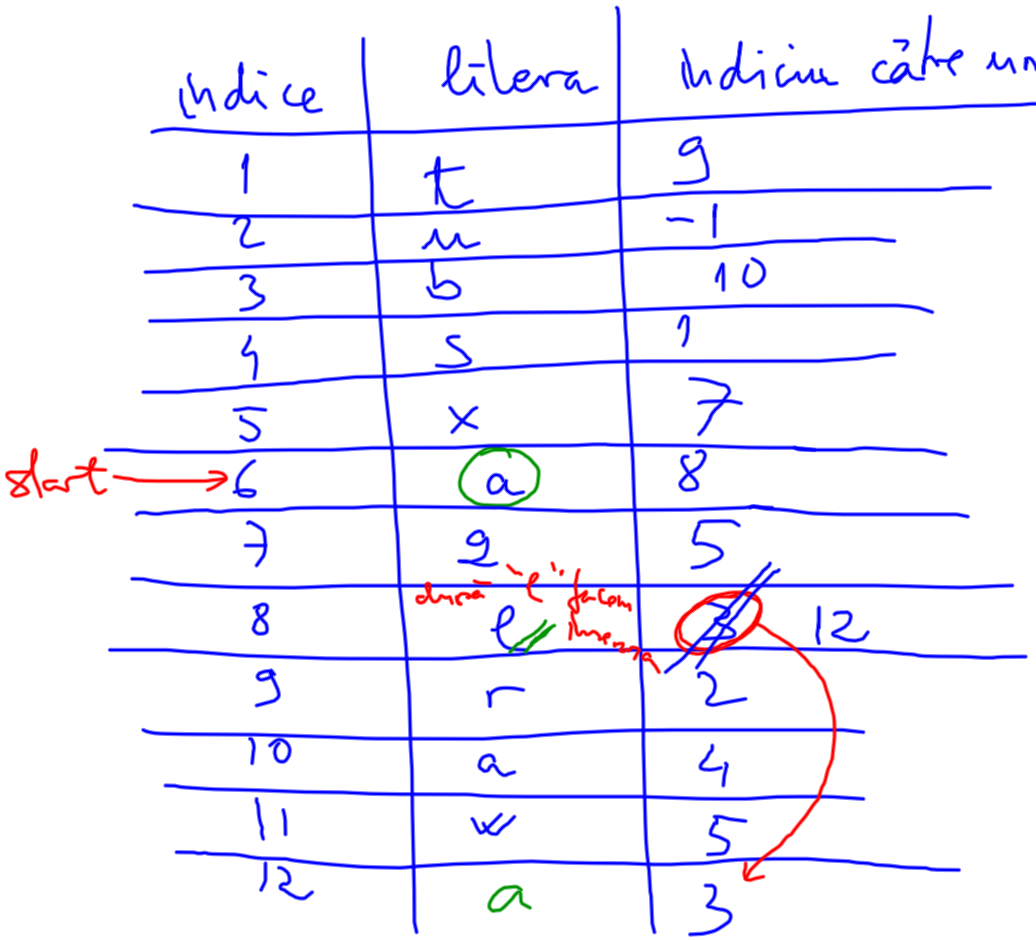
În fiecare clasă există o literă (pe care jucătorul trebuie s-o colecționeze) și un indiciu către următoarea clasă în care trebuie să meargă

În ultima clasă există un indiciu special că jocul s-a terminat (a dat de comoară).

Datele pot fi structurate astfel:

indice	litera	indiciu către următorul
1	t	9
2	u	-1
3	b	10
4	s	1
5	x	7
start → 6	a	8
7	g	5
8	e	3
9	r	2
10	a	4
11	w	5
12		

inserarea unui nou caracter se va face în foarte puțini pași. De ex., în loc de "abastru" să facem "alabastru"



Indice	Litera	Indicii către următorul
1	t	9
2	u	-1
3	b	10
4	s	1
5	x	7
start → 6	a	8
7	g	5
8	l	12
9	r	2
10	a	4
11	w	5
12	a	3

Modelarea practică se poate face fie cu 2 vectori (unul de char și altul de int) fie cu un vector de struct.

Memorarea unor date în această formă face ca operațiile de ștergere / inserare să fie mai rapide.

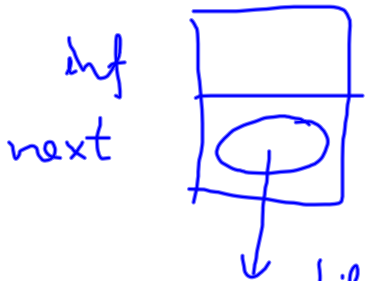
Dezavantajul major al listelor: accesul este secvențial: pentru a ajunge la un anumit indice trebuie parcurse toate elementele de la primul până la cel dorit.

Alocarea dinamică a listelor

De fapt modelul de listă prezentat este de "listă simplu înlănțuită".

Modelul dinamic folosește pointeri către struct-uri.

Mai precis, fiecare element:

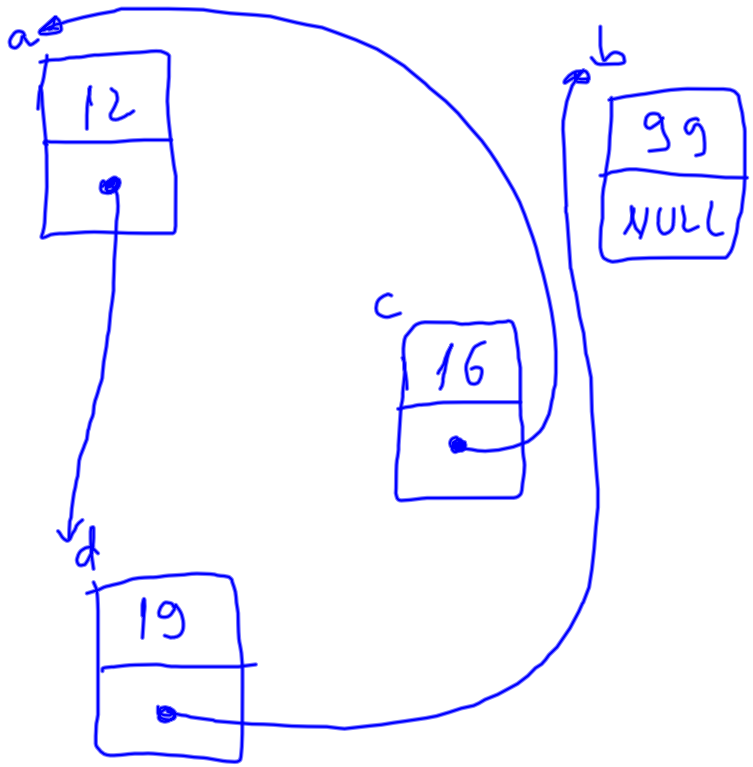


câmp capabil să reține o adresă de memorie a unui pointer de tip nod
⟺ va reprezenta de fapt adresa elem. următor.

```
struct nod
  { int inf; -> informatia
    nod *next;
  };
```

marca specială pt. ultimul element al listei va fi NULL.

Iată cum putem crea manual o listă cu 4 elemente:




```
nod *a, *b, *c, *d;

a = new nod;
b = new nod;
c = new nod;
d = new nod;
(*a).inf = 12;
(*a).next = d;
(*d).inf = 19; (*d).next = b;
(*b).inf = 99; (*b).next = NULL;
(*c).inf = 16; (*c).next = a;
cout << (*c).inf << " " << (*(*c).next).inf << " " << (*(*(*c).next).next).inf
     << " " << (*(*(*(*c).next).next).next).inf;
```

Thank God, exprenta

(*pointer).câmp se poate scrie

sub forma pointer -> câmp

Astfel, ultimul cout se poate rescrie:

```
cout<<c->inf<<" "<<c->next->inf<<" "<<c->next->next->inf
   <<" "<<c->next->next->next->inf;
```

Afișarea automată (de altfel în parcurgerea) unei liste simplu înlănțuite:

fie p adresa de început a listei

```
nod *q;
q=p;
while (q)
{ cout<<q->inf<<" ";
  q=q->next;
}
```

// for (q=p; q; q=q->next)
// cout<<q->inf<<" ";

Crearea automată a unei liste simple

A) De la stânga la dreapta.

Are trei faze:

A1) Crearea capului listei și menținerea unui pointer pe ultimul său element:


```
p = u = new nod;
p -> inf = ....
```

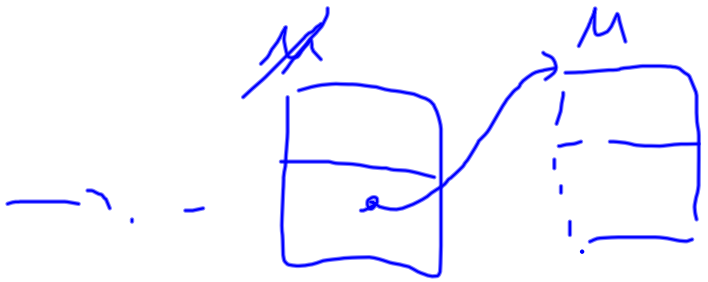
A2) Adăugarea unui elem. non DUPĂ u:

(se va face în mod repetat)


```
q = new nod;
q->inf = ....
q->next = u
u = q;
```

$\Longleftrightarrow$

```
u->next = new nod
u = u->next;
u->inf = ....
```



A3) Inchiderea listei:

```
u->next = NULL;
```

Obs: De regulă dintr-o listă este important să rămânem cu adresa de început = adresa primului element.

(! să încercați ca în programarea unei aplicații cu liste să evitați folosirea lui n = a nr. de elemente)