**Liste circulare**

1. **Liste circulare simplu inlantuite**

O lista liniara simplu inlantuita este **circulara** daca dupa ultimul element din lista urmeaza primul element al listei.

**Obs**. Lista nu mai contine valoarea NULL.

**Operatii**

1. **Creare –** se creeaza o lista liniara simplu inlantuita si dupa ultimul element primul din lista.

**struct nod**

**{ int inf; // informatia utila**

**nod \*urm; // adresa urmatorului element**

**} \*p, \*u;**

// p= adresa primului nod al listei

// u= adresa ultimului nod al listei

void creare (nod \*&p, nod \*&u)

{ int n,i;

nod \*c;

cin>>n;

p=u=NULL;

for (i=1;i<=n;i++)

{ c=new nod; // aloca spatiu la adresa c pentru noul nod

cin>>c->inf; // citim informatia utila de la adresa c

c->urm=NULL; // ii completez cu NULL adresa de legatura

if (p==NULL) p=u=c; // daca lista e vida primul si ultimul nod devin c

else { u->urm=c; // legam c dupa u – urmatorul de dupa u devine c

u=c; // actualizam adresa ultimului nod cu c

}

}

**u->urm=p;**

}

1. **Afisarea (parcugerea) elementelor din lista**

void afis (nod \*p)

{ nod \*c;

c=p;

while (**c->urm!=p**)

{ cout<<c->inf<<” ”; // afisam informatia utila de la adresa c

c=c->urm; // se avanseaza la urmatoarea adresa

}

**cout<<c->inf<<endl; // afisarea ultimului element**

}

1. **Cautarea unui element x in lista**

int caut (nod \*p, int x)

{nod \*c;

c=p;

while (c->urm!=p && c->inf!=x) c=c->urm;

if (c->urm==p) // iese la adresa ultimului element

if (c->inf==x) return 1;

else return 0;

else return 1;

}

// if (c->urm==p && c->inf!=x) return 0;

// else return 1;

1. **Liste circulare dublu inlantuite**

O lista liniara simplu inlantuita este **circulara** daca dupa ultimul element din lista urmeaza primul element al listei.

**Obs**. Lista nu mai contine valoarea NULL.

**Operatii**

1. **Creare –** se creeaza o lista liniara dublu inlantuita si se leaga ultimul element de primul.

**struct nod**

**{ int inf; // informatia utila**

**nod \*urm, \*pred; // adresa urmatorului element si al al elementului predecesor**

**} \*p, \*u;**

// p= adresa primului nod al listei

// u= adresa ultimului nod al listei

void creare (nod \*&p, nod \*&u)

{ int n,i;

nod \*c;

cin>>n;

p=u=NULL;

for (i=1;i<=n;i++)

{ c=new nod; // alocam spatiu la adresa c pentru noul nod

cin>>c->inf; // citim informatia utila de la adresa c

**c->pred**=c->urm=NULL; // ii completez cu NULL adresele de legatura

if (p==NULL) p=u=c; // daca lista e vida primul si ultimul nod devin c

else { u->urm=c; // legam c dupa u – urmatorul de dupa u devine c

**c->pred=u; // predecesorul lui c devine u**

u=c; // actualizam adresa ultimului nod cu c

}

}

**u->urm=p;**

**p->pred=u;**

}

**OBS.**  Afisarea si cautarea raman la fel.