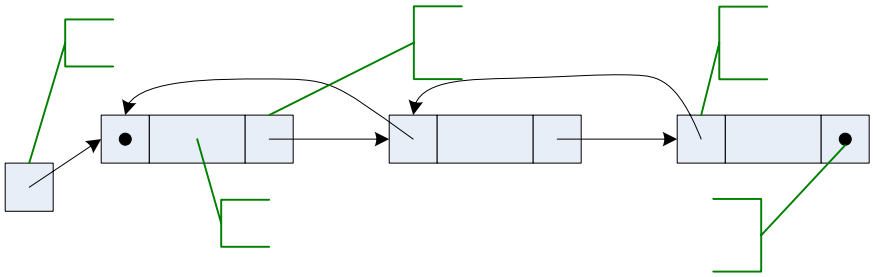
Liste Dublu Inlantuite

Listele dublu inlantuite sunt structuri de date dinamice omogene. Ele au aceleasi caracteristici de baza ca si listele simplu inlantuite. Diferenta fata de acestea consta in faptul ca, pentru fiecare nod, se retine si adresa elementului anterior, ceea ce permite traversarea listei in ambele directii.

Lista dublu inlantuita poate fi reprezentata grafic astfel:



1. Structura listei

Structura folosita este similara cu cea de la liste simple.

Pentru a asigura un grad mai mare de generalitate listei a fost creat un alias pentru datele utile (in cazul nostru un intreg):

```
// Datele asociate unui
// element dintr-o lista
typedef int Date;
```

Pointer catre inceputul listei

Pointe
urr

In cazul in care se doreste memorarea unui alt tip de date, trebuie schimbata doar declaratia aliasului ***Date***.

Pentru memorarea listei se foloseste o structura autoreferita. Acesta structura va avea forma:

```
// Structura unui element
// dintr-o lista dublu inlantuita
struct Element
{
      // datele efective memorate
      Date valoare;
      // legatura catre nodul urmator
      Element* urmator, * anterior;
};
```

Informatii utile

m

In cazul in care elemenul este ultimul din lista, pointerul ***urmator*** va avea valoarea NULL. Pentru primul element, pointerul ***anterior*** va avea valoarea NULL.

Declararea listei se face sub forma:

```
// declarare lista vida
Element* cap = NULL;
```

In cazul in care se doreste crearea unei liste circulare, pointerul ***urmator*** al ultimului element va referi primul element al listei, iar pointerul ***anterior*** al primului element va referi ultimul element.

2. Operatii cu liste duble

Principalele operatii cu liste sunt:

Parcurgere si afisare lista

Lista este parcursa pornind de la pointerul spre primul element si avansand folosind pointerii din structura pana la sfarsitul listei (pointer NULL).

```
// Parcurgere si afisare lista dubla
void Afisare(Element* cap)
{
	// cat timp mai avem elemente
	// in lista
	while (cap != NULL)
	{
		// afiseaza elementul curent
		cout << cap->valoare << endl;

		// avanseaza la elementul urmator
		cap = cap->urmator;
	}
}
```

Pentru parcurgerea inversa a listei se va porni cu un pointer catre ultimul element al listei, iar avansarea se va face folosind secventa `cap = cap->anterior; .`

Inserare element

Inserarea unui element se poate face la inceputul sau la sfarsitul listei.

a) Inserare la inceput
Acesta este cazul cel mai simplu: trebuie doar alocat elementul, legat de primul element din lista si repozitionarea capului listei:

```
// Inserare element la inceputul unei
// liste dublu inlantuite
void InserareInceput(Element* &cap, Date val)
{
	// Alocare nod si initializare valoare
	Element *elem = new Element;
	elem->valoare = val;
	elem->anterior = NULL;

	// legare nod in lista
```

```
	elem->urmator = cap;
	if (cap != NULL)
		cap->anterior = elem;

	// mutarea capului listei
	cap = elem;
}
```

b) Inserare la sfarsitul listei

In acest caz trebuie intai parcursa lista si dupa aceea adaugat elementul si legat de restul listei. De asemenea, trebuie avut in vedere cazul in care lista este vida.

```
// Inserare element la sfarsitul unei
// liste dublu inlantuite
void InserareSfarsit(Element* &cap, Date val)
{
	// Alocare si initializare nod
	Element *elem = new Element;
	elem->valoare = val;
	elem->urmator = NULL;
	elem->anterior = NULL;

	// daca avem lista vida
	if (cap == NULL)
		// doar modificam capul listei
		cap = elem;
	else
	{
		// parcurgem lista pana la ultimul nod
		Element *nod = cap;
		while (nod->urmator != NULL)
			nod = nod->urmator;

		// adaugam elementul nou in lista
		nod->urmator = elem;
		elem->anterior = nod;
	}
}
```

c) inserare dupa un element dat

```
void InserareInterior(Element* &cap, Element* p, Date val)
{
	// Alocare si initializare nod
	Element *elem = new Element;
	elem->valoare = val;
	elem->urmator = NULL;
	elem->anterior = NULL;

    // lista vida
	if (cap == NULL)
	{
		cap = elem;
		return;
```

```
	}

	// inserare la inceputul listei
	if (cap == p)
	{
		elem->urmator = cap;
		cap->anterior = elem;
		cap = elem;
		return;
	}

	// inserare in interior
	elem->urmator = p->urmator;
	elem->anterior = p;
	p->urmator->anterior = elem;
	p->urmator = elem;

}
```

Cautare element

Cautarea unui element dintr-o lista presupune parcurgerea listei pentru identificarea nodului in functie de un criteriu. Cele mai uzuale criterii sunt cele legate de pozitia in cadrul listei si de informatiile utile continute de nod. Rezultatul operatiei este adresa primului element gasit sau NULL.

a) Cautarea dupa pozitie

Se avanseaza pointerul cu numarul de pozitii specificat:

```
// Cautare element dupa pozitie
Element* CautarePozitie(Element* cap, int pozitie)
{
	int i = 0;  // pozitia curenta

	// parcurge lista pana la
	// pozitia ceruta sau pana la
	// sfarsitul listei
	while (cap != NULL && i < pozitie)
	{
		cap = cap->urmator;
		i++;
	}

	// daca lista contine elementul
	if (i == pozitie)
		return cap;
	else
		return NULL;
}
```

b) Cautarea dupa valoare

Se parcurge lista pana la epuizarea acesteia sau identificarea elementului:

```
// Cautare element dupa valoare
Element* CautareValoare(Element* cap, Date val)
{
	// parcurge lista pana la gasirea
	// elementului sau epuizarea listei
	while (cap != NULL && cap->valoare != val)
		cap = cap->urmator;

	return cap;
}
```

Stergere element

a) Stergerea unui element din interiorul listei (diferit de capul listei)

```
// sterge un element din interiorul listei
// primind ca parametru adresa elementului
void StergereElementInterior(Element* elem)
{
	// scurcircuitam elementul
	elem->anterior->urmator = elem->urmator;
	elem->urmator->anterior = elem->anterior;

	// si il stergem
	delete elem;
}
```

b) Stergerea unui element de pe o anumita pozitie

Daca elementul este primul din lista, atunci se modifica capul listei, altfel se cauta elementul si se sterge folosind functia definita anterior:

```
void StergerePozitie(Element* &cap, int pozitie)
{
	// daca lista e vida nu facem nimic
	if (cap == NULL)
		return;

	// daca este primul element, atunci
	// il stergem si mutam capul
	if (pozitie == 0)
	{
		Element* deSters = cap;
		cap = cap->urmator;
		cap->anterior = NULL;
		delete deSters;
		return;
	}

	// daca este in interor, atunci folosim
	// functia de stergere
	Element* elem = CautarePozitie(cap, pozitie);
```

```
	StergereElementInterior(elem);
}
```

c) stergerea dupa o valoare

Se cauta elementul si se foloseste functia de stergere element:

```
void StergereValoare(Element* &cap, Date val)
{
	// daca lista e vida nu facem nimic
	if (cap == NULL)
		return;

	// daca este primul element, atunci
	// il stergem si mutam capul
	if (cap->valoare == val)
	{
		Element* deSters = cap;
		cap = cap->urmator;
		cap->anterior = NULL;

		delete deSters;
		return;
	}

	// cautam elementul
	Element* elem = CautareValoare(cap, val);

	// daca a fost gasit, atunci il stergem
	if (elem->urmator != NULL)
		StergereElementInterior(elem);
}
```

3. Probleme

1. Definiți conceptul de listă dublu circulară și enumerați proprietățile.
2. Scrieți funcția pentru conversia unei liste duble în listă simplă.
3. Sa se scrie functia pentru adunarea a doua matrice rare memorate ca liste duble.
4. Fiind dată o listă dublă rezultată din adunarea a două matrice rare scrieți funcția pentru normalizare.
5. Sa se scrie o functie recursive care afiseaza elementele unei liste duble in ambele sensuri in functie de optiune.
6. Scrieti functia nerecursiva care efectueaza copierea unei liste, cu eliminarea elementelor ce au ca informatie utila o valoare mai mica decat un nivel specificat de parametru.
7. Scrieti si apelati functia care concateneaza o lista dubla cu o copie a sa.
8. Scrieți funcția pentru numărarea elementelor din n liste duble, în mod separat.
9. Sa se scrie functia pentru concatenarea a n liste dublu inlantuite.