DEMONSTRAREA MONOIDITĂȚII LISTELOR CU OPERAȚIA DE CONCATENARE

Lucia BITCOVSCHI

Catedra Tehnologii de Programare

The purpose of the article is to investigate the lists in terms of algebraic structures. There will be considered the most important operations such as concatenation of lists, inserting an item in the list. The elements of lists are considered as abstract data types, which contributed to achievement lists through template classes.

Introducere

În cele ce urmează vom încerca să studiem mulțimea listelor cu diferite operații din punctul de vedere al structurilor algebrice. Vor fi examinate cele mai importante operații asupra listelor, cum sunt: inserarea unui element în listă, concatenarea a două liste.

Vom cerceta pentru care din aceste operații o mulțime de liste formează *grup*.

Elementele listelor reprezintă un tip abstract de date, pe care îl vom nota prin *T*. Acest tip va servi drept parametru la realizarea listelor prin clase template.

Definiția 1. *Structurile de date reprezintă modalități în care datele sunt alocate în memoria calculatorului sau sunt păstrate pe discul magnetic.*

Structurile de date sunt utilizate în diferite circumstanțe, ca de exemplu:
- *memorarea unor date din realitate;*
- *instrumente ale programatorilor;*
- *modelarea unor situații din lumea reală.*

Cel mai des utilizate structuri cu acelaşi tip de date sunt tablourile şi tipuri diferite de date (listele, stivele, cozile, arborii, tabelele de dispersie şi grafele).

1. Lista simplu înlănțuită cu operațiile de inserare şi concatenare

Definiția 2. *O listă liniară este o structură de date omogenă, secvențială formată din elemente ale listei. Un element (numit uneori şi nod) din listă conține o informație specifică şi eventual o informație de legătură.*

Poziția elementelor din listă defineşte ordinea elementelor (primul element, al doilea etc.). Operațiile pentru liste sunt:
- parcurgerea şi afişarea listei;
- adăugarea de noi elemente la sfârşitul listei;
- inserarea de noi elemente în orice loc din listă;
- ştergerea de elemente din orice poziție a listei;
- modificarea unui element dintr-o poziție dată.

Printre operațiile care schimbă structura listei vom considera şi inițializarea unei liste ca o listă vidă. Alte operații sunt operațiile de caracterizare. Ele nu modifică structura listelor, dar furnizează informații despre ele. Dintre operațiile de caracterizare vom menționa următoarele:
- localizarea elementului din listă care satisface un anumit criteriu;
- determinarea lungimii listei.

Pot fi luate în considerare şi operații mai complexe, ca de exemplu:
- separarea unei liste în două sau mai multe subliste în funcție de îndeplinirea unor condiții;
- selecția elementelor dintr-o listă, care îndeplinesc unul sau mai multe criterii, într-o listă nouă;
- crearea unei liste ordonate după valorile crescătoare sau descrescătoare ale unei chei;
- combinarea a două sau mai multor liste prin concatenare (alipire) sau interclasare.

Spațiul de memorie ocupat de listă poate fi alocat în două moduri:

a) prin alocare secvențială *sau* b) prin alocare înlănțuită.

2. Liste simplu înlănțuite

Listele simplu înlănțuite sunt structuri de date dinamice omogene. Spre deosebire de tabele, listele nu sunt alocate ca blocuri omogene de memorie, ci ca elemente separate de memorie. Fiecare nod al listei conține, în afară de informația utilă, adresa următorului element. Această organizare permite numai acces secvențial la elementele listei. Pentru accesarea listei trebuie cunoscută adresa primului element (numit *capul listei*); elementele următoare sunt accesate parcurgând lista.

Definiția 3. *O **listă liniară*** (numită şi *listă înlănțuită*, din engleză – *„Linked List”) este o colecție de* $n \geq 0$ *elemente* $x_1,...,x_n$ *toate de acelaşi tip, numite **noduri**, între care există o relație de ordine determinată de poziția lor relativă.*

Lista simplu înlănțuită poate fi reprezentată grafic cum este arătat în Figura 1. Lista simplu înlănțuită este deci o mulțime eşalonată de elemente de acelaşi tip având un număr arbitrar de elemente. Numărul n al nodurilor se numeşte *lungimea listei*.

Dacă $n = 0$, lista este vidă.

Dacă $n \geq 1$, x_1 este primul nod, iar x_n este ultimul nod. Pentru $1 < k < n$, x_k este precedat de x_{k-1} şi urmat de x_{k+1} [1].

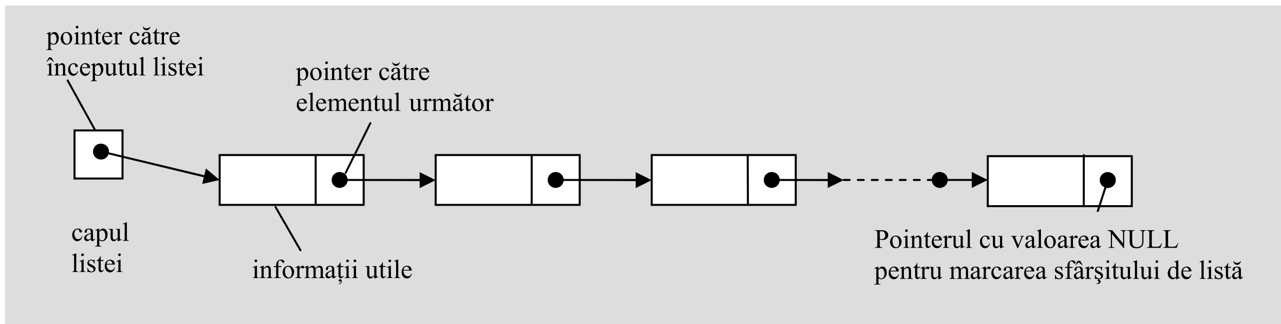

Fig.1. Listă simplu înlănțuită.

Definiția 4. *Listele înlănțuite cu un singur câmp de legătură se numesc **liste simplu inlănțuite** (legătura indică următorul element din listă). Fiecare nod conține un pointer ce indică adresa nodului următor din listă.*

Pentru accesarea listei trebuie cunoscută adresa primului element şi se efectuează prin capul listei, elementele următoare fiind accesate parcurgând lista. Ultimul element poate conține ca adresă de legătură constanta zero (pentru care în limbajul C++ este denumirea simbolică *NULL*), indicând astfel că ultimul nod nu are nici un succesor. În cazul în care elementul este ultimul din listă, *pointerul* următor va avea valoarea NULL.

Descriem fiecare operație folosind funcții şi clase *template* (şabloane).

3. Structura listei

Structura unui nod al listei va fi definită cu ajutorul *struct*, deoarece trebuie să rețină, pe lângă informația propriu-zisă de tipul *T*, şi adresa elementului următor din lista:

```
struct Nod
{
   T info; // T este tipul format template
   Nod *urm; // urm este pointer către următorul element al listei
}
```

Clasa `lista` va conține:

```
template<class T> class lista
{
```

```
   protected:
      struct Nod
      {
         T info;
         Nod*urm;// pointer spre nodul urmator
      };

      Nod *prim;

   public:
      lista();
      lista(int n);
      void adauga_incep(T p);
      void adauga_sfar(T p);
      void adauga_dupa_un_nod(T p);
      void afiseaza_lista();
      int concatliste(lista * cap1, lista * cap2);
}
```

- **Date-membre:**
  - **prim** – reprezentând adresa primului element al listei.
- **Funcţii-membre:**
  - **lista();** – constructor de iniţializare a listei.
  - **lista(int n);** – constructor care încarcă n elemente în listă.
  - **void adauga_incep (T p);** – adaugă un element de tipul *formal T* la începutul listei.
  - **void adauga_dupa_un_nod (T p);** – adaugă un element de tipul *formal T* după un nod specificat.
  - **void adauga_sfar (T p);** – adaugă un element de tipul *formal T* la sfârşitul listei.
  - **void afiseaza_lista();** – afişează elementele din listă.
  - **int concatliste (lista * cap1, lista * cap2);** - concatenarea a două liste.

4. Implementarea listei prin clasă generică

Folosind cuvântul-cheie *template*, se pot crea şabloane pentru funcţii şi clase numite *funcţii generice*, respectiv *clase generice*. Într-o funcţie sau clasă generică tipul de date asupra căruia operează acestea este specificat ca un parametru.

Funcţiile-membre generice definesc un set de operaţii care vor fi aplicate unor tipuri de date variate; tipul de date asupra căruia va opera va fi transmis ca parametru:

```
template <class Tip> tip_rezultat nume_functie (lista_parametri)
{
   //corp functie
}
```

O clasă-şablon (*template*) permite folosirea de atribute al căror tip nu este specificat imediat, ci este lăsat generic. La evidenţierea de obiecte din astfel de clase se specifică şi tipul atributelor generice. Prin acest mecanism se pot dezvolta clase de clase care diferă între ele prin tipul unor atribute. Din acest motiv, clasele respective mai poartă şi numele de clase generice [2].

La crearea clasei, care defineşte toate funcţiile-membre ca funcţii generice, urmează ca tipul de date utilizat efectiv să fie specificat în momentul creării unui obiect-instanţă al clasei generice.

Numele clasei *template* este însoţit de tipul care parametrizează clasa într-o construcţie de forma: *nume_clasa <nume_tip_parametru>*

Această regulă nu se aplică doar în cazul apelului constructorului (care are acelaşi nume cu cel al clasei asociate). În interiorul definiţiei unei clase *template*, ca şi în corpul unei funcţii-membru, se recomandă să se folosească tipul parametru al clasei.

O modalitate de a dezvolta o clasă-listă, care să conţină elemente de un tip de date oarecare, este definirea tipului unui element al listei chiar în interiorul clasei. Fie acest tip *T*.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
template<class T> class lista
{
   protected:
      struct Nod
      {
         T info;
         Nod*urm;// pointer spre nodul urmator
      };


      Nod *prim;

   public:
      lista();
      lista(int n);
      void adauga_incep(T p);
      void adauga_sfar(T p);
      void adauga_dupa_un_nod(T p);
      void afiseaza_lista();
      int concatliste(lista * cap1, lista * cap2);
};

// constructor fara parametri. Se initializeaza prim cu
// valoarea NULL.
template <class T> lista <T>::lista()
{
   prim=NULL;
}
// constructor cu parametru n. Se creează lista cu n elemente.
template <class T> lista <T>::lista(int n)
{
   int i;
   T inf;
   Nod *p;
   Cout << ”construim lista care are ”<< n << ”elemente”<< endl;
   cout<<”Dati informatia pt. primul element”; cin>>inf;
   prim=new Nod;
   prim->info=inf;
   prim->urm=NULL;
   p=prim;
   for(i=2; i<=n; i++)
   {
      cout<<”dati inf=”; cin>>inf;
      p->urm=new Nod;
      p=p->urm;
      p->info=inf;
      p->urm=NULL;
   }
}
```

5. Parcurgerea şi afişarea listei

Lista este parcursă pornind de la pointerul spre primul element şi avansând folosind pointerii din structură până la sfârşitul listei (pointer NULL).

```
template <class T> void lista <T>:: afiseaza_lista()
{
   Nod*p;
   p=prim;
   while(p!=NULL)
   {
      cout<<p->info<<"";
      p=p->urm;
   }
}
```

6. Inserarea la începutul listei

Acesta este cazul cel mai simplu: trebuie doar alocat elementul, legat de primul element din listă, şi repoziţionat capul listei (*a se vedea* Fig.2).

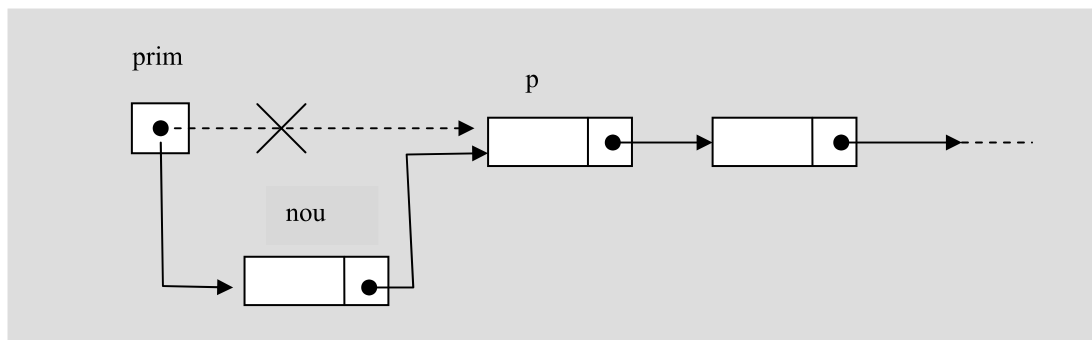

Fig.2. Adaugarea unui element la începutul listei.

```
template <class T> void lista<T>::adauga_incep(T nou)
{
   Nod*nou;
   nou=new Nod;
   if(nou==NULL)
   {
      Cout << "Eroare alocare spatiu la inserare";
      return;
   }
   if(prim==NULL)
      nou->urm=prim;
   prim=nou;
   cout << "Inserare la inceputul listei cu succes!\n";
}
```

Dacă elementul adăugat la începutul listei este primul element al listei (caz în care pointerul prim conţine valoarea NULL), vom actualiza şi valoarea pointerului *ultim*, elementul adăugat fiind în acelaşi timp ultimul şi primul element al listei.

7. Inserarea la sfârşitul listei

În acest caz trebuie mai întâi parcursă lista, după aceea de adăugat elementul, care va fi legat de restul listei. De asemenea, trebuie avut în vedere cazul în care lista este vidă.

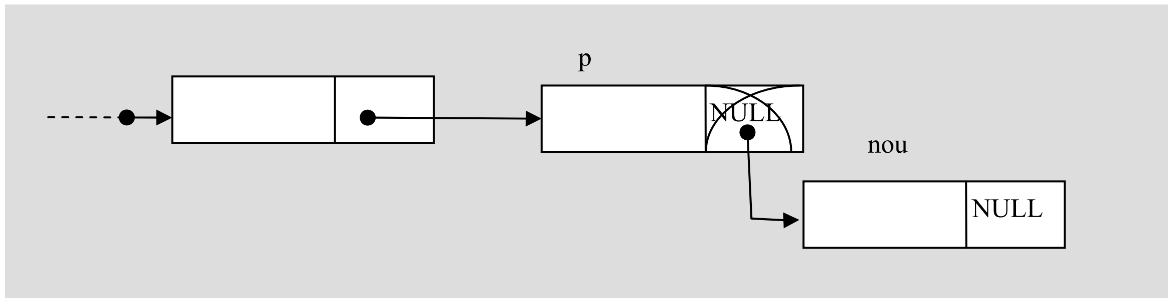

Fig.3. Adaugarea unui element la sfârşitul listei.

```
template<class T> void lista<T>::adauga_sfar(T p)
{
   Nod* p=new Nod;
   if(p==NULL)
   {
      cout<<"Eroare la alocare spatiu pentru inserare";
      return;
   }
   if(ultim)
   {
      p->urm=ultim->urm;
      ultim->urm=p;
      ultim=p;
   }
   else
   {
      ultim=p;
      p->urm=p;
   }
   cout << "Inserare sfarsit cu succes!\n";
}
```

Observăm că după adăugarea unui nou element la sfârşitul listei se actualizează pointerul *ultim*, care va păstra adresa noului element *ultim* al listei. Dacă elementul adăugat la sfârşitul listei este primul element al listei (caz în care pointerul *prim* conține valoarea NULL), vom actualiza şi valoarea pointerului *prim*, elementul adăugat fiind în acelaşi timp primul şi ultimul element al listei.

Operația de inserare a unui nod într-o listă nu se reduce numai la cele două situații prezentate, ci implică de asemenea posibilitatea inserării acestuia într-o poziție oarecare a listei, şi anume: după sau înaintea unui anumit nod specificat prin referința sa. Schematic cele două situații sunt prezentate în figura următoare, observându-se o oarecare dificultate la operația de inserție a noului nod înaintea celui specificat.

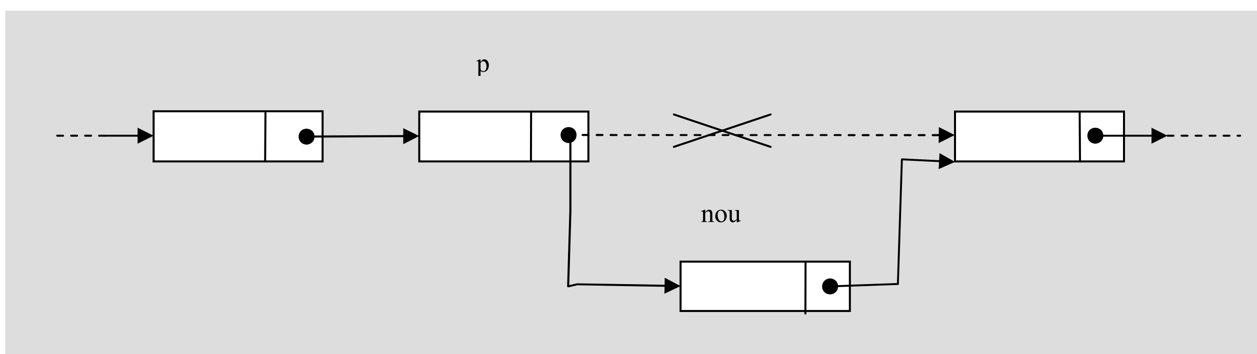

Fig.4. Adăugarea unui element după nodul p al listei.

```
template <class T> void lista<T>::adauga_dupa_un_nod(T nou)
{
   Nod* nou=new Nod;
   if(nou==NULL)
   {
      cout<<"Eroare la alocare spatiu pentru inserare";
      return;
   }
   if(ultim)
   {
      nou->urm=ultim->urm;
      ultim->urm=nou;
      ultim=nou;
   }
   else
   {
      ultim= nou;
      nou->urm= nou;
   }
   cout<<"Inserare sfarsit cu succes!\n";
}
```

8. Concatenarea listelor

Concatenarea este operația prin care din două liste *L1* şi *L2* cu elemente de acelaşi tip se obține a treia listă formată astfel: la lista *L1* sunt alipite elementele listei *L2*. Concatenarea listei *L1* cu lista *L2* presupune că pointerul *urm* la *ultimul* element al listei *L1* trebuie să-şi schimbe valoarea din NULL cu adresa primului element din lista *L2* (*a se vedea* Fig.5).

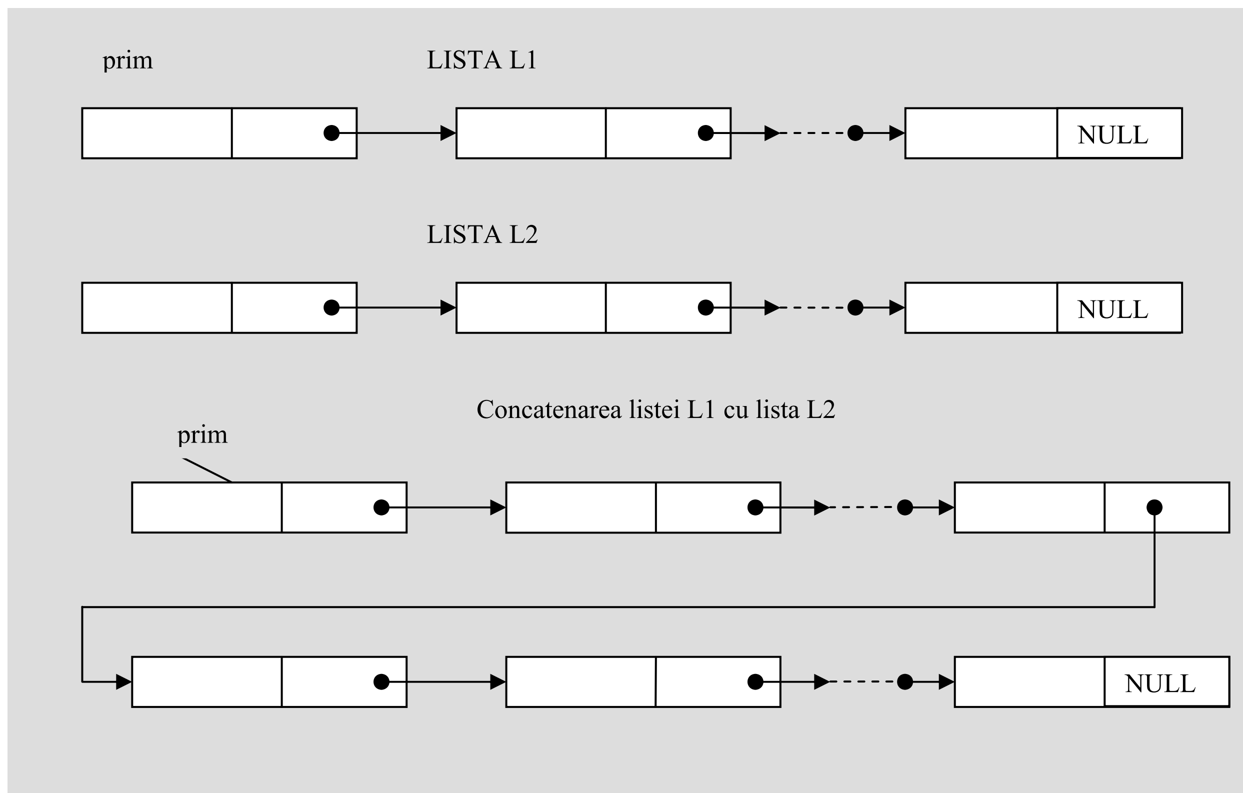

Fig.5. Concatenarea a două liste.

```
   template <class T> void lista <T>::concatliste(lista * cap1,
      lista * cap2)
   {
      Nod * p;
      for(p=cap1; p->urm!=NULL; p=p->urm)
      {
        p->urm=cap2;
        return cap1;
      }
}
```

9. Descrierea structurilor algebrice

În continuare definim un set de definiții pe care le vom folosi la cercetarea proprietăților algebrice ale listelor simplu înlănțuite şi a operațiilor asupra lor.

Definiția 5. *Vom numi **structură algebrică** o mulțime nevidă de elemente de tip T, echipată cu una sau mai multe operații algebrice ce satisfac o listă specifică de proprietăți numite axiomele structurii.*

Specific pentru structurile algebrice este numărul mare de modele (exemple) pe care le admit. Aceasta datorită faptului că la definirea structurilor algebrice sunt declarate axiome-liste de proprietăți care au fost întâlnite în multe cazuri concrete. Rezultatele obținute pentru o structură algebrică se extind imediat asupra oricărui model al structurii, ceea ce reprezintă unul dintre principalele avantaje ale utilizării metodei axiomatice în algebră. Însă, modelele (exemplele) unei structuri algebrice prezintă adesea şi interes particular (grupul permutărilor, inelul polinoamelor, inelul matricelor, spațiul vectorial R^n).

Se cunosc următoarele structuri algebrice: grup, inel şi corp [3].

Definiția 6. *Un cuplu $(G,*)$ format din mulțimea nevidă G şi operația algebrică "*", definită pe G, se numeşte **grup**, dacă sunt satisfăcute următoarele condiții:*

1. $(x*y)*z = x*(y*z)$, $\forall x,y,z \in G$;
2. $\exists e \in G$ *astfel încât* $e*x = x*e = x$, $\forall x \in G$;
3. $\forall x \in G$, $\exists x' \in G$ *astfel încât* $x*x' = x'*x = e$.

Definiția 7. *Grupul $(G,*)$ se numeşte **comutativ** (sau abelian), dacă este verificată axioma:*

$x*y = y*x$, $\forall x,y \in G$.

Definiția 8. *Un cuplu (A,φ) format dintr-o mulțime nevidă A şi o operație algebrică φ, definită pe A, se numeşte **grupoid.***

Definiția 9. *Grupoidul $(\mathrm{A},*)$ format dintr-o mulțime nevidă A şi o operație algebrică "*", definită pe A, se numeşte **semigrup**, dacă este satisfăcută condiția:*

$(x*y)*z = x*(y*z)$, $\forall x,y,z \in \mathrm{A}$.

Definiția 10. *Semigrupul $(\mathrm{A},*)$ se numeşte **monoid**, dacă operația algebrică φ posedă un element neutru.*

Definiția 11. *Un monoid $(\mathrm{A},*)$ pentru care operația "*" este comutativă se numeşte **monoid comutativ**.*

Definiția 12. *Un element e se numeşte **element neutru** pentru o lege de compoziție $x*y$, dacă $e*x = x*e = x$. În notație aditivă, elementul neutru se notează, de regulă, cu 0 şi se numeşte elementul zero, iar în notație multiplicativă elementul neutru se notează, de regulă, cu 1 sau chiar cu e şi se numeşte element-unitate. Pentru operația de adunare, definită pe mulțimea numerelor R, există $0 \in R$ cu proprietatea: $x+0 = 0+x = x$, $\forall x \in R$. Pentru operația de înmulțire, definită pe mulțimea numerelor R, există $1 \in R$ cu proprietatea: $x*1 = 1*x = x$, $\forall x \in R$.*

Definiția 13. *Un grup $(G,*)$ se numeşte **finit** dacă G constă dintr-un număr finit de elemente.*

10. Structura algebrică a listelor

Fie $E=\{e_1,e_2,...,e_n\}$ o mulţime de elemente de natură arbitrară de unul şi acelaşi tip.

Definiţia 14. *Prin listă asupra mulţimii* $E=\{e_1,e_2,...,e_n\}$*, notată* $L(e_{i_1},e_{i_2},...,e_{i_k})$*, vom înţelege mulţimea ordonată a elementelor* $\{e_{i_1},e_{i_2},...,e_{i_k}\}$*, unde* $e_{i_j}\in E, j=1,2,...,k$*, precum k – este numărul de elemente în listă* $L(e_{i_1},e_{i_2},...,e_{i_k})$*, iar elementul* e_{i_k} *se numeşte* ***ultimul element al listei***.

Notă. Lista poate fi şi vidă, dacă nu are nici un element [4]. Vom nota lista vidă prin "$\varnothing$", adică L () = $\varnothing$. Prin $L(e)$ vom nota lista dintr-un singur element $e\in E$. Notăm prin $\Lambda=\{L_1,L_2,...,L_m,...\}$ mulţimea tuturor listelor asupra mulţimii E. Introducem un set de operaţii cu liste.

11. Inserarea unui element în listă

Vom introduce operaţia care ne va da posibilitatea de a înscrie (adăuga) elemente noi în listă, pe care vom numi-o „*adăugare*" şi o vom nota prin semnul "+".

Definiţia 15. *Operaţia "+", care adaugă în lista* $L(e_{i_1},e_{i_2},...,e_{i_k})$ *elementul* $e_{i_{k+1}}$*, este operaţia care preface lista* $L(e_{i_1},e_{i_2},...,e_{i_k})$ *în lista* $L(e_{i_1},e_{i_2},...,e_{i_k},e_{i_{k+1}})$ *astfel încât elementul* $e_{i_{k+1}}$ *devine ultimul element al listei, adică* $L(e_{i_1},e_{i_2},...,e_{i_k})+e_{i_{k+1}}=L(e_{i_1},e_{i_2},...,e_{i_k},e_{i_{k+1}})$.

Notă. Deoarece un singur element $e_{i_j}\in E$ putem considera ca o listă dintr-un singur element $S(e_{i_j})$, vom scrie că $L(e_{i_1},e_{i_2},...,e_{i_k})+e_{i_{k+1}}=L(e_{i_1},e_{i_2},...,e_{i_k})+L(e_{i_{k+1}})=L(e_{i_1},e_{i_2},...,e_{i_k},e_{i_{k+1}})$.

Definiţia 16. Un cuplu (M,O) format dintr-o mulţime nevidă M şi o operaţie algebrică "O", definită pe M, se numeşte ***grupoid.***

Extindem operaţia „*adăugare*" pentru mai multe elemente, care ne va da posibilitatea să concatenăm liste.

Definiţia 17. *Operaţia "+", care concatenează lista* $L_i=L(e_{i_1},e_{i_2},...,e_{i_k})$ *cu lista* $L_j=L(e_{i_{k+1}},e_{i_{k+2}},...,e_{i_{k+m}})$*, este operaţia* $(...((L(e_{i_1},e_{i_2},...,e_{i_k})+e_{i_{k+1}})+e_{i_{k+2}})+...)+e_{i_{k+m}}$ *care are ca rezultat lista* $L(e_{i_1},e_{i_2},...,e_{i_k},e_{i_{k+1}},e_{i_{k+2}},...,e_{i_{k+m}})$*, astfel încât elementul* $e_{i_{k+m}}$ *este ultimul element al listei* L_i+L_J*, pentru orice* $L_i,L_J\in\Lambda$.

Astfel, mulţimea tuturor listelor asupra mulţimii *E* cu operaţia extinsă „*adăugare*" reprezintă un grupoid notat $(\Lambda,+)$.

Definiţia 18. *Vom spune că operaţia algebrică "O", definită pe mulţimea M, posedă element neutru, dacă există un element $\ell\in M$, astfel încât pentru orice element $e\in M$ $eO\ell=\ell Oe=e$.*

Notă. Elementul neutru se notează, de obicei, cu 0 şi se numeşte *element zero*.

Teorema 1. *Grupoidul* $(\Lambda,+)$ *posedă element neutru.*

Demonstraţie:

În calitate de element neutru se va lua lista vidă $\varnothing$. Luăm orice listă arbitrară $L_i\in\Lambda$. Fie $L_i=L(e_{i_1},e_{i_2},...,e_{i_k})$, atunci $L_i+\varnothing=L(e_{i_1},e_{i_2},...,e_{i_k})+\varnothing=\varnothing+L(e_{i_1},e_{i_2},...,e_{i_k})=L(e_{i_1},e_{i_2},...,e_{i_k})=L_i$. Teorema este demonstrată.

Teorema 2. *Grupoidul* $(\Lambda,+)$ *este semigrup.*

Demonstrație:

Fie L_i, L_j, L_r trei liste arbitrare din Λ. Demonstrăm că $(L_i + L_j) + L_r = L_i + (L_i + L_r)$.

Fie $L_i = L(e_{i_1}, e_{i_2}, ..., e_{i_{k_i}}),\ L_j = L(e_{j_1}, e_{j_2}, ..., e_{j_{k_i}}),\ L_r = L(e_{r_1}, e_{r_2}, ..., e_{r_{k_i}})$.

$$(L_i + L_j) + L_r = (L(e_{i_1}, e_{i_2}, ..., e_{i_{k_i}}) + L(e_{j_1}, e_{j_2}, ..., e_{j_{k_i}})) + L(e_{r_1}, e_{r_2}, ..., e_{r_{k_i}}) = L(e_{i_1}, e_{i_2}, ..., e_{i_{k_i}}) +$$
$$+ (L(e_{j_1}, e_{j_2}, ..., e_{j_{k_i}}) + L(e_{r_1}, e_{r_2}, ..., e_{r_{k_i}})) = L_i + (L_j + L_r).$$

pentru orice $L_i, L_j, L_r \in \Lambda$. Teorema este demonstrată**.**

Teorema 3. *Semigrupul* $(\Lambda,+)$ *este monoid.*

Demonstrație:

Din teorema 1 şi teorema 2 rezultă că grupoidul $(\Lambda,+)$ este semigrup. Trebuie să demonstrăm că grupoidul $(\Lambda,+)$ este monoid.

Fie o listă arbitrară $L_i \in \Lambda$. Demonstrăm că $L_i + \varnothing = \varnothing + L_i = L_i$. Fie $L_i = L(e_{i_1}, e_{i_2}, ..., e_{i_k})$, atunci $L_i + \varnothing = L(e_{i_1}, e_{i_2}, ..., e_{i_k}) + \varnothing = \varnothing + L(e_{i_1}, e_{i_2}, ..., e_{i_k}) = L(e_{i_1}, e_{i_2}, ..., e_{i_k}) = L_i$. $L_i \in \Lambda$.

Deci, grupoidul $(\Lambda,+)$ este monoid.

Concluzii

În articol a fost descrisă în detalii una dintre cele mai importante şi utilizate structuri dinamice de date – listă simplu înlănțuită. Au fost introduse definițiile formalizate pentru structura dată şi ale noțiunile legate de ea. Au fost cercetate proprietățile algebrice ale listelor simplu înlănțuite şi operațiile asupra lor. S-a demonstrat că totalitatea listelor simplu înlănțuite cu elementele de unul şi acelaşi tip *T*, dotată cu operația de concatenare, este monoid. A fost propusă realizarea în limbajul C++ a unei clase generice parametrizate cu clasa abstractă arbitrară *T*, care permite modelarea lucrului cu liste simplu înlănțuite din punctul de vedere al structurilor algebrice.

Referințe:

1. Knuth D.E. Tratat de programare a calculatoarelor. - Bucureşti: Editura Tehnică, 1974.
2. Zaharia M.D. Structuri de date şi algoritmi, exemple în limbajele C şi C++. - Cluj-Napoca, 2002.
3. Goian I., Sârbu P., Topală A. Grupuri şi inele. - Chişinău: CEP USM, 2005.
4. Bitcovschi L. Cercetarea listelor ca structuri algebrice. – În: Materialele Conferinței științifice internaționale „Modelarea Matematică, Optimizare şi Tehnologii Informaționale”. Chişinău, 24-26 martie 2010, p.384-391.

Prezentat la 12.08.2011