1 Laborator 3: Liste Circulare si Liste Dublu Inlantuite

1.1 Obiective

În lucrare sunt prezentate principalele operaţii asupra listelor circulare simplu înlănţuite: crearea, inserarea unui nod, ştergerea unui nod şi ştergerea listei. În a doua parte vor fi prezentate operaţiile asupra listelor dublu înlănţuite.

1.2 Noţiuni teoretice

1.2.1 Definiţie liste circulare simplu inlantuite

O *listă circulară simplu înlănţuită* este lista simplu înlănţuită a cărei ultim element este legat de primul element. Fiind o listă circulară, practic nu are capete. Astfel vom folosi un singur pointer $pNode$ pentru a indica un element din listă – cel mai nou element. Figura 1.1 arată modelul unei astfel de liste.

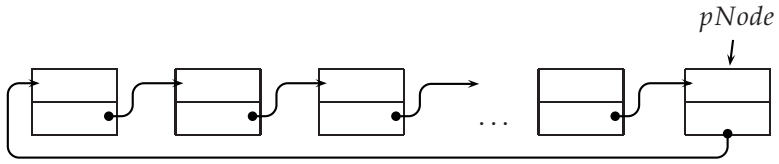

Figure 1.1: Modelul unei liste circulare simplu înlănţuite

Structura unui nod poate fi declarată astfel:

```
typedef struct nodetype
{
  int key; /* camp optional */
  /* alte campuri utile */
  struct nodetype *next;
  /* legatura catre nodul urmator */
} NodeT;
```

1.2.2 Operaţii cu liste circulare simplu înlănţuite

Crearea unei liste circulare simplu înlănţuite

1. Iniţial lista este vidă, adică: $pNode = NULL$.
2. Se generează un nod pentru a fi inserat în listă:

```
/* aloca spatiu */
p = ( NodeT * )malloc( sizeof( NodeT ));
/* Se citesc datele in nodul adresat prin  p*/
```

3. Se leagă nodul în listă:

```
p->next = NULL;
/* nodul este adaugat la lista */
if ( pNode == NULL )
{ /* lista vida */
  pNode = p;
  pNode->next = p;
}
else
{ /* lista nu e vida */
  p->next = pNode->next;
  pNode->next = p;
  pNode = p;
  /* pNode pointeaza la cel mai nou nod din lista */
}
```

Acesarea unui nod dintr-o listă circulară simplu înlănţuită

Nodurile unei liste pot fi accesate secvenţial pornind de la nodul $pNode$ după cum urmează:

```
NodeT *p;

p = pNode;
if ( p != NULL )
  do
  {
                /* TODO -- se acceseaza nodul curent si se iau datele */
    p = p->next;
  }
  while ( p != pNode );
```

O altă posibilitate este să se caute o cheie, de exemplu $givenKey$. Codul care parcurge o listă pana găseşte nodul cu cheia căutată este dat mai jos:

```
NodeT *p;

p = pNode;
if ( p != NULL )
  do
  {
    if ( p->key = givenKey )
    { /* cheia este gasita la adresa p */
      return p;
    }
    p = p->next;
  }
  while ( p != NULL );
return NULL; /* nu s-a gasit nodul */
```

Inserarea unui nod într-o listă circulară simplu înlănţuită

Un nod poate fi inserat *înaintea* unui nod care conţine o cheie dată, sau *după* acesta.

În ambele cazuri se caută nodul cu cheia dată şi dacă cheie există, se crează un nod de inserat şi se ajustează legăturile în mod corespunzător.

Inserarea înaintea unui nod cu cheia *givenKey*

Procedura se execută în doi paşi:

1. Se găseşte nodul cu cheia $givenKey$:

   ```
   NodeT *p, *q, *q1;

   q1 = NULL; /* initializare */
   q = pNode;
   do
   {
     q1 = q;
     q = q->next;
     if ( q->key == givenKey ) break;
   }
   while ( q != pNode );
   ```

2. Inserarea nodului la care pointează p, şi ajustarea legăturilor:

   ```
   if ( q->key == givenKey )
   { /* nodul cu cheia givenKey are adresa q */
     q1->next = p;
     p->next = q;
   }
   ```

Inserarea după un nod cu cheia *givenKey*

Procedura se realizează în doi paşi:

1. Se găseşte nodul cu cheia $givenKey$:

```
NodeT *p, *q;

q = pNode;
do
{
  if ( q->key == givenKey ) break;
  q = q->next;
}
while ( q != pNode );
```

2. Inserarea nodului la care pointează p, şi ajustarea legăturilor:

```
if ( q->key == givenKey )
{ /* nodul cu cheia givenKey are adresa q  */
  p->next = q->next;
  q->next = p;
}
```

Ştergerea unui nod dintr-o listă circulă simplu înlănţuită

Procedura se realizează în doi paşi:

1. Se găseşte nodul cu cheia $givenKey$:

```
NodeT *p, *q, *q1;
q = pNode;
do
{
  q1 = q;
  q = q->next;
  if ( q->key == givenKey ) break;
}
while ( q != pNode );
```

2. Se şterge nodul la care pointeză q. Dacă nodul respectiv este $pNode$ atunci ajustăm legăturile astfel încât $pNode$ pointează la nodul anterior.

```
if ( q->key == givenKey )
{ /* nodul cu cheia givenKey are adresa q */
  if ( q == q->next )
  {
    /* lista e vida */
  }
  else
  {
    q1->next = q->next;
    if ( q == pNode ) pNode = q1;
  }
  free( q );
}
```

Ştergerea unei liste circulare simplu înlănţuite

Pentru ştergerea completă a unei liste circulare simplu înlănţuite se şterge fiecare nod din listă aşa cum se vede în exemplu de cod următor:

```
NodeT *p, *p1;
p = pNode;
do
{
  p1 = p;
  p = p->next;
  free( p1 );
}
while ( p != pNode );
pNode = NULL;
```

1.2.3 Definiţie listă dublu înlănţuită

Lista *dublu înlănţuită* este lista dinamică între nodurile căreia s-a definit o dublă relaţie: de succesor si de predecesor. Modelul unei astfel de liste este dat în figura 1.2. Tipul unui nod dintr-o listă dublu înlănţuită este definit astfel:

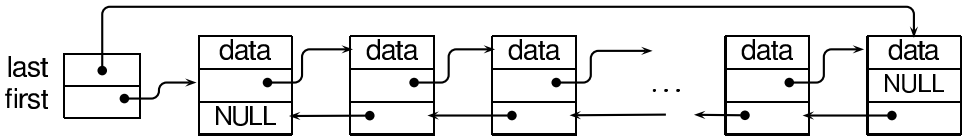

Figure 1.2: Modelul unei liste dublu înlănţuite.

```
typedef struct node_type
{
  KeyT key; /* optional */
  ValueT value;
  /* pointer catre nodul urmator */
  struct node_type *next;
  /* pointer catre nodul anterior */
  struct node_type *prev;
} NodeT;
```

Principalele operaţii cu liste dublu înlăţuite sunt următoarele:

- crearea;
- accesul la un nod;
- inserarea unui nod;
- ştergerea unui nod,
- ştergerea listei.

1.2.4 Operaţii cu liste dublu înlănţuite

Pentru explicarea operaţiilor cu liste dublu înlănţuite vom presupune că lista este dată de un pointer la nodul de început al listei, adică:

```
/* nod de inceput */
struct list_header
{
  NodeT *first;
  NodeT *last;
};
/* lista este definita ca un pointer la primul nod */
struct list_header *L;
```

Crearea unei liste dublu înlănţuite

Iniţial lista este vidă:

```
L->first = L->last = NULL;
```

Se alocă spaţiu pentru un nod nou şi se populează câmpurile de date din nod (campul *value*). Inserarea acestui nod la care pointează variabila *p* este făcută după cum urmează:

```
if ( L->first == NULL )
{ /* lista e vida */
  L->first = L->last = p;
  p->next = p->prev = NULL;
}
else
{ /* lista nu e vida */
  L->last->next = p;
  p->prev = L->last;
  L->last = p;
}
```

Accesul la un nod al unei liste dublu înlănţuite

Accesul la un nod se poate face:

- secvenţial înainte

```
for ( p = L->first; p != NULL; p = p->next )
{
  /* se poate face o operatie asupra nodului curent */
}
```

- Secvenţial înapoi

```
for ( p = L->last; p != NULL; p->p->prev )
{
  /* se poate face o operatie asupra nodului curent */
}
```

- Pe baza unei chei. Căutarea unui nod de cheie dată $givenKey$ se va face identic ca la lista simplu înlănţuită Lab. **??**, §**??**.

Inserarea unui nod într-o listă dublu înlănţuită

Inserarea unui nod într-o listă dublu înlănţuită se poate face astfel:

- înaintea primului nod:

```
if ( L->first == NULL )
{ /* lista vida */
  L->first = L->last = p;
  p_>next = p->prev = NULL;
}
else
{ /* lista nu e vida */
  p->next = L->first;
  p->prev = NULL;
  L->first->prev = p;
  L->first = p;
}
```

- după ultimul nod:

```
if ( L->first == NULL )
{ /* lista vida*/
  L->first = L->last = p;
  p_>next = p->prev = NULL;
}
else
{ /* lista nu e vida */
  p->next = NULL;
  p->prev = L->last;
  L->last->next = p;
  L->last = p;
}
```

- După un nod de cheie dată key, presupunând că acesta există şi are adresa q:

```
p->prev = q;
p->next = q->next;
if ( q->next != NULL ) q->next->prev = p;
q->next = p;
if ( L->last == q ) L->last = p;
```

Ştergerea unui nod dintr-o listă dublu înlănţuită

Există următoarele cazuri de ştergere a unui nod din listă:

- Ştergerea primului nod:

```
p = L->first;
L->first = L->first->next; /* presupunem ca lista nu este vida */
free( p ); /* se elibereaza memoria */
if ( L->first == NULL )
  L->last == NULL; /* lista devine vida*/
else
  L->first->prev = NULL;
```

- Ştergerea ultimului nod:

```
p = L->last;
L->last = L->last->prev; /* presupunem ca lista nu este vida */
if ( L->last == NULL )
  L->first = NULL; /* lista devine vida */
else
  L->last->next = NULL;
free( p ); /* se elibereaza memoria */
```

- Ştergerea unui nod precizat printr-o cheie $givenKey$. Presupunem că nodul de cheie $givenKey$ există şi are adresa p (rezultă din căutarea sa):

```
if ( L->first == p && L->last == p )
{ /* lista are un singur nod */
  L->first = NULL;
  L->last = NULL; /* lista devine vida */
  free( p );
}
else
if ( p == L->first )
{ /* deletion of first node */
  L->first = L->first->next;
  L->first->prev = NULL;
  free( p );
}
else
{ /* stergerea unui nod intern */
  p->next->prev = p->prev;
  p->prev->next = p->next;
  free( p );
}
```

Ştergerea unei liste dublu înlănţuite

Ştergerea întregii liste se realizează ştergând nod cu nod astfel:

```
NodeT *p;

while ( L->first != NULL )
{
  p = L->first;
  L->first = L->first->next;
  free( p );
}
L->last = NULL;
```

1.3 Mersul lucrării

1.3.1 Probleme Obligatorii:

1.1. Să se studieze exemplele de cod date în laborator.

1.2. De la tastatură se citeşte numărul n şi numele a n copii. Să se simuleze următorul joc: cei n copii stau într-un cerc. Începând cu un anumit copil, se numără copiii în sensul acelor de ceasornic. Fiecare al n-lea copil iese din cerc .Câştigă ultimul copil rămas în joc.

1.3. De la tastatură se citesc n cuvinte; să se creeze o listă dublu înlănţuită, care să conţină în noduri cuvintele distincte şi frecvenţa lor de apariţie. Lista va fi ordonată alfabetic. Se vor afişa cuvintele şi frecvenţa lor de apariţie a) în ordine alfabetică crescătoare şi b) în ordine alfabetică descrescătoare.

1.3.2 Probleme Pentru Puncte în Plus

1.4. Să se implementeze o listă dublu înlănţuită de tip XOR. Pentru o astfel de listă nu avem pointerii prev sau next, ci se foloseşte un singur pointer care reprezintă şi pointerul anterior şi pe cel următor. Acest lucru este posibil folosind operaţia XOR, şi următoarele consideraţii:

```
A XOR B = C

C XOR A = B
C XOR B = A
```

Când se crează o astfel de listă se realizează un XOR între next şi previous, şi se salvează pointerul la primul element. Să presupunem că A = previous, B = next, C = valoarea stocată.

```
previous XOR next = valoarea stocata
valoarea stocata XOR previous = next
valoarea stocata XOR next = previous
```

Dacă se traversează lista XOR dublu înlănţuită, se cunoaşte elementul curent şi elementul anterior, asa că utilizând operaţiile de mai sus se poate calcula adresa elementului următor din listă.

1.3.3 Probleme Optionale:

1.5. Definiţi şi implementaţi funcţii care permit operaţii pe structura de date de mai jos având modelul în figura 1.3.

```
struct circularList
{
  int length;
  NodeT *first;
}
```

Codificaţi operaţiile după cum urmează: `ins` $data$= inserează $data$ în listă, în cazul în care nu există deja (verificaţi cheia), `del` key = stergeţi din listă datele care au cheia key (dacă acestea sunt în listă), `ist` $data$= inseraţi nodul avand datele $data$ ca prim nod, `dst` = ştergeţi primul nod, `prt` = afişaţi lista, `fnd` key= găsiţi nodul care conţine key în zona de date.

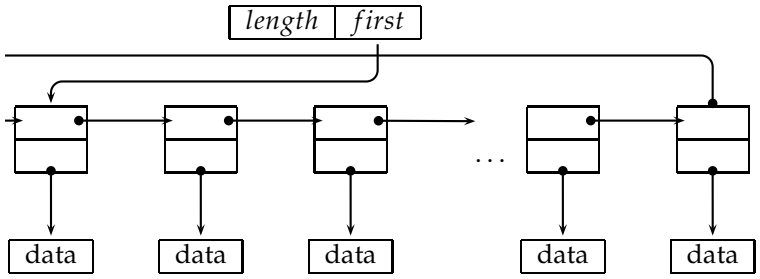

Figure 1.3: Un alt model al unei liste circulare simplu înlănţuite

1.6. Folosiţi o stivă pentru a inversa ordinea elementelor într-o listă circulară ordonată crescător. Datele (care reprezintă şi cheia nodurilor) sunt nume de persoane. .
I/O description. Input:

```
Ionescu␣Ion
Vasilescu␣Traian
Doe␣John
Kong␣King
Mickey␣Mouse
Curtis␣Anthony
Bugs␣Bunny
```

Output:

```
Vasilescu␣Traian
Mickey␣Mouse
Kong␣King
Ionescu␣Ion
Doe␣John
Curtis␣Anthony
Bugs␣Bunny
```

1.7. Folosiţi o stivă pentru a inversa ordinea elementelor într-o listă circulară ordonată descrescător. Datele (care reprezintă şi cheia nodurilor) sunt nume de persoane. .
I/O description. Input:

```
Ionescu␣Ion
Vasilescu␣Traian
Doe␣John
```

```
Kong␣King
Mickey␣Mouse
Curtis␣Anthony
Bugs␣Bunny
```

Output:

```
Bugs␣Bunny
Curtis␣Anthony
Doe␣John
Ionescu␣Ion
Kong␣King
Mickey␣Mouse
Vasilescu␣Traian
```

1.8. Folosind o listă circulară dublu înlănţuită să se simuleze următorul joc: n copii, ale căror nume se citesc de la tastatură, stau în cerc. Începând cu un anumit copil (numele său se citeşte), se numără copiii în sensul acelor de ceasornic. Fiecare al m-lea copil (m se citeşte) iese din joc. Numărătoarea continuă începând cu următorul copil din cerc. Câştigă jocul ultimul copil rămas în cerc.

1.9. Simulaţi următorul joc, folosind liste circulare dublu înlănţuite. Jocul presupune citirea numelor unor copii dintr-un fişier. Copii ale căror nume sunt pe linii cu numere prime trebuie plasaţi în prima listă, iar ceilaţi copii în a doua listă. Pornind de la copilul al cărui nume este în mijlocul (sau $\lfloor numarCopii/2 \rfloor$) celei de a doua liste se numără copiii în sensul acelor de ceasornic. Fiecare al m^{lea} copil, unde m este numărul de elemente din prima listă este eliminat din a doua listă. Se repetă numărătoarea de m ori sau până cand a doua listă devine vidă. Afişaţi listele iniţiale şi a doua listă la final.
Example. Input:

```
John
James
Ann
George
Amy
Fanny
Winston
Bill
Suzanna
Hannah
Alex
Gaby
Thomas
```

Output:

```
Prime␣[7␣children]:
------
John
James
Ann
Amy
Winston
Alex
Thomas

Non-prime␣[6␣children]:
----------
George
Fanny
Bill
Suzanna
Hannah
Thomas

Starter:␣Bill
Count:␣7

Empty␣list
```