Structuri de Date şi Algoritmi
Laborator IV

Adrian Liţă

2015

1 Scopul laboratorului

Laboratorul IV al materiei Structuri de Date şi Algoritmi are ca scop introducerea structurilor de date **coadă**, **stivă** şi **listă generalizată**.
În acest laborator se parcurg următoarele puncte:

- structura de date **FIFO**
- modul de funcţionare al cozilor
- funcţiile de bază ale cozilor
- structura de date **LIFO**
- modul de funcţionare al stivelor
- funcţiile de bază ale stivelor

2 Desfăşurarea lucrării

Cozile şi stivele sunt structuri de date derivate din **lista simplu înlănţuită**.

2.1 Cozi

Cozile sunt structuri de date **FIFO** (First In First Out). Practic o coadă este o listă simplu înlănţuită, cu următoarele proprietăţi:

- primul element adăugat va fi primul element scos
- modelează o coadă în practică

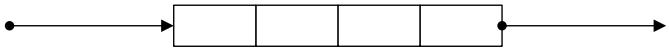

Figura 1: Coada de date

Pentru a creea un tip de date numit **COADĂ** este absolut necesar să ţinem minte şi începutul listei şi sfârşitul listei. Pentru a uşura codul de programare, vom introduce elemente noi la sfârşitul listei şi vom şterge elemente doar de la începutul listei.

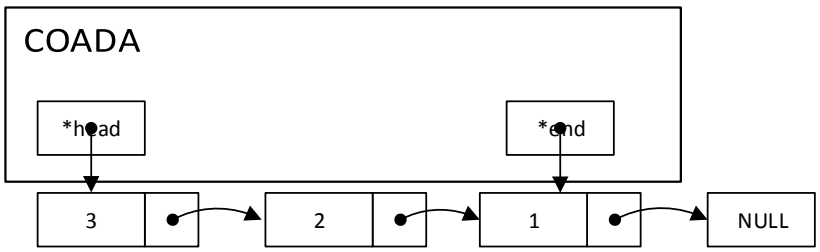

Figura 2: Coadă cu elementele adăugate în următoarea ordine: 1, 2, 3

Pentru a adăuga şi scoate elemente din coadă, funcţiile folosite sunt exact cele de la o listă simplu înlănţuită, particularizate pentru cazul în care se folosesc: adăugarea se face doar pe ultima poziţie şi pointerul ***end** este cunoscut, iar ştergerea se face ştergând doar primul element.

```
typedef struct node
{
  int info;
  struct node *next;
} NODE, *PNODE;

typedef struct coada
{
  PNODE head;
  PNODE end;
} COADA, *PCOADA;

PCOADA creare_coada()
{
  PCOADA n = (PCOADA) malloc(sizeof(COADA)*1);
  n->head = NULL;
  n->end = NULL;

  return n;
}

void adauga_in_coada(PCOADA c, int info_new)
{
  if(c == NULL)
    return;

  PNODE temp = NULL;
  temp = (PNODE) malloc(sizeof(NODE)*1);
  if(temp == NULL)
  {
    printf("Eroare la alocare!\n");
    exit(0);  //nu are sens sa continui
  }
```

```
34    temp->info = info_new;
35    temp->next = NULL;
36
37    if(c->head == NULL)
38    {
39      c->head = temp;
40      c->end = temp;
41    }
42    else
43    {
44      c->end->next = temp;
45      c->end = temp;
46    }
47  }
48
49  int extragere_din_coada(PCOADA c, int *info_return)
50  {
51    //functia returneaza 1 daca s-a extras corect din coada
52    //sau 0 daca coada era goala si nu s-a extras nimic
53    if(c == NULL)
54      return 0;
55
56    if(c->head == NULL)
57      return 0;
58
59    PNODE elem1 = c->head;
60    *info_return = elem1->info;
61    c->head = elem1->next;
62    free(elem1);
63
64    return 1;
65  }
66
67  int main()
68  {
69    PCOADA s = creare_coada();
70    adauga_in_coada(s,1);
71    adauga_in_coada(s,2);
72    adauga_in_coada(s,3);
73
74    int info;
75    if(extragere_din_coada(s, &info) == 0)
76    {
77      printf("Coada este goala!\n");
78    }
79    else
80    {
81      printf("%d\n", info);
82    }
83
84    if(extragere_din_coada(s, &info) == 0)
85    {
86      printf("Coada este goala!\n");
87    }
88    else
89    {
90      printf("%d\n", info);
```

```
91    }
92
93    if(extragere_din_coada(s, &info) == 0)
94    {
95      printf("Coada este goala!\n");
96    }
97    else
98    {
99      printf("%d\n", info);
100   }
101
102   if(extragere_din_coada(s, &info) == 0)
103   {
104     printf("Coada este goala!\n");
105   }
106   else
107   {
108     printf("%d\n", info);
109   }
110
111   return 0;
112 }
```

Funcţia **creare_coada()** crează dinamic o coadă goală, iniţializează pointerii ***head** şi ***end** cu **NULL** şi returnează adresa cozii nou create. În **main()** este exemplificată folosirea acestei funcţii.

Funcţia **adauga_in_coada(PCOADA c, int info_new)** adaugă un element de tip **int** (info_new) într-o coadă dată prim pointerul ***c**. Dacă coada dată ca argument este neiniţializată (necreată), funcţia nu face nimic.

Funcţia **extrage_din_coada(PCOADA c, int *info_return)** extrage un element de tip **int** din coada dată prin pointerul ***c** şi îl va returna la ieşirea din funcţie prin pointerul ***info_return**. Dacă operaţiunea de extragere a decurs corect, funcţia va returna **1**. Dacă coada dată ca argument este neiniţializată (necreată), sau dacă coada dată ca argument este goală (iniţializată dar nu conţine elemente) funcţia va returna valoarea **0**.

Notă. *Dacă în funcţia de extragere am fi returnat direct valoarea extrasă, funcţia trebuie să returneze mereu o informaţie (prin definiţia funcţiei) indiferent dacă acea informaţie este corectă sau nu. Returnând şi informaţia şi statusul ei (faptul că este validă sau nu) ne asigură că operaţia se desfăşoară corect, sau ne atenţionează dacă nu (va returna **0**.*

2.2 Stive

Stivele sunt structuri de date **LIFO** (Last In First Out). Practic o stivă este o listă simplu înlănţuită, cu următoarele proprietăţi:

- ultimul element adăugat va fi primul element scos
- modelează o stivă în practică

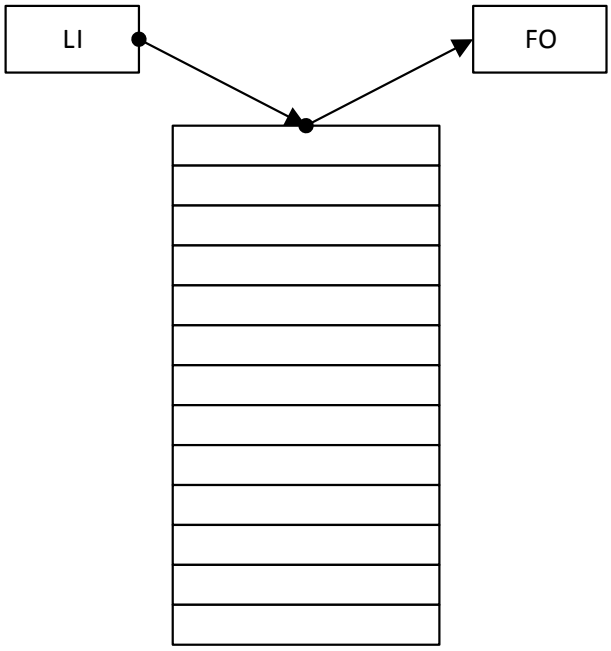

Figura 3: Stiva de date

Pentru a creea un tip de date numit **STIVĂ** nu trebuie memorat decât începutul listei. Acela va fi locul din care vom adăuga şi din care vom extrage informaţii. Pentru a adăuga şi scoate elemente din stivă, funcţiile folosite sunt exact cele de la o listă simplu înlănţuită, particularizate pentru cazul în care se folosesc: adăugarea se face doar pe prima poziţie, iar extragerea se face ştergând doar primul element.

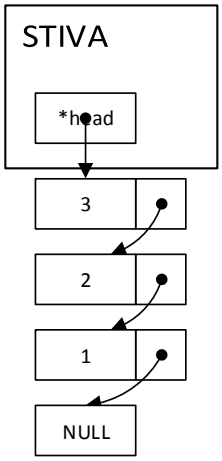

Figura 4: Stivă cu elementele adăugate în următoarea ordine: 1, 2, 3

Funcţiile pentru lucru cu stiva se numesc în literatură **push()** (adăugare) şi **pop()** (extragere). La fel ca în exemplul cozii, funcţia **pop()** extrage un element de tip **int** din stiva dată prin pointerul ***c** şi îl va returna la ieşirea din funcţie prin pointerul ***info_return**. Dacă operaţiunea de extragere a decurs corect, funcţia va returna **1**. Dacă stiva dată ca argument este neiniţializată (necreată), sau dacă stiva dată ca argument este goală (iniţializată dar nu conţine elemente) funcţia va returna valoarea **0**.

```
typedef struct node
{
  int info;
  struct node *next;
} NODE, *PNODE;

typedef struct stiva
{
  PNODE head;
} STIVA, *PSTIVA;

PSTIVA creare_stiva()
{
  PSTIVA n = (PSTIVA)malloc(sizeof(STIVA)*1);
  n->head = NULL;

  return n;
}

void push(PSTIVA s, int info_new)
{
  if(s == NULL)
    return;

  PNODE temp = NULL;
```

```
26    temp = (PNODE)malloc(sizeof(NODE)*1);
27    if(temp == NULL)
28    {
29      printf("Eroare la alocare!\n");
30      exit(0);  //nu are sens sa continui
31    }
32    temp->info = info_new;
33    temp->next = s->head;
34
35    s->head = temp;
36  }
37
38  int pop(PSTIVA s, int *info_return)
39  {
40    if(s == NULL)
41      return 0;
42
43    if(s->head == NULL)
44      return 0;
45
46    PNODE elem1 = s->head;
47    *info_return = elem1->info;
48    s->head = elem1->next;
49    free(elem1);
50
51    return 1;
52  }
53
54  int main()
55  {
56    PSTIVA s = creare_stiva();
57    push(s,1);
58    push(s,2);
59    push(s,3);
60
61    int info;
62    if(pop(s, &info) == 0)
63    {
64      printf("Stiva este goala!\n");
65    }
66    else
67    {
68      printf("%d\n", info);
69    }
70
71    if(pop(s, &info) == 0)
72    {
73      printf("Stiva este goala!\n");
74    }
75    else
76    {
77      printf("%d\n", info);
78    }
79
80    if(pop(s, &info) == 0)
81    {
82      printf("Stiva este goala!\n");
```

```
83    }
84    else
85    {
86      printf("%d\n", info);
87    }
88
89    if(pop(s, &info) == 0)
90    {
91      printf("Stiva este goala!\n");
92    }
93    else
94    {
95      printf("%d\n", info);
96    }
97
98    return 0;
99  }
```

3 Probleme

1. Implementaţi funcţiile de cozi şi stive cu ajutorul fişierelor .h
2. Reimplementaţi funcţia de adăugare la cozi şi funcţia push() la stive să returneze 0 sau 1 în funcţie de starea de execuţie a adăugarii (1 dacă elementul s-a adăugat, 0 dacă nu).
3. Implementaţi o bandă rulantă pentru transportul pachetelor cu ajutorul unei cozi.
4. Implementaţi o un palet de pachete cu ajutorul unei stive.
5. Implementaţi o bandă rulantă pentru transportul unor paleţi cu pachete.