STIVA

I. ASPECTE TEORETICE

O stiva este o lista simplu inlantuita gestionata conform principiului LIFO (Last In First Out).

Conform acestui principiu, ultimul nod pus in stiva este primul nod care este scos din stiva. Stiva, ca si lista, are doua capete: **baza stivei** si **varful stivei**.

Asupra unei stive se definesc cateva operatii, dintre care cele mai importante sunt:

1. pune un element pe stiva (**PUSH**);

2. scoate un element din stiva (**POP**);

3. sterge (videaza) stiva (**CLEAR**).

Primele doua operatii se realizeaza in varful stivei. Astfel, daca se scoate un element din stiva, atunci acesta este cel din varful stivei si in continuare, cel pus anterior lui pe stiva ajunge in varful stivei.
Daca un element se pune pe stiva, atunci acesta se pune in varful stivei si in continuare el ajunge in varful stivei.

Pentru a implementa o stiva printr-o lista simplu inlantuita va trebui sa identificam baza si varful stivei cu capetele listei simplu inlantuite. Exista doua posibilitati:

a. nodul spre care pointeaza variabila prim este baza stivei, iar nodul spre care pointeaza variabila ultim este varful stivei;

b. nodul spre care pointeaza variabila prim este varful stivei, iar nodul spre care pointeaza variabila ultim este baza stivei.

In cazul a, functiile PUSH si POP se identifica prin functiile **adauga** si respectiv **sun**, definite in sedinta trecuta de laborator. Daca functia adauga este eficienta, in schimb functia sun nu este eficienta.

In cazul b, functiile PUSH si POP se identifica prin functiile **iniprim** si respectiv **spn**, ce vor fi definite in aceasta lucrare de laborator. In acest caz, ambele functii sunt eficiente. De aceea, se recomanda implementarea stivei printr-o lista simplu inlantuita conform cazului b indicat mai sus.

II. DESFASURAREA LUCRARII

Vom avea de rezolvat urmatoarea problema:

Intr-o gara se considera un tren de marfa ale carui vagoane sunt inventariate intr-o lista, in ordinea vagoanelor. Lista contine, pentru fiecare vagon, urmatoarele date:

- codul vagonului (9 cifre);
- codul continutului vagonului (9 cifre);
- adresa expeditorului (4 cifre);
- adresa destinatarului (4 cifre).

Deoarece in gara se inverseaza pozitia vagoanelor, se cere listarea datelor despre vagoanele respective in noua lor ordine.

In acest scop se creeaza o stiva in care se pastreaza datele fiecarui vagon. datele corespunzatoare unui vagon constituie un element al stivei, adica un nod al listei simplu inlantuite. Dupa ce datele au fost puse pe stiva, ele se scot de acolo si se listeaza. In acest mod se obtine lista vagoanelor in ordine inversa celei initiale.

Va fi definita in fisierul programului principal structura:

```
typedef struct tnod
        {
         long cvag;
         long cmarfa;
         int exp;
         int dest;
         struct tnod *urm;
        } TNOD;
```

OBSERVATIE: Fiecare punct al DESFASURARII LUCRARII va fi scris intr-un fisier separat cu extensia **.cpp**.

1. Sa se scrie o functie

int pcit_int(char text[], int *x)

care afiseaza caracterele unui tablou si citeste un intreg de tip int. Ea va returna 0 la intalnirea sfarsitului de fisier si 1 altfel.

Functia are doi parametri:

text - tablou unidimensional de tip caracter; functia afiseaza sirul de caractere, pastrat in tabloul text, inainte de a citi intregul.

x - pointer spre intregi de tip int (are ca valoare adresa zonei de memorie in care se pastreaza valoarea intregului citit).

O posibila solutie:

```
int pcit_int(char text[], int *x)
/* - citeste un intreg si-l pastreaza in zona de memorie a carei adresa este valoarea lui x;
   - returneaza:
          0 - la intalnirea sfarsitului de fisier;
          1 - altfel. */
{
 char t[255];
 for( ; ; )
          {
           printf(text);
           if(gets(t) == NULL)
                    return 0;
           if(sscanf(t,"%d",x) == 1)
                    return 1;
          }
}
```

2. Sa se scrie o functie

int pcit_int_lim(char text[], int inf, int sup, int *pint)

care citeste un intreg de tip int care apartine unui interval dat. Ea are parametri:

text - tablou unidimensional de tip char;

inf - intreg de tip int, care reprezinta limita inferioara a intervalului carui trebuie sa-i apartina numarul citit;

sup - intreg de tip int, care reprezinta limita superioara a intervalului carui trebuie sa-i apartina numarul citit;
pint - pointer spre intregi de tip int; are ca valoare adresa zonei de memorie in care se pastreaza numarul citit.

Functia returneaza:
0 - la intalnirea sfarsitului de fisier;
1 - altfel.

O posibila solutie:

```
int pcit_int_lim(char text[], int inf, int sup, int *pint)
/* - citeste un intreg de tip int ce apartine intervalului [inf, sup] si-l pastreaza in zona de memorie a carei
    adresa este valoarea parametrului pint;
  - returneaza:
        0 - la intalnirea sfarsitului de fisier;
        1 - altfel. */
{
 for( ; ; )
        {
         if(pcit_int(text, pint) == 0)
                return 0; /* s-a intalnit EOF */
         if(*pint >= inf && *pint <= sup)
                return 1;
         printf("\nIntregul tastat nu apartine intervalului:");
         printf("[%d,%d]\n", inf, sup);
         printf("Se reia citirea\n");
        }
}
```

3. Sa se scrie o functie

int incnod(TNOD *p)

care incarca nodul curent cu datele despre vagoane (codul vagonu-lui, codul continutului vagonului, adresa expeditorului si adresa destinatarului).

O pasibila solutie:

```
int incnod(TNOD *p)
/* incarca un nod cu datele despre vagoane */
{
 char t[255];
 char er[] = "S-a tastat EOF in pozitie rea\n";
 long cod;
 int icod;

 /* citeste cod vagon */
 for( ; ; )
 {
 printf("cod vagon: ");
 if(gets(t) == 0)
        return -1; /* nu mai sunt date */
 if(sscanf(t, "%ld", &cod) == 1 && cod >= 0 && cod <= 999999999)
        break;
 printf("cod vagon eronat\n");
 }
 p -> cvag = cod;
```

```
 /* citeste cod marfa */
 for( ; ; )
 {
 printf("cod marfa: ");
 if(gets(t) == 0)
    {
     printf(er);
     return 0;
    }
 if(sscanf(t, "%ld", &cod) == 1 && cod >= 0 && cod <= 999999999)
    break;
 printf("cod marfa eronat\n");
 }
 p -> cmarfa = cod;

 /* citeste cod expeditor */
 if(pcit_int_lim("cod expeditor: ", 0, 9999, &icod) == 0)
  {
   printf(er);
   return 0;
  }
 p -> exp = icod;

 /* citeste cod destinatar */
 if(pcit_int_lim("cod destinatar: ", 0, 9999, &icod) == 0)
  {
   printf(er);
   return 0;
  }
 p -> dest = icod;

 return 1;
} /* sfarsit incnod */
```

Observatie: Daca in aceasta functie se modifica introducerea datelor despre expeditor si destinatar in maniera celor referitoare la codul marfii si al vagonului, se simplifica functia? Ce se intampla cu fisierele scrise la punctele 1si 2 ? Concluzii?

4. Sa se scrie o functie

void elibnod(TNOD *p)

care elibereaza nodul din stiva spre care pointeaza un pointer p.

5. Sa se scrie o functie PUSH:

TNOD *iniprim()

care insereaza nodul curent inaintea primului nod al listei. Functia apeleaza functiile anterior definite care ii sunt necesare.

O posibila solutie:

```
TNOD *iniprim()  /* - PUSH -   insereaza nodul curent inaintea primului nod al listei */
{
 extern TNOD *prim, *ultim;
 TNOD *p;
 int n;
```

```
 n = sizeof(TNOD);
 if(((p = (TNOD *)malloc(n)) != 0) && (incnod(p) == 1))
   {
    if(prim == 0)
            {
             prim = ultim = p;
             p -> urm = 0;
            }
    else
            {
             p -> urm = prim;
             prim = p;
            }
    return p;
   }
 if(p == 0)
   {
    printf("memorie insuficienta\n");
    exit(1);
   }
 elibnod(p);
 return 0;
} /* sfarsit iniprim */
```

6. Sa se scrie o functie POP:

void spn()

care sterge primul nod din lista.

O posibila solutie:

```
void spn() /* POP - sterge primul nod din lista */
{
 extern TNOD *prim, *ultim;
 TNOD *p;

 if(prim == 0)
            return ;
 p = prim;
 prim = prim -> urm;
 elibnod(p);
 if(prim == 0)
            ultim = 0;
} /* sfarsit spn */
```

7. Sa se scrie un program care creaza o stiva, apeland functia iniprim, pana cand aceasta returneaza valoarea zero, apoi listea-za inventarul vagoanelor in ordinea inversa citirii lor. Se va introduce modelul structurii definite la inceputul DESFASURARII LUCRARII. Se vor declara variabilele globale prim si ultim ca pointeri de tip TNOD.

LABORATOR 2

RASPUNSURI:

3. Functia modificata devine:

```
int incnod(TNOD *p)
/* incarca un nod cu datele despre vagoane */
{
 char t[255];
 char er[] = "S-a tastat EOF in pozitie rea\n";
 long cod;
 int icod;

 /* citeste cod vagon */
 for( ; ; )
 {
 printf("cod vagon: ");
 if(gets(t) == 0)
          return -1; /* nu mai sunt date */
 if(sscanf(t, "%ld", &cod) == 1 && cod >= 0 && cod <= 999999999)
          break;
 printf("cod vagon eronat\n");
 }
 p -> cvag = cod;

 /* citeste cod marfa */
 for( ; ; )
 {
 printf("cod marfa: ");
 if(gets(t) == 0)
   {
    printf(er);
    return 0;
   }
 if(sscanf(t, "%ld", &cod) == 1 && cod >= 0 && cod <= 999999999)
   break;
 printf("cod marfa eronat\n");
 }
 p -> cmarfa = cod;

 /* citeste cod expeditor */
 for( ; ; )
 {
 printf("cod expeditor: ");
 if(gets(t) == 0)
          return -1; /* nu mai sunt date */
 if(sscanf(t, "%ld", &cod) == 1 && cod >= 0 && cod <= 9999)
          break;
 printf("cod expeditor eronat\n");
 }
 p -> exp = cod;
 /* citeste cod destinatar */
 for( ; ; )
 {
 printf("cod destinatar: ");
 if(gets(t) == 0)
```

```
            return -1; /* nu mai sunt date */
 if(sscanf(t, "%ld", &cod) == 1 && cod >= 0 && cod <= 9999)
            break;
 printf("cod destinatar eronat\n");
 }
 p -> dest = cod;

 return 1;
} /* sfarsit incnod */
```

In acest caz nu mai sunt apelate functiile de la punctele 1 si 2. S-a realizat o simplificare majora in implementarea programului?

7. Fisierul program este:

```
# include <stdio.h>
# include <conio.h>
# include <alloc.h>
# include <stdlib.h>

typedef struct tnod
                        {
                         long cvag;
                         long cmarfa;
                         int exp;
                         int dest;
                         struct tnod *urm;
                        } TNOD;

# include "pcit_int.cpp" /* functia de la punctul 1 */
# include "p_i_l.cpp"    /* functia de la punctul 2 */
# include "incnod.cpp"   /* functia de la punctul 3 */
# include "elibnod.cpp"
# include "iniprim.cpp"
# include "spn.cpp"

TNOD *prim, *ultim;

main() /* listeaza inventarul vagoanelor in ordinea inversa citirii lor */
{
 prim = ultim = 0; /* la inceput lista este vida */

 /* se creeaza stiva apeland iniprim pana cand aceasta returneaza valoarea zero */
 while(iniprim() != 0)
            ;
 /* - se listeaza elementul din varful stivei si apoi se sterge din stiva;
    - se repeta pana cand stiva devine vida. */
 while(prim != 0)
 {
 printf("\ncod vagon: %ld\tcontinut: %ld\n", prim -> cvag, prim -> cmarfa);
 printf("expeditor: %d\tdestinatar: %d\n", prim -> exp, prim -> dest);
 spn();
 }
 getch();
} /* sfarsit main */
```

Obs.: In cazul modificarii functiei de la punctul 3, nu mai trebuie incluse primele doua fisiere (de la punctele 1 si 2).