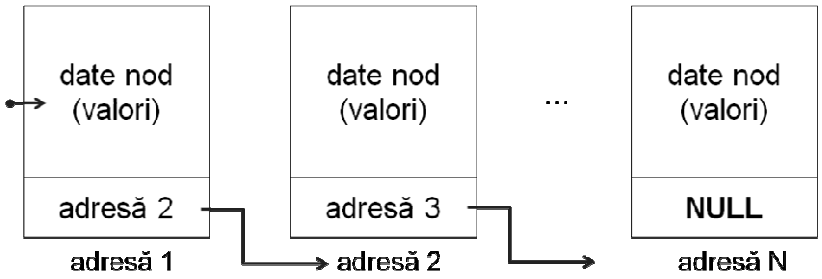
Laborator 3 – Liste

3.1 Probleme rezolvate

O listă înlănțuită de date este o mulțime dinamică de structuri recursive de același tip, pentru care sunt definite una sau mai multe relații de ordine cu ajutorul unor pointeri din compunerea structurilor respective. Elementele unei liste se numesc noduri. Acestea reprezintă o colecție de variabile ce stochează practic informația din listă și asigură în același timp relaționarea elementelor listei. Numărul nodurilor într-o listă este variabil. La început, lista este o mulțime vidă, iar pe parcursul folosirii acesteia se pot adăuga elemente noi sau elimina în funcție de aplicație.



Primul nod din listă nu este relaționat anterior cu nici un alt nod, reprezentând practic începutul listei. Este relaționat indirect cu celelate noduri, stocarea întregii liste realizându-se prin el. Ultimul nod marchează sfârșitul listei, nefiind relaționat următor cu nici un alt nod. Operațiile uzuale ce se realizează cu o listă sunt:

- crearea listei;
- accesul la un nod oarecare al listei;
- inserarea unui nod în listă;
- ștergerea unui nod din listă;
- ștergerea listei.

P3.1 Să se realizeze un program ce permite citirea de la tastatură a unui text neformatat. Acesta va fi introdus sub forma unei liste de cuvinte. Programul permite afișarea pe ecran a unui meniu cu următoarele operații posibile:

- [1] Introducerea numărului de cuvinte dorit;
- [2] Popularea listei;
- [3] Afișarea cuvintelor din listă;
- [0] Ieșire din program.

Fiecare opțiune din meniu va fi implementată folosind funcții.

Rezolvare:

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

/* definire structura stocare date in lista */
struct NOD
{
 char cuvant[20];
 struct NOD *NOD_urmator;
};

struct NOD *lista_cuvinte=NULL;

/* functie creare a unui nod */
struct NOD *creare_nod(int i)
{
 struct NOD *nod;

 /* alocare memorie nod*/
 nod=(struct NOD *)malloc(sizeof(struct NOD));

 if (nod==NULL)
 {
  printf("Eroare: memoria nu a putut fi alocata!\n");
  return NULL;
 }

 /* citire valori nod */
 printf("%d: ", i);
 scanf("%s", nod->cuvant);

 nod->NOD_urmator=NULL;

 return nod;
}

/* functie creare lista noua */
struct NOD *creare_lista()
{
 struct NOD *prim;
 prim=creare_nod(1);
 return prim;
}
```

```
/* functie populare lista cu n cuvinte */
void populare_lista(struct NOD *prim, int n)
{
 int i;
 struct NOD *nod_nou, *nod_curent;

 /* verificare daca lista este goala */
 if (prim==NULL)
 {
  printf("Atentie: lista este goala.\n");
  return;
 }

 /* nodul curent va pargurge lista */
 nod_curent=prim;
 for (i=2; i<=n; i++)
 {
  nod_nou=creare_nod(i);
  if (nod_nou==NULL)
   return;

  /* inserare nod in lista si pozitionare pe acesta */
  nod_curent->NOD_urmator=nod_nou;
  nod_curent=nod_nou;
 }
}

/* functie afisare lista cuvinte */
void afisare_lista(struct NOD *prim)
{
 int i=0;
 struct NOD *nod_curent;

 if (prim==NULL)
 {
  printf("Atentie: lista este goala.\n");
  return;
 }

 nod_curent=prim;

 while(nod_curent!=NULL)
 {
  /* afisare valoare curenta si pozitionare nod urmator */
  i++;
```

```
  printf("%d: %s\n", i, nod_curent->cuvant);
  nod_curent=nod_curent->NOD_urmator;
 }
}

int main()
{
 int operatie, n=0;
 char cuvant[20];

 printf("MENIU:\n");
 printf("[1] Introducerea numarului de cuvinte dorit \n");
 printf("[2] Populatea listei\n");
 printf("[3] Afisarea cuvintelor din lista\n");
 printf("[0] Iesire din program\n");

 do
 {
  printf("\nIntroduceti operatie: ");
  scanf("%d", &operatie);

  if (operatie==1)
  {
   printf("\nIntroducerea unui numar de cuvinte specificat in lista: ");
   do
   {
    scanf("%d", &n);
    if ((n<0) || (n>10))
     printf("\nNumarul trebuie sa fie cuprins in intervalul [0, 10].
                Reintroduceti: ");
   } while((n<0) || (n>10));
   printf("Numar salvat.\n");
  }
  else if (operatie==2)
  {
   printf("\nPopularea listei\n");
   if (n==0)
    printf("Eroare: numarul de cuvinte este 0");
   else
   {
    lista_cuvinte=creare_lista();
    populare_lista(lista_cuvinte, n);
   }
  }
  else if (operatie==3)
```

```
  {
   printf("\nAfisarea cuvintelor din lista\n");
   afisare_lista(lista_cuvinte);
  }
  else if (operatie==0)
  {
   printf("\nIesire din program\n");
   break;
  }
  else
   printf("\nNumar invalid.\n");
 } while(1);

 system("PAUSE");
 return 0;
}
```

Programul va afişa pe ecran, după compilare şi execuţie următoarele:

```
MENIU:
[1] Introducerea numarului de cuvinte dorit
[2] Populareа listei
[3] Afisarea cuvintelor din lista
[0] Iesire din program

Introduceti operatie: 1

Introducerea unui numar de cuvinte specificat in lista: 2
Numar salvat.

Introduceti operatie: 2

Populareа listei
1: cuvant2
2: cuvant3

Introduceti operatie: 3

Afisarea cuvintelor din lista
1: cuvant2
2: cuvant3

Introduceti operatie: 0

Iesire din program
```

Discuţie:

- Acest program pune în evidenţă folosirea listelor pentru a stoca şi manipula cuvintele unui text neformatat. Fiecare cuvânt reprezintă un nod al listei, fiind de tipul `struct NOD`. Nodurile relaționează între ele prin intermediul pointerului `struct NOD *NOD_urmator`, care stochează adresa de memorie a nodului următor din listă;
- Funcția `struct NOD *creare_nod(int i)` este o funcție generală, prin intermediul căreia se creează și se alocă memorie pentru un nod nou. Conținutul acestuia este inițializat cu informația citită de la tastatură, iar locația nodului următor este inițializată cu NULL, deoarece în prima fază acest nod nu este inserat în listă. Adresa nodului nou creat este returnată printr-un pointer la structura NOD;
- Funcția `struct NOD *creare_lista()` creează primul nod al listei prin apelarea funcției de creare a unui nod, a cărui adresă de memorie o returnează ca parametru de ieșire, printr-un pointer la o structură NOD;
- Inserarea mai multor noduri în listă se face cu ajutorul funcției `void populare_lista(struct NOD *prim, int n)`. Aceasta primește ca parametru de intrare adresa de memorie a primului nod (începutul listei). În primă fază se verifică dacă lista a fost creată, după care, în caz afirmativ se permite adăugarea unui număr specificat de noduri, prin apelarea succesivă a funcției `creare_nod()`. Lista este parcursă progresiv, pe măsură ce sunt adăugate noduri, printr-un pointer temporar, la nodul curent `nod_curent`. Inserarea nodului nou în listă se realizează prin intermediul pointerului `NOD_urmator`, care preia adresa nodului nou creat: `nod_curent->NOD_urmator=nod_nou`. Noul nod curent devine nodul nou adăugat și procesul se repetă;
- Afișarea informației din cadrul fiecărui nod al listei se realizează cu ajutorul funcției `void afisare_lista(struct NOD *prim)`, care pornește de la primul nod și parcurge întreaga listă, afișând câmpul "cuvant" corespunzător fiecărui nod. Parcurgerea listei se realizează cu ajutorul structurii repetitive `while`, iar condiția de oprire este ca nodul curent să fie `NULL`: `nod_curent==NULL` (cu alte cuvinte, am ajuns la ultimul nod al listei care nu mai indică un alt nod);
- Meniul programului este implementat în funcția `main()`, prin intermediul căruia utilizatorul poate apela funcțiile definite mai sus.

P3.2 Să se modifice programul anterior prin completarea meniului cu următoarele operații noi:

- [1] Introducerea numărului de cuvinte dorit;
- [2] Popularea listei;
- [3] Afișarea cuvintelor din listă;
- [4] Adăugarea unui cuvânt la începutul listei;
- [5] Adăugarea unui cuvânt la finalul listei;
- [6] Adăugarea unui cuvânt în listă, pe o poziție specificată de utilizator;
- [0] Ieșire din program.

Fiecare opțiune din meniu va fi implementată folosind funcții.

Rezolvare:

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

...

/* functie adaugare cuvant la inceputul listei */
struct NOD *adaugare_nod_inceput_lista(struct NOD *prim)
{
 struct NOD *nod_nou;

 if (prim==NULL)
 {
  printf("Atentie: lista este goala.\n");
  return NULL;
 }

 nod_nou=creare_nod(1);

 /* noul nod devine primul nod in lista */
 nod_nou->NOD_urmator=prim;

 printf("Cuvantul a fost adaugat.\n");
 return nod_nou;
}

/* functie adaugare cuvant la sfarsitul listei */
struct NOD *adaugare_nod_sfarsit_lista(struct NOD *prim)
{
 int i=0;
 struct NOD *nod_curent, *nod_nou;

 if (prim==NULL)
```

```
 {
  printf("Atentie: lista este goala.\n");
  return NULL;
 }

 /*parcurge lista element cu element pentru a ajunge la ultimul nod*/
 nod_curent=prim;

 while (nod_curent!=NULL)
 {
  i++;
  if (nod_curent->NOD_urmator==NULL)
  {
   /* creare si inserare nod nou in lista */
   nod_nou=creare_nod(i+1);
   nod_curent->NOD_urmator=nod_nou;

   printf("Cuvantul a fost adaugat.\n");
   return prim;
  }
  nod_curent=nod_curent->NOD_urmator;
 }
}

/* functie adaugare cuvant in lista pe o pozitie */
struct NOD *adaugare_nod_lista(struct NOD *prim)
{
 int i=0, poz;
 struct NOD *nod_curent, *nod_nou;

 if (prim==NULL)
 {
  printf("Atentie: lista este goala.\n");
  return NULL;
 }

 printf("Introduceti pozitia: ");
 scanf("%d", &poz);

 if (poz==1)
  return adaugare_nod_inceput_lista(prim);

 /* parcurgere lista cu for */
 for (nod_curent=prim; nod_curent!=NULL;
          nod_curent=nod_curent->NOD_urmator)
```

```
 {
  i++;
  if (i==(poz-1)) // pozitia inainte de cea dorita
   break;         // pozitionare pe nodul de la aceasta pozitie
 }

 /* adaugare nod dupa cel curent*/
 if (nod_curent->NOD_urmator==NULL)
  // ultimul nod
  return adaugare_nod_sfarsit_lista(prim);
 else
 {
  nod_nou=creare_nod(poz);
  nod_nou->NOD_urmator=nod_curent->NOD_urmator;
  nod_curent->NOD_urmator=nod_nou;

  printf("Cuvantul a fost adaugat.\n");
  return prim;
 }
}

int main()
{
 int operatie, n=0;
 char cuvant[20];

 printf("MENIU:\n");
 printf("[1] Introducerea numarului de cuvinte dorit \n");
 printf("[2] Populatarea listei\n");
 printf("[3] Afisarea cuvintelor din lista\n");
 printf("[4] Adaugarea unui cuvant la inceputul listei\n");
 printf("[5] Adaugarea unui cuvant la finalul listei\n");
 printf("[6] Adaugarea unui cuvant in lista, pe o pozitie specificata de
              utilizator\n");
 printf("[0] Iesire din program\n");

 do
 {
  printf("\nIntroduceti operatie: ");
  scanf("%d", &operatie);

  if (operatie==1)
  {
   printf("\nIntroducerea unui numar de cuvinte specificat in lista: ");
   do
```

```
 {
  scanf("%d", &n);
  if ((n<0) || (n>10))
   printf("\nNumarul trebuie sa fie cuprins in intervalul [0, 10].
              Reintroduceti: ");
 } while((n<0) || (n>10));
 printf("Numar salvat.\n");
}
else if (operatie==2)
{
 printf("\nPopularea listei\n");
 if (n==0)
  printf("Eroare: numarul de cuvinte este 0");
 else
 {
  lista_cuvinte=creare_lista();
  populare_lista(lista_cuvinte, n);
 }
}
else if (operatie==3)
{
 printf("\nAfisarea cuvintelor din lista\n");
 afisare_lista(lista_cuvinte);
}
else if (operatie==4)
{
 printf("\nAdaugarea unui cuvant la inceputul listei\n");
 lista_cuvinte=adaugare_nod_inceput_lista(lista_cuvinte);
}
else if (operatie==5)
{
 printf("\nAdaugarea unui cuvant la finalul listei\n");
 lista_cuvinte=adaugare_nod_sfarsit_lista(lista_cuvinte);
}
else if (operatie==6)
{
 printf("\nAdaugarea unui cuvant in lista, pe o pozitie
            specificata de utilizator\n");
 lista_cuvinte=adaugare_nod_lista(lista_cuvinte);
}
else if (operatie==0)
{
 printf("\nIesire din program\n");
 break;
}
```

```
  else
   printf("\nNumar invalid.\n");
 } while(1);

 system("PAUSE");
 return 0;
}
```

Programul va afişa pe ecran, după compilare şi execuţie următoarele:

```
MENIU:
[1] Introducerea numarului de cuvinte dorit
[2] Populareа listei
[3] Afisarea cuvintelor din lista
[4] Adaugarea unui cuvant la inceputul listei
[5] Adaugarea unui cuvant la finalul listei
[6] Adaugarea unui cuvant in lista, pe o pozitie specificata de utilizator
[0] Iesire din program

Introduceti operatie: 1

Introducerea unui numar de cuvinte specificat in lista: 2
Numar salvat.

Introduceti operatie: 2

Populareа listei
1: cuvant2
2: cuvant3

Introduceti operatie: 3

Afisarea cuvintelor din lista
1: cuvant2
2: cuvant3

Introduceti operatie: 4

Adaugarea unui cuvant la inceputul listei
1: cuvant1
Cuvantul a fost adaugat.

Introduceti operatie: 3

Afisarea cuvintelor din lista
1: cuvant1
2: cuvant2
3: cuvant3

Introduceti operatie: 5
```

```
Adaugarea unui cuvant la finalul listei
4: cuvant5
Cuvantul a fost adaugat.

Introduceti operatie: 3

Afisarea cuvintelor din lista
1: cuvant1
2: cuvant2
3: cuvant3
4: cuvant5

Introduceti operatie: 6

Adaugarea unui cuvant in lista, pe o pozitie specificata de utilizator
Introduceti pozitia: 4
4: cuvant4
Cuvantul a fost adaugat.

Introduceti operatie: 0

Iesire din program
```

Discuţie:

- Funcția `struct NOD *adaugare_nod_inceput_lista(struct NOD *prim)` permite adăugarea unui nod la începutul listei, prin apelarea funcției `creare_nod()`. Fiind noul prim nod al listei, acesta este relaționat următor cu primul nod al listei. Astfel, nodul adăugat devine primul nod din listă: `nod_nou->NOD_urmator=prim`. La final, funcția returnează adresa de memorie a nodului nou creat care va trebui preluată în programul principal în pointerul ce indică primul element din lista (`prim`);

- Funcția `struct NOD *adaugare_nod_sfarsit_lista(struct NOD *prim)` permite adăugarea unui nod la sfârșitul listei. Nodul este creat cu ajutorul funcției `creare_nod()` și relaționat anterior cu ultimul nod al listei. Pentru a accesa ultimul nod este necesară parcurgerea listei element cu element, pornind de la primul nod, lucru ce se realizează prin intermediul unui pointer temporar `nod_curent`. Având în vedere faptul că nodurile listei nu memorează adresele nodurilor anterioare, pentru a putea realiza legătura cu nodul dinainte trebuie să ne poziționăm pe nodul anterior ultimului nod, și nu exact pe ultimul nod, adică nodul pentru care `nod_curent->NOD_urmator==NULL`. Inserarea la final presupune astfel crearea legăturii cu nodul nou astfel: `nod_curent->NOD_urmator=nod_nou`. La final, funcția returnează adresa de memorie a primului nod;

- Adăugarea unui nod pe o poziție specificată de utilizator se realizează cu ajutorul funcției `struct NOD *adaugare_nod_lista(struct NOD *prim)`. Aceasta are un

caracter mai general și poate apela funcțiile definite anterior, de adăugare a unui nod la începutul sau la sfârșitul listei, în funcție de poziția specificată. Dacă poziția specificată nu se încadrează în aceste cazuri particulare, atunci se parcurge lista până la elementul anterior acestei poziții (vezi observația anterioară), folosind structura repetitivă `for`. Parcurgerea în buclă se oprește cu funcția `break` pentru a se poziționa exact pe nodul de dinainte de poziția dorită, nod accesat prin pointerul `nod_curent`. De această dată, inserarea unui nod nou presupune relaționarea atât cu nodul anterior cât și cu nodul următor. Nodul nou creat este inserat după `nod_curent`, pe poziția specificată. Acesta este relaționat următor cu nodul următor nodului curent: `nod_nou->NOD_urmator=nod_curent->NOD_urmator` și anterior cu nodul curent: `nod_curent->NOD_urmator=nod_nou`. La final, se returnează adresa de memorie a primului nod, pentru a indica începutul listei;

- Meniul programului este implementat în funcția `main()`, prin intermediul căruia utilizatorul poate apela funcțiile definite mai sus.

P3.3 Să se modifice programul anterior prin completarea meniului cu următoarele operații noi:

- [1] Introducerea numărului de cuvinte dorit;
- [2] Popularea listei;
- [3] Afișarea cuvintelor din listă;
- [4] Adăugarea unui cuvânt la începutul listei;
- [5] Adăugarea unui cuvânt la finalul listei;
- [6] Adăugarea unui cuvânt în listă, pe o poziție specificată de utilizator;
- [7] Ștergerea primului cuvânt din listă;
- [8] Ștergerea ultimului cuvânt din listă;
- [9] Ștergerea unui cuvânt din listă, aflat pe o poziție specificată de utilizator;
- [10] Ștergerea listei;
- [0] Ieșire din program.

Fiecare opțiune din meniu va fi implementată folosind funcții.

Rezolvare:

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

...

/* functie stergere primul cuvant din lista */
struct NOD *stergere_primul_nod_lista(struct NOD *prim)
{
 struct NOD *nod_sters;
```

```
 if (prim==NULL)
 {
  printf("Atentie: lista este goala.\n");
  return NULL;
 }

 /* verificare daca este singurul nod */
 if (prim->NOD_urmator==NULL)
 {
  printf("Atentie: lista are un singur nod si va fi stearsa complet.\n");
  free(prim);

  printf("Cuvantul a fost sters.\n");
  return NULL;
 }
 else
 {
  nod_sters=prim;     // nod_sters indica spre primul nod
  prim=prim->NOD_urmator;  // primul nod devine urmatorul
  free(nod_sters);         // se elibereaza memoria

  printf("Cuvantul a fost sters.\n");
  return prim;
 }
}

/* functie stergere ultimul cuvant din lista */
struct NOD *stergere_ultimul_nod_lista(struct NOD *prim)
{
 struct NOD *nod_curent, *nod_sters;

 if (prim==NULL)
 {
  printf("Atentie: lista este goala.\n");
  return NULL;
 }

 /* verificare daca este singurul nod */
 if (prim->NOD_urmator==NULL)
 {
  printf("Atentie: lista are un singur cuvant si va
           fi stearsa complet.\n");
  free(prim);

  printf("Cuvantul a fost sters.\n");
```

```
  return NULL;
 }

 /* parcurgere lista cu for pana la penultimul element */
 for (nod_curent=prim; nod_curent->NOD_urmator->NOD_urmator!=NULL;
        nod_curent=nod_curent->NOD_urmator);

 nod_sters=nod_curent->NOD_urmator;   // nod_sters indica catre ultimul nod
 nod_curent->NOD_urmator=NULL;  // nodul curent devine ultimul
 free(nod_sters);

 printf("Cuvantul a fost sters.\n");
 return prim;
}

/* functie stergere cuvant din lista dupa valoare */
struct NOD *stergere_nod_lista(struct NOD *prim)
{
 int i=0, poz;
 struct NOD *nod_curent, *nod_sters;

 if (prim==NULL)
 {
  printf("Atentie: lista este goala.\n");
  return NULL;
 }

 printf("Introduceti pozitia: ");
 scanf("%d", &poz);

 /* daca este primul nod, se apeleaza functia anterioara */
 if (poz==1)
  return stergere_primul_nod_lista(prim);

 /* parcurge lista pana se gaseste elementul anterior cuvantului cautat */
 for (nod_curent=prim; nod_curent!=NULL;
        nod_curent=nod_curent->NOD_urmator)
 {
  i++;
  if (i==poz-1)       // pozitia inainte de cea dorita
   break;       // memoreaza nodul de la aceasta pozitie
 }

 /* daca este utimul nod, se apeleaza functia anterioara */
 if (nod_curent->NOD_urmator==NULL)
```

```
  return stergere_ultimul_nod_lista(prim);
 else
 {
  // nod_sters indica catre nodul urmator care va fi sters
  nod_sters=nod_curent->NOD_urmator;
  // nodul curent indica catre nodul urmator
  nod_curent->NOD_urmator=nod_curent->NOD_urmator->NOD_urmator;
  free(nod_sters);    // eliberare memorie nod

  printf("Cuvantul a fost sters.\n");
  return prim;
 }
}

/* functie stergere lista */
struct NOD *stergere_lista(struct NOD *prim)
{
 struct NOD *nod_curent;

 if (prim==NULL)
  return NULL;

 /* parcurgere lista si stergere elemente */
 while (prim!=NULL)
 {
  nod_curent=prim;    // nod_curent indica spre primul nod
  prim=prim->NOD_urmator;  // primul nod devine urmatorul
  printf("Stergere: %s\n", nod_curent->cuvant);
  free(nod_curent);   // se elibereaza memoria
 }

 printf("Lista a fost stearsa.\n");
 return NULL;
}

int main()
{
 int operatie, n=0;
 char cuvant[20];

 printf("MENIU:\n");
 printf("[1] Introducerea numarului de cuvinte dorit \n");
 printf("[2] Populatea listei\n");
 printf("[3] Afisarea cuvintelor din lista\n");
 printf("[4] Adaugarea unui cuvant la inceputul listei\n");
```

```
printf("[5] Adaugarea unui cuvant la finalul listei\n");
printf("[6] Adaugarea unui cuvant in lista, pe o pozitie specificata de
             utilizator\n");
printf("[7] Stergerea primului cuvant din lista\n");
printf("[8] Stergerea ultimului cuvant din lista\n");
printf("[9] Stergerea unui cuvant din lista, aflat pe o pozitie
             specificata de utilizator\n");
printf("[10] Stergerea listei\n");
printf("[0] Iesire din program\n");

do
{
 printf("\nIntroduceti operatie: ");
 scanf("%d", &operatie);

 if (operatie==1)
 {
  printf("\nIntroducerea unui numar de cuvinte specificat in lista: ");
  do
  {
   scanf("%d", &n);
   if ((n<0) || (n>10))
    printf("\nNumarul trebuie sa fie cuprins in intervalul [0, 10].
                Reintroduceti: ");
  } while((n<0) || (n>10));
  printf("Numar salvat.\n");
 }
 else if (operatie==2)
 {
  printf("\nPopularea listei\n");
  if (n==0)
   printf("Eroare: numarul de cuvinte este 0");
  else
  {
   stergere_lista(lista_cuvinte);
   lista_cuvinte=creare_lista();
   populare_lista(lista_cuvinte, n);
  }
 }
 else if (operatie==3)
 {
  printf("\nAfisarea cuvintelor din lista\n");
  afisare_lista(lista_cuvinte);
 }
 else if (operatie==4)
```

```
  {
   printf("\nAdaugarea unui cuvant la inceputul listei\n");
   lista_cuvinte=adaugare_nod_inceput_lista(lista_cuvinte);
  }
  else if (operatie==5)
  {
   printf("\nAdaugarea unui cuvant la finalul listei\n");
   lista_cuvinte=adaugare_nod_sfarsit_lista(lista_cuvinte);
  }
  else if (operatie==6)
  {
   printf("\nAdaugarea unui cuvant in lista, pe o pozitie
              specificata de utilizator\n");
   lista_cuvinte=adaugare_nod_lista(lista_cuvinte);
  }
  else if (operatie==7)
  {
   printf("\nStergerea primului cuvant din lista\n");
   lista_cuvinte=stergere_primul_nod_lista(lista_cuvinte);
  }
  else if (operatie==8)
  {
   printf("\nStergerea ultimului cuvant din lista\n");
   stergere_ultimul_nod_lista(lista_cuvinte);
  }
  else if (operatie==9)
  {
   printf("\nStergerea unui cuvant din lista, aflat pe
              o pozitie specificata de utilizator\n");
   lista_cuvinte=stergere_nod_lista(lista_cuvinte);
  }
  else if (operatie==10)
  {
   printf("\nStergerea listei\n");
   lista_cuvinte=stergere_lista(lista_cuvinte);
  }
  else if (operatie==0)
  {
   printf("\nIesire din program\n");
   lista_cuvinte=stergere_lista(lista_cuvinte);
   break;
  }
  else
   printf("\nNumar invalid.\n");
 } while(1);
```

```
 system("PAUSE");
 return 0;
}
```

Programul va afişa pe ecran, după compilare şi execuţie următoarele:

```
MENIU:
[1] Introducerea numarului de cuvinte dorit
[2] Populatea listei
[3] Afisarea cuvintelor din lista
[4] Adaugarea unui cuvant la inceputul listei
[5] Adaugarea unui cuvant la finalul listei
[6] Adaugarea unui cuvant in lista, pe o pozitie specificata de utilizator
[7] Stergerea primului cuvant din lista
[8] Stergerea ultimului cuvant din lista
[9] Stergerea unui cuvant din lista, aflat pe o pozitie specificata de
utilizator
[10] Stergerea listei
[0] Iesire din program

Introduceti operatie: 1

Introducerea unui numar de cuvinte specificat in lista: 2
Numar salvat.

Introduceti operatie: 2

Populatea listei
1: cuvant2
2: cuvant3

Introduceti operatie: 3

Afisarea cuvintelor din lista
1: cuvant2
2: cuvant3

Introduceti operatie: 4

Adaugarea unui cuvant la inceputul listei
1: cuvant1
Cuvantul a fost adaugat.

Introduceti operatie: 3

Afisarea cuvintelor din lista
1: cuvant1
2: cuvant2
3: cuvant3
```

```
Introduceti operatie: 5

Adaugarea unui cuvant la finalul listei
4: cuvant5
Cuvantul a fost adaugat.

Introduceti operatie: 3

Afisarea cuvintelor din lista
1: cuvant1
2: cuvant2
3: cuvant3
4: cuvant5

Introduceti operatie: 6

Adaugarea unui cuvant in lista, pe o pozitie specificata de utilizator
Introduceti pozitia: 4
4: cuvant4
Cuvantul a fost adaugat.

Introduceti operatie: 3

Afisarea cuvintelor din lista
1: cuvant1
2: cuvant2
3: cuvant3
4: cuvant4
5: cuvant5

Introduceti operatie: 7

Stergerea primului cuvant din lista
Cuvantul a fost sters.

Introduceti operatie: 3

Afisarea cuvintelor din lista
1: cuvant2
2: cuvant3
3: cuvant4
4: cuvant5

Introduceti operatie: 8

Stergerea ultimului cuvant din lista
Cuvantul a fost sters.

Introduceti operatie: 3

Afisarea cuvintelor din lista
```

```
1: cuvant2
2: cuvant3
3: cuvant4

Introduceti operatie: 9

Stergerea unui cuvant din lista, aflat pe o pozitie specificata de
utilizator
Introduceti pozitia: 2
Cuvantul a fost sters.

Introduceti operatie: 3

Afisarea cuvintelor din lista
1: cuvant2
2: cuvant4

Introduceti operatie: 10

Stergerea listei
Stergere: cuvant2
Stergere: cuvant4
Lista a fost stearsa.

Introduceti operatie: 3

Afisarea cuvintelor din lista
Atentie: lista este goala.

Introduceti operatie: 0

Iesire din program
```

Discuţie:

- Funcția `struct NOD *stergere_primul_nod_lista(struct NOD *prim)` permite ștergerea primului nod din listă, după ce în prealabil s-a verificat că aceasta nu este vidă. Dacă lista are un singur element, acesta va fi șters, iar lista va deveni vidă, returnându-se astfel NULL. În cazul în care lista are mai mult de un element, se memorează adresa primului nod într-un pointer, `nod_sters`, prin `nod_sters=prim`, apoi, primul nod al listei devine nodul următor prin `prim=prim->NOD_urmator` și în final se șterge primul nod inițial (indicat acum prin pointerul `nod_sters`) care acum nu mai este practic parte a listei: `free(nod_sters)`. În final se returnează adresa noului prim nod al listei, care indică acum începutul listei;

- Funcția `struct NOD *stergere_ultimul_nod_lista(struct NOD *prim)` permite ștergerea ultimului nod din listă. Procedeul de accesare a ultimului nod este asemănător ca în cazul funcției `adaugare_nod_sfarsit_lista()`, în acest caz

optându-se pentru structura repetitivă `for`. Parcurgerea listei se realizează cu pointerul temporar `nod_curent` până când se ajunge la penultimul element, adică `nod_curent->NOD_urmator->NOD_urmator==NULL`. Nodul care trebuie șters este preluat în pointerul `nod_sters=nod_curent->NOD_urmator` și eliberat cu funcția `free()`, `free(nod_sters)`. Ceea ce mai rămâne de realizat este definirea ultimului element care să nu mai indice către nici un alt nod, și anume `nod_curent->NOD_urmator=NULL` (se marchează sfârșitul listei). Funcția returnează la final începutul listei;

- Ștergerea unui nod de pe o poziție specificată de utilizator se realizează cu ajutorul funcției `struct NOD *stergere_nod_lista(struct NOD *prim)`, care este asemănătoare ca implementare cu funcția `adaugare_nod_lista()`. Dacă poziția pecificată de utilizator nu se încadrează în unul din cazurile particulare, început sau sfârșit de listă, atunci se parcurge succesiv lista prin intermediul pointerului temporar `nod_curent` până la nodul anterior nodului specificat. În acest caz nodul care trebuie șters este preluat în pointerul `nod_sters` iar legăturile din listă sunt refăcute astfel încât să sară peste acest nod: `nod_curent->NOD_urmator=nod_curent->NOD_urmator->NOD_urmator`, și `free(nod_sters);`

- Funcția `struct NOD *stergere_lista(struct NOD *prim)` permite stergerea întregii liste, în cazul în care aceasta este nevidă. Se aplică procedeul ștergerii progresive a primului nod al listei, operație ce este iterată până când primul element nu mai există, adică până când lista devine vidă;

- Meniul programului este implementat în funcția `main()`, prin intermediul căruia utilizatorul poate apela funcțiile definite mai sus. Pentru popularea listei, este necesar ca lista să fie creată automat înainte și eventual stearsă în cazul în care aceasta există. Lista se șterge automat la ieșirea din program.

3.2 Probleme propuse

1. Să se modifice corespunzător programul anterior astfel încât să includă în meniu și posibilitatea de ștergere a unui cuvânt din listă prin specificarea acestuia de către utilizator. În cazul în care cuvântul nu este găsit, se va semnala utilizatorului această situație.