STIVE

2.1. MODELUL STIVEI

Stiva este o structură de date în care introducerea (extragerea) de elemente se poate efectua doar într-o (dintr-o) singură poziție, numită ***vârful stivei***, care este de fapt, sfârşitul listei. Stivele sunt cunoscute de asemenea ca liste ***LIFO (Last In First Out)***.

Operaţiile fundamentale care se pot efectua asupra unei stive sunt: introducerea în stivă (***PUSH***) şi extragerea din stivă (***POP***) a celui mai recent element introdus. De asemenea, acest element poate fi accesat printr-o operație care nu presupune şi extragerea sa din stivă (***TOP***). Operațiile ***POP*** şi ***TOP*** asupra unei stive vide sunt considerate erori.

Fig.2.1. Modelul stivei. Introducerea, extragerea şi accesarea primului element

Modelul general al stivei este acela în care doar un singur element este accesibil (acela care se află în vârful stivei).

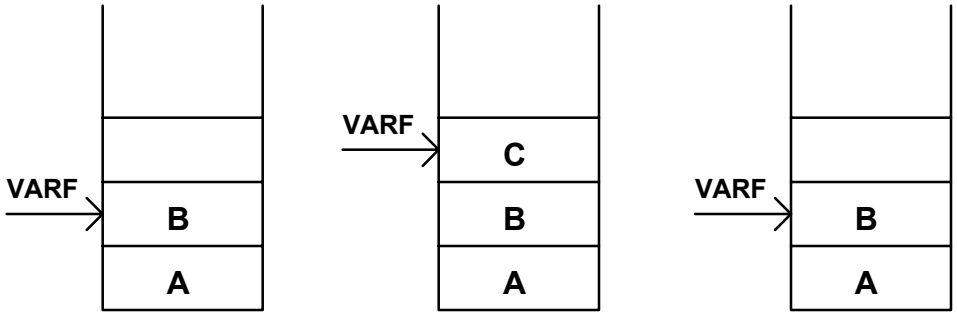

Fig.2.2. Modelul stivei. Accesibilitatea elementului din vârf după operaţiile PUSH şi POP

2.2. IMPLEMENTAREA STIVELOR

Stivele pot fi implementate în două moduri clasice: static şi dinamic. Implementarea dinamică utilizează *pointeri*, iar cea statică, *tablouri*. Implementarea sub formă de tablou are dezavantajul lungimii finite a stivei.

Operaţiile care pot fi efectuate asupra unei stive sunt următoarele:
- crearea stivei;
- introducerea unui element în stivă;
- extragerea unui element din stivă;
- extragerea informației elementului aflat în vârful stivei;
- testul de stivă vidă.

2.2.1. IMPLEMENTAREA DINAMICĂ

Implementarea dinamică utilizează modelul listei înlănțuite. O operație de tip ***PUSH*** realizează introducerea unui element în capul listei, iar o operație de tip ***POP*** determină eliminarea elementului din capul listei. O operație de tip ***TOP*** va accesa elementul din capul listei şi va întoarce informația conținută de acesta.

Crearea unei stive constă în inițializarea vârfului stivei cu valoarea **NULL**. Introducerea elementului următor se va realiza prin:

- alocarea unui spațiu în memorie pentru acesta şi
- atribuirea valorii din vârful stivei câmpului care indică spre elementul următor.

Apoi, se va actualiza vârful stivei.

Exemplu:

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

typedef struct
        {
        int Valoare;
        } InfoT;

typedef struct Nod
        {
        InfoT Date;
        struct Nod *Urm;
        } NodT;

typedef struct
        {
        NodT *Varf;
        }StivaT;
```

```
/* functie creare stiva */

void Creare (StivaT *StivaPtr)
        {
        StivaPtr->Varf=NULL;
        }

/* functie testare stiva vida */

int TestVida (StivaT *StivaPtr)
        {
        return StivaPtr->Varf==NULL;
        }

/* functie introducere in stiva (PUSH) */

void Introducere (StivaT *StivaPtr, InfoT *InfoPtr)
        {
        NodT *Elem;
        Elem= (struct Nod T *) malloc (sizeof(NodT));
        Elem->Date=*InfoPtr;
        Elem->Urm=StivaPtr->Varf;
        StivaPtr->Varf=Elem;
        };

/* functie extragere din stiva (POP) */

int Extragere (StivaT *StivaPtr, InfoT *InfoPtr)
        {
        NodT *Temp;
        if (TestVida(StivaPtr)) return 0;
        else
                {
                *InfoPtr=StivaPtr->Varf->Date;
                Temp=StivaPtr->Varf->Urm;
                free (StivaPtr->Varf);
                StivaPtr->Varf=Temp;
                return 1;
                };
        }


/* functie de returnare a informatiei din varful stivei (TOP) */
        {
        int InfoVarf (StivaT *StivaPtr, InfoT *InfoPtr)
                {
```

```
if (TestVida(StivaPtr)) return 0;
else
        {
        * InfoPtr=StivaPtr->Varf->Date;
        return 1;
        };
}
```

2.2.2. IMPLEMENTAREA STATICĂ

O alternativă la implementarea stivelor cu pointeri este implementarea cu tablouri. Dezavantajul acestei metode constă în declarea apriorică a dimensiunii maxime a stivei.

Exemplu:

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

# define DimMax 100

typedef struct
        {
        int Valoare;
        } InfoT;

typedef struct
        {
        int Varf;
        InfoT Date [DimMax];
        }StivaT;

/* functie creare stiva */

void Creare (StivaT *StivaPtr)
        {
        StivaPtr->Varf=-1;
        }

/* functie testare stiva vida */

int TestVida (StivaT *StivaPtr)
        {
        return StivaPtr->Varf==-1;
        }
/* functie introducere in stiva (PUSH) */
```

```
void Introducere (StivaT *StivaPtr, InfoT *InfoPtr)
        {
        if (StivaPtr->Varf==DimMax-1) return 0;
        else
                {
                StivaPtr->Date[++StivaPtr->Varf]=*InfoPtr;
        return 1;
        };
        };

/* functie extragere din stiva (POP) */

int Extragere (StivaT *StivaPtr, InfoT *InfoPtr)
        {
        if (TestVida (StivaPtr)) return 0;
        else
                {
                * InfoPtr=StivaPtr->Date[StivaPtr->Varf--];
                return 1;
                };
        }

/* functie de returnare a informatiei din varful stivei (TOP) */
        {
        int InfoVarf (StivaT *StivaPtr, InfoT *InfoPtr)
                {
                if (TestVida(StivaPtr)) return 0;
                else
                        {
                        * InfoPtr=StivaPtr->Date[StivaPtr->Varf];
                        return 1;
                        };
                }
```

2.3. APLICAȚIILE STIVELOR

Structurile de tip stivă au aplicații în domeniul sistemelor de calcul, la *apelul subrutinelor, în calculul recursiv, sau în evaluarea unor expresii.*

Într-un program principal poate fi apelată o subrutină S_1, care la rândul său poate apela o subrutină S_2.Pentru revenire în subrutina S_1, după execuția subrutinei S_2, trebuie ca adresa următoarei instrucțiuni din $S_1(Adr_2)$ să fie disponibilă. În mod identic, la

revenirea din subrutina S_1, este necesară cunoaşterea adresei de revenire în programul principal (Adr_1).

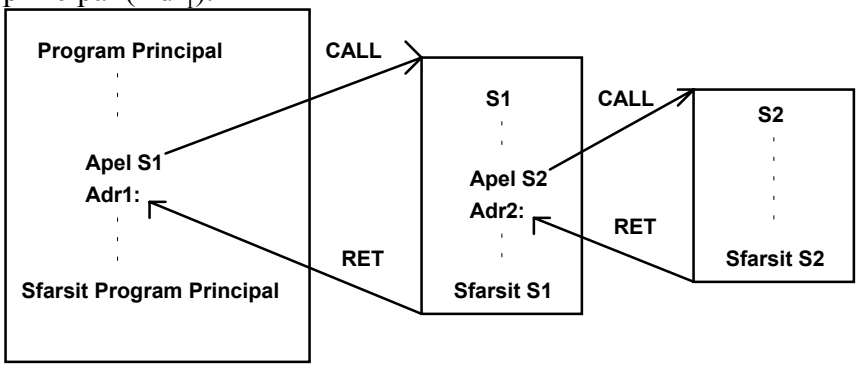

Fig.2.3. Secvenţa apelurilor de subrutine

Pentru ca adresele să fie cunoscute, trebuie memorată mai întâi Adr_1, iar apoi Adr_2. Deci este necesară memorarea într-o structură de tip stivă.

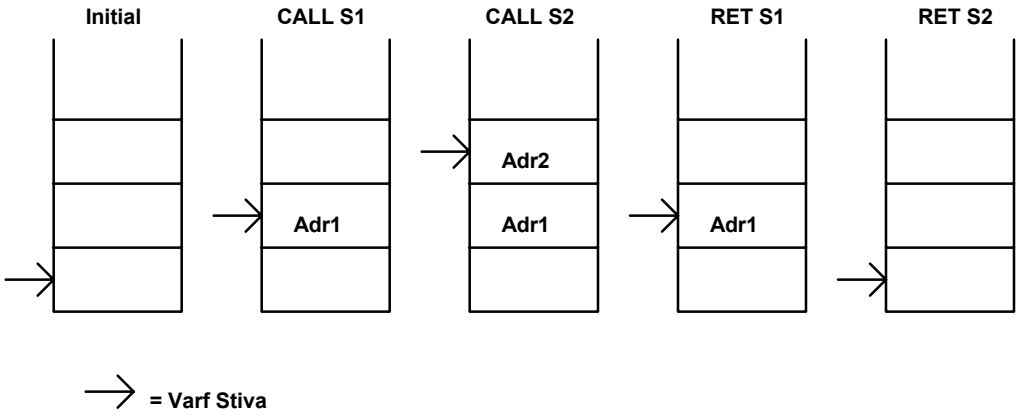

Fig.2.4. Memorarea adreselor de revenire în cazul apelurilor de subrutine

Evaluarea expresiilor aritmetice cu ajutorul stivelor poate fi exemplificată prin studiul conversiei unei expresii infixate într-o expresie postfixată. O expresie infixată este o expresie de forma **a+b*c**, a cărei formă postfixată este **abc*+**. În exemplele următoare se vor folosi expresii care au ca operanzi litere mici, iar ca operatori caracterele **(, *, +,)**. Operatorii au asociate priorităţi în ordinea de mai sus, **(** fiind *cel mai prioritar*, iar **)**, *cel mai puţin prioritar*. O proprietate a unei expresii postfixate este că nu sunt necesare paranteze pentru evaluarea sa.

Exemplu: Expresia infixată: **a+b*c+(d*e+f)*g** are ca echivalent expresia postfixată **abc*+de*f+g*+**.

Conversia se realizează pornind de la un şir de caractere de intrare, care identifică expresia infixată şi construind apoi un şir de caractere de ieşire, care va identifica

expresia postfixată. Şirul de caractere de intrare va fi citit caracter cu caracter, iar atunci când se întâlneşte un *operand*, acesta va fi scris în *şirul de caractere de ieşire*. Dacă este citit un *operator*, acesta va fi introdus într-o *stivă rezervată operatorilor*, după ce în prealabil au fost extraşi din stivă (dacă nu era vidă) şi scrişi în şirul de caractere de ieşire operatorii cu prioritate mai mare sau egală. La terminarea şirului de caractere de intrare se vor extrage toți operatorii din stivă şi se vor scrie în şirul de caractere de ieşire.

Exemplu: a+b*c se transformă în **abc*+**.

La început, se citeşte operandul **a**, apoi operatorul **+**, care va fi introdus în stivă, iar la sfârşit, operandul **b**.

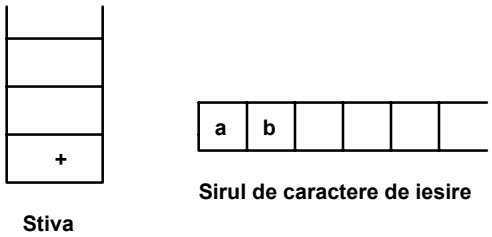

Fig.2.5. Prima etapă a procesului de conversie a expresiei aritmetice

În continuare, se va citi operatorul *****, care va fi introdus în stivă, deoarece operatorul precedent din stivă are o prioritate mai mică. Va fi citit apoi operandul **c**, care va fi scris în şirul de caractere de ieşire (Fig.2.6).

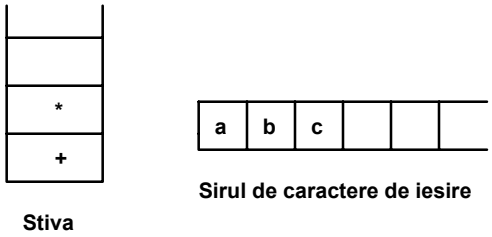

Fig.2.6. A doua etapă a procesului de conversie a expresiei aritmetice.

Şirul de caractere de intrare se termină şi astfel este necesară extragerea din stivă a operatorilor deja introduşi şi scrierea lor în şirul de caractere de ieşire (Fig.2.7).

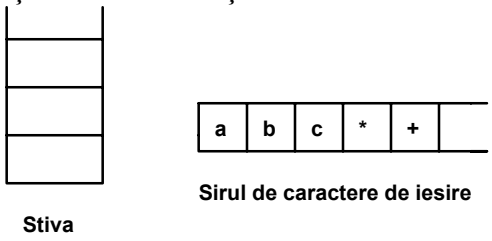

Fig.2.7. Ultima etapă a procesului de conversie a expresiei aritmetice

2.4. DESFĂŞURAREA LUCRĂRII

Să se scrie un program de conversie a unei expresii infixate într-o expresie postfixată, conform următorului algoritm, cunoscând că operanzii sunt reprezentaţi de litere mici (**a, ..., z**), iar operatorii de caracterele **(, *, +,)**, care sunt scrise în ordinea priorităţilor lor. Se va realiza o implementarea dinamică a stivei.

- Se citeşte un şir de caractere de intrare de la tastatură.
- Se citeşte şirul caracter cu caracter şi se execută următoarele operaţii:
  - Caracterul este un operand (**a, ..., z**): - se va scrie în şirul de caractere de ieşire.
  - Caracterul este un operator *****: - se vor extrage din stivă şi se vor scrie în şirul de caractere de ieşire toţi operatorii de prioritate mai mare sau egală cu a sa (*****), până la întâlnirea unui operator **(** şi cât timp stiva nu este vidă. Apoi se va introduce acest operator în stivă.
  - Caracterul este un operator **+**: - se vor extrage din stivă şi se vor scrie în şirul de caractere de ieşire toţi operatorii de prioritate mai mare sau egală cu a sa (***, +**), până la întâlnirea unui operator **(** şi cât timp stiva nu este vidă. Apoi se va introduce acest operator (**+**) în stivă.
  - Caracterul este un operator **(**: - se va introduce în stivă.
  - Caracterul este un operator **)**: - se vor extrage din stivă şi se vor scrie toţi operatorii de prioritate mai mare (*, +), până la întâlnirea caracterului **(** care, la rândul său, va fi extras din stivă, dar nu va fi scris în şirul de caractere de la ieşire.
  - Când se termină de citit şirul de caractere de intrare, se vor extrage din stivă (până când aceasta devine vidă) toţi operanzii, care se vor scrie în şirul de caractere de ieşire.
- Se afişează şirul de caractere de ieşire.

```
#include <stdio.h>
#include <alloc.h>
#include <conio.h>
#include <string.h>
#define N           80
typedef struct
        {
        int Valoare;
        }InfoT;
typedef struct Nod
        {
        InfoT Date;
        struct Nod *Urm;
        } NodT;
```

```
typedef struct
        {
        NodT *Varf;
        } StivaT;

void Creare(StivaT *StivaPtr)
        {
        StivaPtr->Varf=NULL;
        }

int TestVida(StivaT *StivaPtr)
        {
        return StivaPtr->Varf==NULL;
        }

void Introducere (StivaT *StivaPtr, InfoT *InfoPtr)
        {
        NodT *Elem;
        Elem=(struct NodT *) malloc(sizeof(NodT));
        Elem->Date=*InfoPtr;
        Elem->Urm=StivaPtr->Varf;
        StivaPtr->Varf=Elem;
        }

int Extragere(StivaT *StivaPtr, InfoT *InfoPtr)
        {
        NodT *Temp;
        if (TestVida(StivaPtr)) return 0;
        else
                {
                *InfoPtr=StivaPtr->Varf->Date;
                Temp=StivaPtr->Varf->Urm;
                free(StivaPtr->Varf);
                StivaPtr->Varf=Temp;
                return 1;
                }
        }

int InfoVarf(StivaT *StivaPtr, InfoT *InfoPtr)
        {
        if (TestVida(StivaPtr)) return 0;
        else
                {
                *InfoPtr=StivaPtr->Varf->Date;
                return 1;
                }
        }

void write_out(char out[N],int k,int poz)
{
int i;

gotoxy(1,poz);
if(k==0) printf("vida");
```

```
for(i=0;i<k;i++) printf("%c",out[i]);
}


void PrintStack(StivaT *StivaPtr,char o[N],int v,int poz)
        {
        NodT *s; //auxiliar
        write_out(o,v,poz);
        gotoxy(40,poz);
        s=StivaPtr->Varf;
        if(TestVida(StivaPtr)) printf("vida");
        while(StivaPtr->Varf!=NULL)
                {
                printf("%c",StivaPtr->Varf->Date.Valoare);
                StivaPtr->Varf=StivaPtr->Varf->Urm;
                }
        StivaPtr->Varf=s;
        }



void main(void)
{
int p; // pozitie afisare

char input_buf[N]; // buffer de intrare
char output_buf[N];//buffer de iesire
char *term="#";    //terminator de expresie

InfoT *a;
InfoT *b;

StivaT *stiva;

char c; //caracterul curent
char i; //index buffer de intrare
char j; //index buffer de iesire
char k;

clrscr();
printf("Introduceti expresia in forma infixata : ");
scanf("%s",input_buf);

//adauga terminatorul #
strcat(input_buf,term);

// creaza o stiva
Creare(stiva);

//algoritmul de conversie a expresiei din forma infixata in forma postfixata
i=0;
j=0;
c=input_buf[i];
i++;

gotoxy(1,3);
```

```
printf("Iesire");
gotoxy(40,3);
printf("Continutul stivei");
gotoxy(1,5);
printf("vida");
gotoxy(40,5);
printf("vida");

p=6;
while(c!='#')
        {
// verifica daca c este operand "a" = 0x61, ..., "z" =0x7a;
        if((c>=0x61)&&(c<=0x7a))
        {
        output_buf[j]=c;
        j++;
        PrintStack(stiva,output_buf,j,p);
        }
        else
        {
                InfoVarf(stiva,a);

                if(c=='*')
                {
                                if(a->Valoare=='*')
                                {
                                        while(!TestVida(stiva))
                                        {
                        if ((a->Valoare=='(')||(a->Valoare=='+')) break;
                                        Extragere(stiva,b);
                                        output_buf[j]=b->Valoare;
                                        j++;
                                        InfoVarf(stiva,a);
                                        }
                                }
                                b->Valoare=c;
                                Introducere(stiva,b);
                }
                if(c=='+')
                {
                                if((a->Valoare=='*')||(a->Valoare=='+'))
                                {
                                        while(!TestVida(stiva))
                                        {
                                        if (a->Valoare=='(') break;
                                        Extragere(stiva,b);
                                        output_buf[j]=b->Valoare;
                                        j++;
                                        InfoVarf(stiva,a);
                                        }
                                }
                                b->Valoare=c;
                                Introducere(stiva,b);
                }
                if(c=='(')
                {
```

```
                        b->Valoare=c;
                        Introducere(stiva,b);
                }
                if(c==')')
                {
                        if((a->Valoare=='*')||(a->Valoare=='+'))
                        {
                                while(!TestVida(stiva))
                                {
                                if (a->Valoare=='(') break;
                                Extragere(stiva,b);
                                output_buf[j]=b->Valoare;
                                j++;
                                InfoVarf(stiva,a);
                                }
                        }
                        Extragere(stiva,b); // se extrage '('
                }

                PrintStack(stiva,output_buf,j,p);

        }//end else
//citeste urmatorul caracter;
        c=input_buf[i];i++;
        p++;
        }//end while c!="#'

// descarca stiva
        while(!TestVida(stiva))
                {
                Extragere(stiva,b);
                output_buf[j]=b->Valoare;
                j++;
                }
        PrintStack(stiva,output_buf,j,p);



//scrie expresia in forma postfixata
printf("\n\nExpresia in forma postfixata : ");
for(k=0;k<j;k++) printf("%c",output_buf[k]);
printf("\n");

getch();

}// end main
```