RECURSIVITATE

1. Conţinutul lucrării

În lucrare este prezentată noţiunea de recursivitate, avantajele şi dezavantajele funcţiilor recursive în raport cu cele nerecursive, pe baza unor exemple simple.

2. Consideraţii teoretice

2.1. Mecanismul recursivităţii

Un obiect este recursiv dacă este definit prin el însuşi. O funcţie este recursivă dacă ea se autoapelează.

Recursivitatea poate fi:

- directă - când funcţia conţine un apel direct la ea însăşi;
- indirectă - când funcţia conţine un apel al altei funcţii, care la rândul său o apelează pe prima.

La fiecare apel al unei funcţii, parametrii şi variabilele automatice ale ei se alocă pe stivă într-o zonă independentă. Acest lucru se întâmplă la fiecare apel sau autoapel al funcţiei. De aceea datele amintite au valori distincte la fiecare reapelare. Variabilele statice şi cele globale ocupă tot timpul aceeaşi locaţie de memorie. Ca urmare, orice modificare asupra lor se face numai la adresa fixată în memorie, deci ele îşi păstrează valoarea de la un reapel la altul.

Revenirea dintr-o funcţie se face în punctul următor celui din care s-a făcut apelul. Adresa de revenire se păstrează tot în stivă. La revenire, stiva se reface la starea ei dinaintea apelului, deci variabilele automatice şi parametrii vor reveni la valorile lor dinaintea reapelului respectiv.

O problemă importantă este stoparea autoapelului. De aceea trebuie să existe o condiţie de terminare, fără de care un apel recursiv ar conduce la o buclă infinită. În aplicaţiile practice este necesar nu numai ca adâncimea recursivităţii sa fie finită, ci să fie relativ mică, deoarece fiecare apel recursiv necesită alocarea pe stivă a zonei de memorie pentru:

- parametrii funcţiei;
- variabilele automatice locale funcţiei;
- adresa de return (revenire în punctul de apel).

Ca urmare, stiva poate creşte foarte mult şi repede se ajunge la ocuparea întregului spaţiu de memorie alocat ei.

Un exemplu clasic de proces recursiv este calculul factorialului definit astfel:

$$fact(n) = \begin{cases} 1 & daca \quad n = 0 \\ n * fact(n-1) & daca \quad n > 0 \end{cases}$$

Se observă că în definiţia funcţiei *fact* există o parte care nu se defineşte prin ea însăşi şi anume fact(n)=1 dacă n=0.

În limbajul C, codul funcţiei corespunzătoare este următoarea:

```
double fact(int n)
{
        if (n==0)
                return 1.0;
        else
                return n*fact(n-1);
}
```

Recursivitatea liniară se caracterizează prin faptul că nu pot apărea pe ramuri diferite ale execuţiei programului mai multe apeluri recursive, adică pe un anumit nivel apare doar un singur apel recursiv.

Recursivitatea liniară întotdeauna poate fi transformată în iteraţie, ducând la economie de memorie şi timp de calcul (se elimină operaţiile de salvare de context la autoapelurile funcţiei).

Avantajul principal al recursivităţii este scrierea mai compactă şi mai clară a funcţiilor care exprimă procese de calcul recursive. În această clasă de procese intră și cele generate de metodele de căutare cu revenire (*backtracking*) şi metodele de divizare (*divide et impera*).

2.2. Exemple

2.2.1. Citirea a n cuvinte (şiruri de caractere), fiecare terminat cu spaţiu şi tipărirea lor în oglindă.

```
/* Programul L8Ex1 */

#include <stdio.h>
/* Programul citeste n cuvinte separate cu spatiu; dupa ultimul cuvant va exista spatiu si <ENTER>)  si le afiseaza "in oglinda" */

void revers() {
        char c;
        scanf("%c",&c);
        if (c!=' ') {
                printf("%c",c);
                revers();
        }
        printf("%c",c);
}

int main() {
         int n,i;
         printf("Numarul de cuvinte=");
         scanf("%d ",&n);
         for(i=1;i<=n;++i)
        {
                revers();
                printf("\n");
         }
         return 0;
 }
```

Funcţia revers citeşte câte un caracter pe care îl afişează până la întâlnirea spaţiului (terminatorul şirului de caractere). Fiecare autoapel conduce la păstrarea în stivă a variabilei locale c. Apariţia spaţiului conduce la terminarea apelurilor recursive ale funcţiei, urmând scrierea spaţiului şi a caracterelor în ordinea inversă introducerii lor.

2.2.2. Determinarea termenului minim al unui şir de n întregi.

```
/* Programul L8Ex2 */

#include <stdio.h>
/* Programul calculeaza minimul dintr-un sir cu termeni intregi */

#define NMAX 100
#define MAXIM 0x7FFFFFFF
int sir[NMAX];
int minim(int x, int y) {
   if (x<=y)
      return x;
   else
      return y;
}

int termen_minim(int dim_sir)  {
   if (dim_sir>=0)
       return minim(sir[dim_sir],termen_minim(dim_sir-1));
   else
       return MAXIM;
}

int main()
{
    int i,n;
    printf("Introduceti nr de termeni ai sirului n=");
    scanf("%d",&n);
    printf("Introduceti valorile termenilor\n");
    for(i=0;i<n;++i)
    {
      printf("sir[%d]=",i);
      scanf("%d",&sir[i]);
    }
    printf("\nSIRUL INTRODUS:\n");
    for(i=0;i<n;++i)
    {
       printf("%6d",sir[i]);
       if ((i+1) % 10 == 0) printf("\n");
    }
    printf("\nCel mai mic termen este %d\n",termen_minim(n-1));
    return 0;
}
```

2.2.3. Varianta recursivă şi nerecursivă a găsirii valorii celui de al n-lea termen al şirului lui Fibonacci.

Şirul lui Fibonacci este definit astfel:

Fib(0)=0; Fib(1)=1;

Fib(n)=Fib(n-1)+Fib(n-2) pentru n ≥ 2

```
/* Programul L8Ex3 */
#include <stdio.h>
#include <conio.h>
/* Numerele lui Fibonacci */
int fib1(int n)
/* VARIANTA RECURSIVA */
{
      if (n==0) return 0;
      else if (n==1) return 1;
           else return (fib1(n-1)+fib1(n-2));
}
int fib2(int n)
/* VARIANTA NERECURSIVA */
{
      int i,x,y,z;
      if (n==0) return 0;
      else if (n==1) return 1;
           else {
                    x=1;y=0;
                    for(i=2;i<=n;++i)
                    {
                       z=x; x=x+y; y=z;
                    }
                    return x;
                }
}
int main(void)
{
   int n;
   char ch;
   ch='D';
   while ((ch=='d')|| (ch=='D')) {
       printf("\nIntroduceti n=");
       scanf("%d",&n);
       printf("\nCALCUL RECURSIV:   fib(%d)=%d\n",n,fib1(n));
       printf("\nCALCUL NERECURSIV: fib(%d)=%d\n",n,fib2(n));
       printf("\nDoriti sa continuati ? Da=D/d");
       ch=getch();
    }
    return 0;
 }
```

Apelul recursiv conduce la creşterea rapidă a stivei, de aceea este preferabilă implementarea nerecursivă.

3. Mersul lucrării

3.1. Se vor analiza şi executa programele din lucrare şi din materialul de curs. Pentru un caz concret, se va trasa starea stivei, urmărindu-se creşterea pe măsura autoapelării funcţiei şi scăderea ei la revenirea din autoapel.

3.2. Să se scrie o funcţie recursivă şi una nerecursivă pentru calculul valorii polinoamelor Hermite H(x) definite astfel:

$H_0(x)=1$; $H_1(x)=2x$; $H_n(x)=2nH_{n-1}(x)-2(n-1)H_{n-2}(x)$, pentru $n\geq 2$

3.3. Problema turnurilor din Hanoi: Se consideră trei tije verticale A,B,C şi n discuri de diametre diferite. Iniţial toate discurile sunt puse în tija A, în ordinea descrescătoare a diametrului (discul cel mai mare la bază, iar cel mai mic în vârf). Se cere să se mute discurile de pe tija A pe tija C folosind tija B ca intermediar, folosind condiţiile:

a) la o manevră se mută un singur disc şi anume cel din vârful unei tije;
b) nu se poate pune un disc de diametru mai mare peste unul de diametru mai mic;
c) în final, pe tija C, discurile trebuie să fie în aceeaşi ordine ca în starea iniţială de pe tija A.

3.4. Să se scrie un program recursiv care citeşte n cuvinte şi le afişează în ordinea inversă a introducerii lor.

3.5. Să se scrie un program recursiv de generare a produsului cartezian a n mulţimi.

3.6. Să se scrie un program de generare recursivă a submulţimilor de k elemente ale mulţimii A cu n elemente (combinaţiile de n elemente luate câte k).

3.7. Să se scrie un program de rezolvare a problemei celor 8 regine (determinarea tuturor aşezărilor pe tabla de şah a celor 8 regine astfel încât să nu se atace).

3.8. Să se genereze recursiv permutările mulţimii A de n elemente.

3.9. Se consideră o bară de lungime m şi n repere de lungimi l_1, l_2, ... , l_n. Din bară trebuie tăiate bucăţi de lungimea reperelor date, astfel încât să rezulte din fiecare reper cel puţin o bucată şi pierderea să fie minimă.

3.10. Funcţia lui Ackermann. Să se scrie programul recursiv care calculează funcţia lui Ackermann definită astfel:

$Ack(0,n)=n+1$	pentru $n \in N$
$Ack(m,0)=Ack(m-1,1)$	pentru $m \in N^*$
$Ack(m,n)=Ack(m-1,Ack(m,n-1))$	pentru $m,n \in N^*$

3.11. Să se scrie un program recursiv pentru căutarea eficientă a unei valori într-un tablou care conține numere reale ordonate crescător.

3.12. Să se scrie un program recursiv pentru găsirea eficientă a monedei false dintr-un sac cu 177147 monede. Se știe ca moneda falsă este mai ușoară decât celelalte și că sacul conține doar o astfel de monedă. Aveți la dispoziție doar o balanță cu talere care poate realiza cântăriri precise.