Recursivitate. Functii recursive

BREVIAR TEORETIC

RECURSIVITATE

Recursivitatea este o tehnică de programare în care un subprogram se auto-apelează, eventual cu alte valori pentru parametri reali (de apel). Limbajul C face parte din clasa limbajelor de programare care admit scrierea de funcții recursive.

SCHELETUL UNEI FUNCȚII RECURSIVE

Acest schelet este prezentat în continuare.

```
functieR( int k )
{
     if( <conditie_de_final> )
          return /*…*/;        //revine pe nivelul anterior

     /*
      * ... operatii ...
      */

     functieR( k + 1 );        //apel pe nivelul superior
     return /*...*/;           //revine pe nivelul anterior
}
```

CALCULUL FACTORIALULUI. PREZENTAREA PROBLEMEI

Problema calcului factorialului implica două abordări:

- problema iterativă:

$$n!=1*2*3*\ldots*n$$

- problema recursivă:

$$n!=\begin{cases}1, n=0\\ n(n-1)!, n>0\end{cases}$$

CALCULUL FACTORIALULUI. VERSIUNEA ITERATIVĂ

```
//Versiunea iterativă
int fact( int n )
{
```

```
     int i,f;

     for( i = f = 1; i <= n; i++ )
          f *= i;

     return f;
}
```

CALCULUL FACTORIALULUI. VERSIUNEA RECURSIVĂ

```
//Versiunea recursivă
int fact( int n )
{
     if( n == 1 )                    //conditie de final
          return 1;                  //apel recursiv
     return n * fact( n - 1 );
}
```

GENERAREA DE SUBMULȚIMI

Pentru mulțimea: { a, b, c } se pot genera următoarele mulțimi:

- Permutări:

```
{{a, b, c}, {a, c, b}, {b, a, c}, {b, c, a}, {c, a, b}, {c, b, a}}
```

- Aranjamente de 2 elemente cu repetiție:

```
{{a, a}, {a, b}, {a, c}, {b, a}, {b, b}, {b, c}, {c, a}, {c, b}, {c, c}}
```

- Combinări de 2 elemente cu repetiție:

```
{{a, a}, {a, b}, {a, c}, {b, b}, {b, c}, {c, c}}.
```

Pentru determinarea acestor mulțimi se pot folosi cu succes funcții recursive, lucru ce va fi exemplificat în problema 2, rezolvată la sfârșitul laboratorului.

PROBLEME REZOLVATE

1. Problema calculului combinărilor:

```
#include <stdio.h>

/* Funcția de calcul a combinărilor C(n, k) utilizând n! */
int combinariF( int n, int k )
{
     return factorialR( n ) / factorialR( k ) / factorialR( n-k );
}

/* Funcția de calcul a combinărilor C(n, k)
 * RECURSIV! C(n,k) = C(n-1,k) + C(n-1,k-1)
 */
int combinariR( int n, int k )
```

```
{
     if ( k > n ) return 0;
     if ( k == 0 || k == n ) return 1;
     return combinariR( n-1, k ) + combinariR( n-1, k-1 );
}

/* Funcţia de calcul a factorialului RECURSIVĂ */
int factorialR( int n )
{
     int f;
     if ( n <= 1 ) //conditie de final
          return 1; //apel recursiv
     return n * factorialR( n-1 );
}

/*
 * Funcţia main( ... )
 */
int main( void )
{
     int N = 7, K = 3;
     printf( "\nSTART.\nCALCUL COMBINARI:" );

     printf( "\n\t- CombinariF( %d, %d ) = %d ", N, K, combinariF( N, K ) );
     printf( "\n\t- CombinariR( %d, %d ) = %d ", N, K, combinariR( N, K ) );

     printf( "\nEND.\n" );

     return 0;
}
```

```
START.
CALCUL COMBINARI:
        - CombinariF( 7, 3 ) = 35
        - CombinariR( 7, 3 ) = 35
END.

Process returned 0 (0x0)   execution ti
```

2. Problema generării submulțimilor:

```
/*
 * Generarea aranjamentelor de N elemente luate cate M.
 */
#include <stdio.h>

// Declaraţii funcţii
void Aranjamente( int k, int n, int ind[], char car[], int m );
void Print( int ind[], char car[], int m );
/*
 * Funcţia main( ... )
 */

int main( void )
{
```

```
     int i, N, M, idx[ 20 ];
     char sir[ 20 ];

     /* Citire sirului de N caractere */
     printf( "\nIntrodu caracterele [ fara spatii ]: ");
     N = strlen( gets( sir ) );

     printf( "\nSirul preluat, cu N = %d: ", N );
     for( i = 0; i < N; i++ )
          printf( "%c", sir[ i ] );

     /* Citire lui M */
     do
     {    printf("\nIntrodu M: ");
          scanf( "%d", &M );
     } while( M <= 0 || M > N );

     /* Apelul functiei recursive.
      * Pornim de la prima pozitie.  */
     Aranjamente( 0, N, idx, sir, M );

     return 0;
}

/*
 * Funcția de tipărire a soluției în care se face
 * asocierea dintre poziția generata și caracter.
 */
void Print( int ind[], char car[], int m )
{
     int i;
     /* Tiparirea caracterelor pe baza indicilor  */
     for( i = 0; i < m; i++ )
          printf("%c", car[ ind[ i ] ] );
     printf("\n");
}

/*
 * Funcția recursivă pentru calculul aranjamentelor.
 * Parametrul k indica nivelul de recursivitate.
 */
void Aranjamente( int k, int n, int ind[], char car[], int m )
{
     int j;

     /* Când submulțimea de prelucrat este plina,
        se tipărește soluția găsită până în acest moment. */
     if( k >= m )
          Print( ind, car, m );
     else
     {    /* Pe poziția dată de k punem, pe rand, fiecare caracter. */
          for( j = 0; j < n; j++ )
          {    ind[ k ] = j;
               /* Generarea submulțimii se reduce la generarea
                    unei alte submulțimi având un element mai mult
                    decât în etapa curentă. */
```

```
                Aranjamente( k+1, n, ind, car, m );
            }
        }
}
```

PROBLEME PROPUSE SPRE REZOLVARE

1. Scrieți un program ce va calcula valorile funcției Ackermann. Funcția Ackermann este definită pentru argumentele *m*,*n* numere naturale prin:

$$a(m,n)=\begin{cases} n+1, m=0 \\ a(m-1,1), n=0 \\ a(m-1,a(m,n-1)), altfel \end{cases}$$

2. Scrieți un program ce va rezolva problema calculului celui mai mare divizor comun dintre două numere naturale a și b. Aceasta problemă poate fi rezolvată recursiv, conform definiției următoare:

$$(a,b)=\begin{cases} a, a=b \\ (a-b,b), a>b \\ (a,b-a), b>a \end{cases}$$

3. Știind că aplicația din laborator rezolva problema „aranjamentelor de M elemente cu repetiție”, modificați aplicația astfel încât să rezolvați problema „permutărilor” și a „combinărilor de M elemente cu repetiție”.

TEMA. SOLUȚIE

GENERAREA DE SUBMULȚIMI

Pentru mulțimea: { a, b, c } se pot genera următoarele mulțimi:

- Permutări:

```
{{a, b, c}, {a, c, b}, {b, a, c}, {b, c, a}, {c, a, b}, {c, b, a}}
```

- Aranjamente de 2 elemente cu repetiție:

```
{{a, a}, {a, b}, {a, c}, {b, a}, {b, b}, {b, c}, {c, a}, {c, b}, {c, c}}
```

- Combinări de 2 elemente cu repetiție:

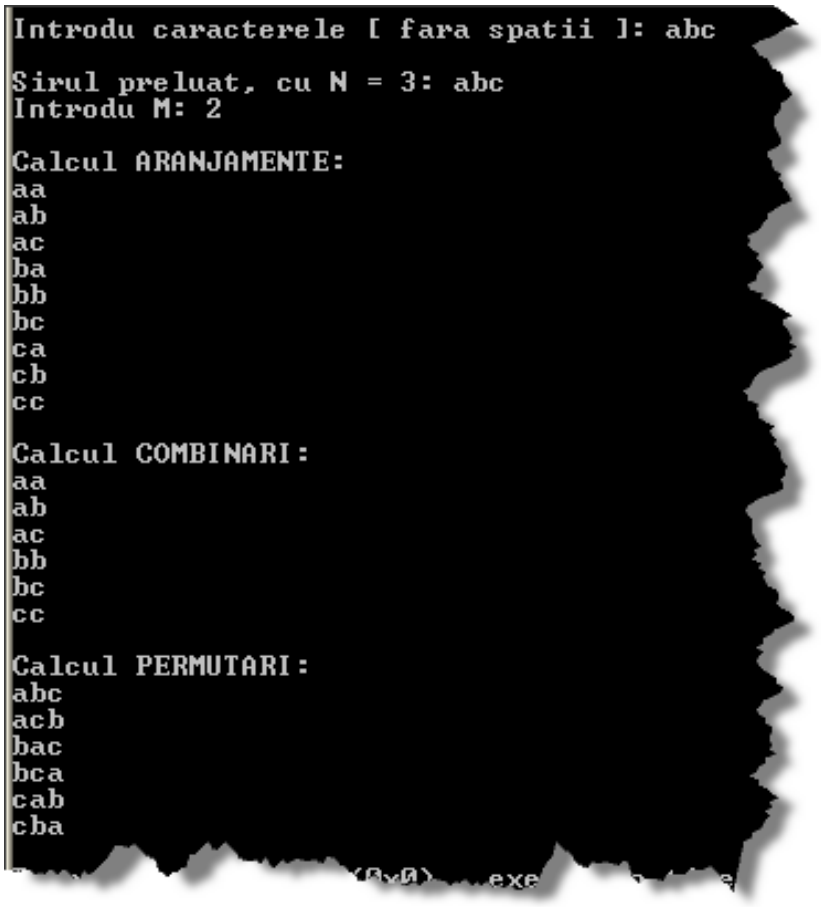
```
{{a, a}, {a, b}, {a, c}, {b, b}, {b, c}, {c, c}}.
```

SOLUȚIE

```
#include <stdio.h>
/*
* Declarații funcții */

void Aranjamente( int k, int n, int ind[], char car[], int m );
void Combinari( int k, int j,int n, int ind[], char car[], int m );
void Permutari( int k, int n, int ind[], char car[], int bifat[] );
void Print( int ind[], char car[], int m );

/*
 * Funcția principală - int main( ... ) */
int main( void )
{    /* START */
     int i, N, M, idx[ 20 ], bifat[ 20 ];
     char sir[ 20 ];

     /*
      * Citirea șirului de N caractere */
     printf( "\nIntrodu caracterele [ fara spatii ]: ");
     N = strlen( gets( sir ) );
     printf( "\nSirul preluat, cu N = %d: ", N );
     for ( i = 0; i < N; i++ )
          printf( "%c", sir[ i ] );

     /*
      * Citirea valorii lui M */
     do
     {    printf("\nIntrodu M: ");
          scanf( "%d", &M );
     }
     while ( M <= 0 || M > N );

     /*
      * Apelul funcției recursive. Pornim de la prima poziție. */
```

```
     printf( "\nCalcul ARANJAMENTE:\n" );
     Aranjamente( 0, N, idx, sir, M );
     /*  */
     printf( "\nCalcul COMBINARI:\n" );
     Combinari( 0, 0, N, idx, sir, M );
     /*  */
     printf( "\nCalcul PERMUTARI:\n" );
     for( i = 0; i < N; i++ )
          bifat[ i ] = 0;          // Bifam ca niciun caracter nu a fost ales
     Permutari( 0, N, idx, sir, bifat );
     /* END */

     return 0;
}

/*
 * Funcția de tipărire a soluției în care se face
 * asocierea dintre poziția generata și caracter. */
void Print( int ind[], char car[], int m )
{    int i;
     /*
      * Tipărirea caracterelor pe baza indicilor */
     for ( i = 0; i < m; i++ )
          printf("%c", car[ ind[ i ] ] );
     printf("\n");
}

/*
 * Funcția recursivă pentru calculul Aranjamentelor.
 * Parametrul k indica nivelul de recursivitate. */
void Aranjamente( int k, int n, int ind[], char car[], int m )
{    int j;
     /*
      * Când submulțimea de prelucrat este plina,
      * se tipărește soluția găsita până în acest moment. */
     if ( k >= m )
          Print( ind, car, m );
     else
     {    /*
           * Pe poziția data de k punem, pe rand, fiecare caracter. */
          for ( j = 0; j < n; j++ )
          {    ind[ k ] = j;
               /* Generarea submulțimii se reduce la generarea
                * unei alte submulțimi având un element mai mult
                * decât în etapa curenta. */
               Aranjamente( k+1, n, ind, car, m );
          }
     }
}

/*
 * Funcția recursivă pentru calculul Combinărilor.
 * Parametrul k indica nivelul de recursivitate.
 * Parametrul j ne permite ca la nivel k sa alegem
 * caracterele superioare, caracterului curent. */
void Combinari( int k, int _j, int n, int ind[], char car[], int m )
```

```
{    int j;
     /*
      * Când submulțimea de prelucrat este plina,
      * se tipărește soluția găsită până în acest moment. */
     if ( k >= m )
          Print( ind, car, m );
     else
     {    /* Pe poziția data de k punem, pe rand, fiecare caracter. */
          for ( j = _j; j < n; j++ )
          {    ind[ k ] = j;
               /* Generarea submulțimii se reduce la generarea
                * unei alte submulțimi având un element mai mult
                * decât în etapa curenta. */
               Combinari( k+1, j, n, ind, car, m );
          }
     }
}

/*
 * Funcția recursivă pentru calculul Permutărilor.
 * Parametrul k indica nivelul de recursivitate. */
void Permutari( int k, int n, int ind[], char car[], int bifat[] )
{    int j;
     char c;

     /*
      * Când submulțimea de prelucrat este plina,
      * se tipărește soluția găsită până în acest moment. */
     if ( k >= n )
          Print( ind, car, n );
     else
     {    /*
           * Pe poziția data de k punem, pe rand, fiecare caracter. */
          for ( j = 0; j < n; j++ )
          {    //...dacă acest caracter nu a fost deja ales
               if( bifat[ j ] == 0 )
               {    ind[ k ] = j;
                    bifat[ j ] = 1;
                    /* Generarea submulțimii se reduce la generarea
                     * unei alte submulțimi având un element mai mult
                     * decât în etapa curenta. */
                    Permutari( k+1, n, ind, car, bifat );
                    /*   */
                    bifat[ j ]  = 0;
               }
          }
      }
}
```


Introdu caracterele [fara spatii]: abc
Sirul preluat, cu N = 3: abc
Introdu M: 2
Calcul ARANJAMENTE:
aa
ab
ac
ba
bb
bc
ca
cb
cc
Calcul COMBINARI:
aa
ab
ac
bb
bc
cc
Calcul PERMUTARI:
abc
acb
bac
bca
cab
cba