Tehnici de programare

Recursivitate

Recursivitatea reprezintă proprietatea unor noțiuni de a se defini prin ele însele.

Exemple:

- factorialul unui număr: $N!=N\cdot(N-1)!$;
- ridicarea la putere: $a^n=a\cdot a^{n-1}$;
- termenul unei progresii aritmetice: $a_n = a_{n-1} + r$;
- șirul lui Fibonacci: $F_n = F_{n-1} + F_{n-2}$;

Să observăm că aceste reguli nu se aplică întotdeauna. De exemplu, pentru $3!$ am obține:

$3! = 3\cdot2!$, $2!=2\cdot1!$, $1!=1\cdot0!$, $0!=0\cdot(-1)!$

De aici am putea deduce că $0!=0$ și înlocuind în relațiile de mai sus obținem că $n!=0!=0$, pentru orice număr natural n. Bineînțeles, nu este corect. De fapt, formula recursivă pentru $n!$ se aplică numai pentru $n>0$, iar prin definiție $0!=1$.

Astfel, identificăm următoarea definiție pentru $n!$, acum completă:

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

Similar, pentru toate formulele de mai sus exista cel puțin o situație în care formula recursivă nu se mai poate aplica, iar rezultatul se determină în mod direct.

În C, recursivitatea se realizează prin intermediul funcțiilor, care se pot *autoapela*.

Ne amintim că o funcție trebuie definită iar apoi se poate apela. Recursivitatea constă în faptul că în definiția unei funcție apare apelul ei însăși. Acest apel, care apare în însăși definiția funcției, se numește **autoapel**. Primul apel, făcut în altă funcție, se numește **apel principal**.

Exemplu de funcție recursivă în C

Să scriem o funcție C care returnează factorialul unui număr natural transmis ca parametru. Varianta nerecursivă (iterativă) este următoarea:

```
int fact(int n){
    int p = 1;
    for(int i = 1 ; i <= n ; i ++)
    return p;
}
```

Să observăm că această funcție determină rezultatul corect pentru valori ale lui `n` mai mari sau egale cu `0` (valori mici, practic `n <= 12`). Funcția determină corect rezultatul și pentru `n == 0`.

O variantă recursivă pentru determinarea lui `n!`, care folosește observațiile de mai sus, este:

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

Cum funcționează recursivitatea?

Ne amintim că toate variabilele locale din definiția unei funcții precum și valorile parametrilor formali se memorează la apel în memoria de tip STIVĂ (STACK).

Pentru fiecare apel al unei funcții se adaugă pe stivă o zonă de memorie în care se memorează variabilele locale și parametrii pentru apelul curent. Această zonă a stivei va exista până la finalul apelului, după care se va elibera. Dacă din apelul curent se face un alt apel, se adaugă pe stivă o nouă zonă de memorie, iar conținutul zonei anterioare este inaccesibil până la finalul acelui apel. Aceste operații se fac la fel și dacă al doilea apel este un autoapel al unei funcții recursive.

Să considerăm acum următoarea secvență:

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

int main(){
    int x = fact(3);
    printf("fact(%d) = %d", 3, x);
    return 0;
}
```

Să urmărim pas cu pas execuția acestui program:

Pas	Conținut stivă	Observații
`int x =`	`x = ??`	În zona curentă a stivei se alocă memorie pentru variabila `x`. Să o numim `Zona 0`.
`fact(3)`	`Zona 1: n = 3, f = 3 * fact(2) = ??` `Zona 0: x = ??`	În apelul principal are loc autoapelul `fact(3)`. Se alocă o nouă zonă pe stivă, pentru acest apel, `Zona 1`. Deoarece `n>0`, are loc apelul `fact(2)`.

`fact(2)`	`Zona 2: n = 2, f = 2 * fact(1) = ??`
	`Zona 1: n = 3, f = 3 * fact(2) = ??`
	`Zona 0: x = ??`
		În `Zona 1` a stivei se face autoapelul `fact(2)`. Se alocă o nouă zonă pe stivă, pentru acest apel, `Zona 2`. Deoarece `n>0`, are loc autoapelul `fact(1)`.

`fact(1)`	`Zona 3: n = 1, f = 2 * fact(0) = ??`
	`Zona 2: n = 2, f = 2 * fact(1) = ??`
	`Zona 1: n = 3, f = 3 * fact(2) = ??`
	`Zona 0: x = ??`
		În `Zona` a stivei se face autoapelul `fact(1)`. Se alocă o nouă zonă pe stivă, pentru acest apel, `Zona 3`. Deoarece `n>0`, are loc autoapelul `fact(0)`.

`fact(0)`	`Zona 4: n = 0, f = 1`
	`Zona 3: n = 1, f = 2 * fact(0) = ??`
	`Zona 2: n = 2, f = 2 * fact(1) = ??`
	`Zona 1: n = 3, f = 3 * fact(2) = ??`
	`Zona 0: x = ??`
		În `Zona 3` a stivei se face autoapelul `fact(0)`. Se alocă o nouă zonă pe stivă, pentru acest apel, `Zona 4`. Suntem în cazul particular și nu mai are loc autoapelul. Rezultatul autoapelului `fact(0)` este `1`. `Zona 4` se eliberează.

`fact(1)`	`Zona 3: n = 1, f = 1 * 1 = 1`
	`Zona 2: n = 2, f = 2 * fact(1) = ??`
	`Zona 1: n = 3, f = 3 * fact(2) = ??`
	`Zona 0: x = ??`
		Se revine în apelul `fact(1)`, adică în `Zona 3`. Se calculează `f=1` și se termină și autoapelul `fact(1)` cu valoarea `1`. Se eliberează `Zona 3`.

`fact(2)`	`Zona 2: n = 2, f = 2 * 1 = 2`
	`Zona 1: n = 3, f = 3 * fact(2) = ??`
	`Zona 0: x = ??`
		Se revine în apelul `fact(2)`, adică în `Zona 2`. Se calculează `f=2` și se termină și autoapelul `fact(2)` cu valoarea `2`. Se eliberează `Zona 2`.

`fact(3)`	`Zona 1: n = 3, f = 3 * 2 = 6` `Zona 0: x = ??`	Se revine în apelul `fact(3)`, adică în `Zona 1`. Se calculează `f=6` și se termină și

	autoapelul `fact(3)` cu valoarea `6`. Se eliberează `Zona 1`.
`printf` `return 0;`	`Zona 0: x = 6` Se revine în apelul funcției `main`, adică în `Zona 0`. Se calculează `x=6` și se afișează această valoare. După instrucțiunea `return 0;` se eliberează `Zona 0`. Execuția programului se încheie.

Observații: La fiecare apel al funcției `fact` avem variabilele `n` și `f`. Ele însă sunt variabile diferite în fiecare apel de funcție, cu valori diferite, memorate în zone diferite ale stivei. La un moment dat, se pot folosi numai variabilele din zona de memorie curentă, cea din "vârful" stivei.

Observații

- este obligatoriu ca în definiția unei funcții recursive **să apară cazul particular** (în care să nu aibă loc autoapelul). În caz contrar autoapelurile vor avea loc "la nesfârșit". De fapt, în urma prea multor autoapeluri, stiva se va ocupa în totalitate și execuția programului se va întrerupe.
- este obligatoriu ca, pentru cazurile neelementare, valorile la autoapel a parametrilor să se apropie de valorile din cazul elementar. Altfel se va întâmpla situația descrisă mai sus: stiva se va ocupa în totalitate și programul se va opri, fără a determina/afișa rezultatele dorite :).

Tipuri de recusivitate

- recursivitate directă: în definiția funcției `F` apare apelul funcției `F`;
- recursivitate indirectă: în definiția funcției `F` apare apelul funcției `G`, iar în definiția funcției `G` apare apelul lui `F`.

Întrebarea 1 Ce se afiseaza la apelul lui f(65,100)?

```
1  void f (int x, int y)
2  {
3      if(x>0){
4          if(x%4==0){
5              printf("%c",'x');
6              f(x-1,y+1);
7          }
8          else{
9              f(x/3,y-1);
10             printf("%c",'y');
11         }
12     }
13 }
```

Întrebarea 2 Care este rezultatul la apelul lui f(15, 2)

```
void f (int n, int x){
    if(x>n)
        printf(""%d"",0);
    else if(x%4<=1)
        f(n,x+1);
    else{
        f(n,x+3);
        printf(""%d"",1);
    }
}
```

Întrebarea 3 Care este rezultatul apelului lui f(3, 17)?

```
void f( int a, int b){
    if(a<=b){
        f(a+1,b-2);
        printf(""%c"",'*');
    }
    else
        printf(""%d"",b);
}
```

Aplicații propuse

Aplicația 10.1: Calculați recursiv cel mai mare divizor comun a două numere.

$$cmmdc(a,b) = \begin{cases} a, & dacă\ b = 0 \\ cmmdc(b, a\%b), & dacă\ b > 0 \end{cases}$$

Aplicația 10.2: Calculați recursiv suma cifrelor unui număr natural.

$$suma\_cifrelor(n) = \begin{cases} n, & dacă\ n < 10 \\ n\%10 + suma\_cifrelor(n/10), & dacă\ n >= 10 \end{cases}$$

Aplicația 10.3: Determinați recursiv cifra maximă a unui număr natural.

$$cifra\_max(n) = \begin{cases} n, & dacă\ n < 10 \\ cifra\_\max(n\%10, cifra\_max(n/10)), & dacă\ n >= 10 \end{cases}$$

Aplicația 10.4: Sa se determine recursiv al n-lea termen Fibonacci.

$$fib(n) = \begin{cases} 1, & dacă\ n = 1\ sau\ n = 0 \\ fib(n-1) + fib(n-2), & dacă\ n > 1 \end{cases}$$

Aplicația 10.5: Se considera sirul-de-caractere de tip Fibbonaci in care primii doi termeni sunt sirurile de caractere s1 si s2 (pot fi transmise ca si parametri functiei), iar orice alt termen se obtine prin concatenarea celor doi termeni

anteriori. Sa se implementeze o functie care un numar natural n (si orice alti parametri considerati necesari) si afiseaza termenul de pe pozitia n din sirul construit conform celor de mai sus.

Aplicația 10.6: Se poate demonstra (v. Knuth) ca limita raportului a doi termeni invers consecutivi din sirul lui Fibbonaci este egala cu Phi (proportia de aur).

Astfel $\phi = \lim_{n\to\infty}(\frac{F_{n+1}}{F_n})$

Implementati o functie (recursiva) care sa determine Phi cu o anumita precizie.

Aplicația 10.7: Serii de puteri. Calculaţi, cu o precizie dată, valoarea lui ex după dezvoltarea în serie Taylor:
$ex = 1 + x^1/1! + x^2/2! + x^3/3! + ...$
Calculaţi suma până când termenul curent devine mai mic decât o valoare dată (de ex. 10^{-6}). Pentru a evita recalcularea lui n! transmiteţi ca parametru şi termenul curent, şi calculaţi-l pe următorul după relaţia: $x^n/n! = x^{n-1}/(n-1)! * x/n$.

Aplicația 10.8: Să se implementeze cu o functie recursivă căutarea binară pe un vector ordonat (Binary Search - bsearch). Functia trebuie să returneze dacă numărul căutat există sau nu în vector. Principiul după care funcționează bsearch este că vectorul se împarte la fiecare pas în 2 vectori de dimensiuni egale (sau aproximativ egale) și continuă căutarea doar în unul dintre ei (apelează recursiv funcția de căutare doar pe jumătate din vectorul inițial), cel din partea stângă sau cel din partea dreaptă, în funcție de valoarea numărului căutat.

Exemplu:
v = 1, 5, 8, 12, 17, 20, 33, 40 , n = 33
imparte v in 2 vectori egali :v1 = 1, 5, 8, 12 v2 = 17, 20, 33, 40

continua cautarea lui n = 33 in vectorul v2
acum v = 17, 20, 33, 40, n=33
imparte v in 2 vectori egali :v1 = 17, 20 v2 = 33, 40

continua cautarea lui n = 33 in vectorul v2
acum v = 33, 40, n=33
imparte v in 2 vectori egali :v1 = 33 v2 = 40

continua cautarea lui n = 33 in vectorul v1
acum v = 33, n=33
l-a gasit

*Bibliografie material laborator:
https://www.pbinfo.ro/articole/3873/recursivitate
https://staff.cs.upt.ro/~marius/curs/lp/tema2.html