Ciobanu Rareș-Constantin

SEMINAR 6 – ATP

Recapitulare:

Memoria Stack vs Heap?

Ambele sunt stocate în memoria RAM a computerului.

- Stack - este utilizată pentru alocarea statică a memoriei;
  - Alocarea variabilelor alocate pe stack se produce la COMPILARE.
  - Memoria in stack(stiva) este rezervata in ordinea LIFO(last in first out).
  - Accesarea memoriei este foarte rapida.
- Heap - este utilizată pentru alocarea dinamică a memoriei;
  - Alocarea variabilelor alocate pe heap se produce la RUNTIME.
  - Memoria poate fi accesata in orice moment la intamplare(nu exista o ordine de stocare a variabilelor in memorie).
  - Accesarea memoriei este putin mai lenta.

Puteți utiliza memoria stack dacă știți exact câtă memorie trebuie să alocați înainte de COMPILAREA programului și dacă ea nu este prea mare.

Puteți utiliza memoria heap dacă nu știți exact câtă memorie veți avea nevoie la RUNTIME sau dacă trebuie să alocați multă memorie.

Recursivitate

Spunem că o noţiune este definită recursiv, dacă în definirea ei apare însăşi noţiunea care se defineşte.

- **Recursivitate directă** - proprietatea funcţiilor de a se autoapela.
  - Atunci cand in interiorul functiei se autoapeleaza functia.
    1. functieA apeleaza functieA
- **Recursivitate indirectă** - proprietatea funcţiilor de a se autoapela prin intermediul altor funcţii.
  - Atunci cand o functie apeleaza o alta functie care va apelea functia initiala**:**
    1. functieB apeleaza functieA
    2. functieA apeleaza functieB

Cum gândim o funcţie recursivă în general?

Stabilim ce se execută la o anumită etapă, apoi **reluăm** procedeul pentru toate celelalte elemente rămase, prin reapelarea funcţiei, având grijă să impunem **condiţii de oprire** ale reapelării funcţiei.

Observaţii importante:

- Orice funcţie recursivă trebuie să conţină o condiţie de oprire a apelului recursiv. Fără această condiţie, funcţia teoretic se reapelează la infinit, dar nu se întâmpla acest lucru, deoarece se umple segmentul de stiva alocat funcţiei şi programul se înterupe cu eroare.
- La fiecare reapel al funcţiei se execută aceeaşi secvenţă de instruciuni.
- Ţinând seama de observaţiile anterioare, pentru a implementa o funcţie recursivă, trebuie să identificăm:
  - relaţia de recurenţa (ceea ce se execută la un moment dat şi se reia la fiecare reapel)
  - condițiile de oprire ale reapelului
- În cazul în care funcţia are parametrii, aceştia se memorează ca şi variabilele locale pe stiva, astfel:
  - parametrii transmişi prin valoare se memorează pe stivă cu valoarea din acel moment
  - pentru parametrii transmişi prin referinţă se memorează adresa lor

Divide et Impera

Metoda **Divide et Impera** (Imparte si Stapaneste) este o metoda de programare care se aplica problemelor care pot fi descompuse in subprobleme independente, similare problemei initiale, de dimensiuni mai mici si care pot fi rezolvate foarte usor. Procesul se reia pana cand (in urma descompunerilor repetate) se ajunge la probleme care admit rezolvare imediata.

Exerciții

1) Să se calculeze n! in varianta iterativa.

```
#include<stdio.h>
long int factorial(int n)
{
	long int f = 1;
	for (int i = 1; i <= n; i++)
		f = f * i;
	return f;
}
void main()
{
	int n;
	printf("Introduceti n= ");
	scanf("%d", &n);

	if (!n)
		printf("0!=1\n");
	else
		printf("%d!=%ld\n", n, factorial(n));

}
```

2) Să se calculeze n! in varianta recursiva.

```
// factorial(3)=3*factorial(2)=3*2*factorial(1)=3*2*1
#include<stdio.h>
long int factorial(int n)
{
	if (n == 1) return 1;
	else return n * factorial(n - 1);
}
void main()
{
	int n;
	printf("Introduceti n= ");
	scanf("%d", &n);

	if (!n)
		printf("0!=1\n");
	else
		printf("%d!=%ld\n", n, factorial(n));
}
```

3) Să se calculeze recursiv suma elementelor unui sir.

```
#include<stdio.h>
int suma(int n)
{
	if (n == 0) return 0;
	else return (n + suma(n - 1));
}
void main()
{
	int n;
	printf("Introduceti n: ");
	scanf("%d", &n);
	printf("Suma elementelor este %d\n", suma(n));
}
```

4) Scrieti o functie proprie care realizeaza calculul recursiv al sumei elementelor unui vector, de n<=10, de numere intregi. Scrieti functia main care citeste datele de la tastatura, calculeaza suma, utilizand functia recursiva anterior definita si afiseaza valoarea obtinuta.

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

int Suma(int n, int a[10])
{
	if (n == 0) return 0;
	else return(a[n] + Suma(n - 1, a));
}
void main()
{
	int a[10], n;
	// Citire date de intrare
	printf("Introduceti nr de elemente: ");
	scanf("%d", &n);
	for (int i = 1; i <= n; i++)
	{
		printf("Elementul [%d] = ", i);
		scanf("%d", &a[i]);
	}
	// Afisarea rezultatelor
```

```
	printf("Suma = %d", Suma(n, a));

	_getch();
}
```

5) Să se scrie un program C, pentru rezolvarea cmmdc-ului dintre două numere întregi fără semn (pentru determinarea cmmdc-ului vom folosi algritmul lui Euclid prin scăderi). Varianta iterativa.

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

unsigned int cmmdc(unsigned int a, unsigned int b)
{
	while (a != b)
	{
		if (a > b)
			a = a - b;
		else
			b = b - a;
	}
	return a;
}
void main()
{
	unsigned int x, y;
	printf("Introduceti x: ");
	scanf("%u", &x);
	printf("Introduceti y: ");
	scanf("%u", &y);

	if (!x || !y)		//daca x=0 sau y=0
		printf("cmmdc(%u,%u) = 1\n", x, y);
	else
		printf("cmmdc(%u,%u) = %u\n", x, y, cmmdc(x, y));

	_getch();
}
```

6) Să se scrie un program C, pentru rezolvarea cmmdc-ului dintre două numere întregi fără semn (pentru determinarea cmmdc-ului vom folosi algritmul lui Euclid prin scăderi). Varianta recursiva.

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

unsigned int cmmdc(unsigned int a, unsigned int b)
{
	if (a == b) return a;
	else
		if (a > b) return cmmdc(a - b, b);
		else return cmmdc(a, b - a);
}
void main()
{
	unsigned int x, y;
	printf("Introduceti x: ");
	scanf("%u", &x);
	printf("Introduceti y: ");
	scanf("%u", &y);
```

```
	if (!x || !y)		//daca x=0 sau y=0
		printf("cmmdc(%u,%u) = 1\n", x, y);
	else
		printf("cmmdc(%u,%u) = %u\n", x, y, cmmdc(x, y));

	_getch();
}
```

7) Sa se scrie o functie recursiva pentru determinarea sumei cifrelor unui numar natural.

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

int suma(int n)
{
	if (!n) return 0;			//!n adica n=0
	else return n % 10 + suma(n / 10);
}
void main()
{
	int n;
	printf("Introduceti numarul: ");
	scanf("%d", &n);

	printf("Suma cifrelor numarului este: %d", suma(n));

	_getch();
}
```

8) Se considera sirul lui Fibonacci (Un) definit astfel:

$$Un = \begin{cases} 0, & n=0 \\ 1, & n=1 \\ Un-1+Un-2, & \text{altfel} \end{cases}$$

Se citeste n, un numar natural. Sa se calculeze Un, in varianta iterativa.

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

void main()
{
	int n, U0 = 0, U1 = 1, U2;
	printf("n="); scanf("%d", &n);
	if (!n) printf("%d", U0);
	else
		if (n == 1) printf("%d", U1);
		else
		{
			for (int i = 2; i <= n; i++)
			{
				U2 = U0 + U1;
				U0 = U1;
				U1 = U2;
			}
			printf("%d", U2);
		}
	/*
	ptr. n=3
	i=2: U2=U0+U1
```

```
        U0=U1
        U1=U2
        i=3: U2=U1+U2
        */
        _getch();
}
```

9) Se considera sirul lui Fibonacci (Un) definit astfel:

$$Un = \begin{cases} 0, & n=0 \\ 1, & n=1 \\ Un-1+Un-2, & \text{altfel} \end{cases}$$

Se citeste n, un numar natural. Sa se calculeze Un, in varianta recursiva.

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

int U(int n)
{
        if (!n) return 0;
        else   if (n == 1) return 1;
        else   return U(n - 1) + U(n - 2);
}

void main()
{
        int n;
        printf("Introduceti n=");
        scanf("%d", &n);

        printf("Valoarea sirului in n este: %d", U(n));

        _getch();

}
```

10) Algoritmul de sortare QuickSort in mod recursiv. Algoritmul este de tip **divide et impera.** Pentru un set oarecare de **n** valori algoritmul **QuickSort** efectuează O(n*log(n)) comparații, iar în cazul cel mai nefavorabil caz se efectuează O(n*n) comparații.

```
#include<stdio.h>

void interschimbaValori(int &a, int &b) {
        int aux = a;
        a = b;
        b = aux;
}

/*
   1) aceasta functie alege ca pivot ultimul element din tablou;
   2) se rearanjează elementele în așa fel încât, toate elementele care sunt mai mari
decât pivotul merg în partea dreaptă a tabloului. Valorile egale cu pivotul pot sta în
orice parte a tabloului. În plus, tabloul poate fi împărțit în părți care nu au
aceeași dimensiune (nu sunt egale).

        Functia returneaza indexul unde se va imparti tabloul(se va imparti de la
indexul pivotului);
*/
int impartire(int tablou[], int stanga, int dreapta)
```

```
{
	// pivot (elementul care va fi mutat la pozitia sa corecta in tabloul sortat)
	// practic ultimul element din tablou va fi mutat undeva....
	int pivot = tablou[dreapta];

	int i = (stanga - 1);  // se pastreaza indexul celui mai mic element

	for (int j = stanga; j <= dreapta - 1; j++)
	{
		// daca elementul curent este mai mic decat pivotul
		if (tablou[j] < pivot)
		{
			i++;    // se mareste indexul celui mai mic element
			interschimbaValori(tablou[i], tablou[j]);
		}
	}
	//se muta pivotul pe pozitia lui corespunzatoare in tablou;
	interschimbaValori(tablou[i + 1], tablou[dreapta]);

	return i + 1;
}

/*
	tablou  --> tabloul care trebuie sortat;
	stanga  --> Indexul de start;
	dreapta  --> Indexul de final;

	Algoritmul imparte tabloul in bucati(nu neaparat egale) din ce in ce mai mici
si dupa care il sorteaza.
*/
void quickSort(int tablou[], int stanga, int dreapta)
{
	if (stanga < dreapta)
	{
		/*  pi este indexul de impartire
			practic se calculeaza unde se imparte tabloul(divide et impera)
		*/
		int pi = impartire(tablou, stanga, dreapta);

		quickSort(tablou, stanga, pi - 1);  // sortează elementele pana la
indexul de impartire PI; pentru ca știi deja ca pivotul folosit mai sus e deja pus pe
poziția corespunzătoare in tablou…nu mai trebuie sortat încă o dată și el

		quickSort(tablou, pi + 1, dreapta); // sorteaza elementele dupa indexul
de impartire PI;
	}
}

void afisareTablou(int tablou[], int N)
{
	int i;
	for (i = 0; i < N; i++)
		printf("%d ", tablou[i]);
	printf("\n");
}

void main() {
	int tablou[] = { 10, 7, 8, 9, 1, 5 };
	int n = sizeof(tablou) / sizeof(tablou[0]);//asa se poate afla cate elemente
are un vector
```

```
	quickSort(tablou, 0, n - 1);//n-1 pt elementele sunt stocate in vector de la
pozitia 0;
	printf("Vector sortat:\n");
	afisareTablou(tablou, n);
	}
```