12. Structuri. Structuri imbricate. Pointeri la structuri.

(Data structures. Imbricate structures. Pointers to structures).

1. Obiective:

- Înţelegerea noţiunii de date definite de utilizator(user-datatypes)
- Înţelegerea modalităţii de lucru cu structuri: declarare, iniţializări, acces la câmpuri
- Scrierea de programe folosind structuri simple şi imbricate
- Exersarea accesului cu pointeri la structuri de date

1'. Obiectives:

- Understanding the user-datatypes notion.
- Undestanding how structures work: declation, initialization, field access
- Writing simple programs using simple and nested structures
- Accessing the data structures with pointers

2. Rezumat:

Nevoia pentru structuri era una dintre motivaţiile principale pentru crearea limbajului C.
O structură reprezintă o mulţime ordonată de elemente grupate în vederea utilizării lor în comun. Ea se declara cu constructia *struct* astfel:

struct [nume_struct] {
lista_declaraţii;
}[$nume_1$*,*$nume_2$*,…,*$nume_n$*];*
unde:

- *nume_struct*, este un nou tip de date, tip definit de utilizator, cu construcţia *struct*;
- *lista_declaraţii*, este o listă prin care se declară componentele unei structuri, putând conţine elemente de forma tip_i $element_i$, unde tip_i este un tip admis de limbajul C inclusiv o nouă structură (in acest ultim caz avem *structuri imbricate*);
- $nume_1,\ldots,nume_n$ este o listă de variabile de tip *nume_struct*, putând lipsi la definirea structurii, caz în care este obligatoriu să fie precizat *nume_struct*.

Dacă avem *nume_struct* şi nu avem $nume_i$ atunci putem defini ulterior alte structuri de tip *nume_struct* cu declaraţia:

struct *nume_struct nume_nou;*
unde *nume_struct* reprezintă un nou tip de dată, un tip utilizator.

Elementele unei structuri se numesc *câmpuri*, iar gruparile de elemente *inregistrari*.
Printre elementele unei structuri se pot găsi şi alte structuri (inclusiv pointeri către structura însăşi), care sunt referite din exterior spre interior:

Accesul la elementele unei structuri se face precizând numele variabilei de tip structură, de exemplu *angajat*, şi câmpul care ne interesează (inclusiv daca e o structura imbricata), separate prin operatorul . *(punct)*.

angajat.data_nast.zi = 9; //zi *este camp al variabilei struct.* data_nast, *care este camp al variabilei-structura* angajat

Acest mod de acces se numeste *acces prin nume calificat*.

Accesul la elementele unei structuri se poate face şi printr-un pointer la structură, folosind operatorul -> (săgeată).

p = &angajat;
(p->data_nast).an = 1995; //acces prin pointer la campul data_nast, şi sub-campul an

Un tablou de structuri se declară considerând *struct nume_struct* ca fiind tipul tabloului.

Structurile pot fi transferate ca şi parametri funcţiilor.

O funcţie poate să returneze o structură.

În general, ***structurile nu se transferă prin valoare, ci prin pointeri spre structură.***

Pointeri la structuri

Un pointer către o structură se declară la fel ca şi orice pointer către orice alt tip de variabilă. Pointerul va conţine adresa primei componente a structurii la care pointează.

Pointerii la structuri sunt utili când dorim să transmitem o structură unei funcţii, prin intermediul lor putând să transmitem doar adresa structurii.
În corpul funcţiei accesul la câmpurile structurii se face prin intermediul pointerului p, astfel:

*(*p).zi*	*sau*	*p ->zi*
*(*p).luna*	*sau*	*p ->luna*
*(*p).an*	*sau*	*p ->an*

3. Exemple:

Exemplul 1.

```
/*  Citeste de la consolă a elementele unei structuri data_calend şi
afiseaza data  citita în alt format : zi/luna/an.     */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>  // pt. getch()
#include <string.h>
#include <stdlib.h> // pt. exit(int)

struct data_calend {
      int ziua;
      char luna[11];
      int anul;
};

// tablou global de pointeri la siruri de caractere, definit
char *tab[] = {"ian", "feb", "mar", "apr", "mai", "iun", "iul",
"aug", "sep", "oct", "noi", "dec"};

void puts_getch_exit(char *); // mentine afisat mesajul de eroare
pasat pana la o tasta, apoi termina procesul

void main(void)
{
      struct data_calend dc;
      int i, luna=0; // acest luna e alt spatiu de nume decat dc.luna

      puts("\nIntroduceti data curenta: ");
      printf("\n\t ziua (1,2,...31): ");
      scanf("%d", &dc.ziua);
      if((dc.ziua <= 0) || (dc.ziua > 31))
      puts_getch_exit("\nZiua incorecta !");
```

```
	printf("\n\t luna (ian, feb...): ");
	scanf("%s", dc.luna);
	for(i=0; i <12; i++)  {
		if(_stricmp(dc.luna, tab[i]) == 0)  {
			luna = i+1;
			break;
		}
	}
	if(luna == 0) // daca var. luna nu s-a modificat, cuvantul
tastat nu era identic cu nici unul dintre elementele lui tab[]
		puts_getch_exit("\nLuna incorecta !");
	printf("\n\t anul : ");
	scanf("%d", &dc.anul);
	if(dc.anul <= 0)puts_getch_exit("\nAnul incorect (negativ)!");
	printf("\n Data introdusa este :\n %d/%d/%d", dc.ziua, luna,
dc.anul);
	_getch();
}

void puts_getch_exit(char *ptr){
	puts(ptr);// printf(ptr); -- de asemenea e OK
	_getch();
	exit(0);
}
/*-------------------------------------------------------------------*/
```

Exemplul 2.

```
// Programul declara un nou tip de date, forma, si defineste o
variabilă de acest tip, cerc.
// Afisarea o acceseaza prin metoda "nume calificat"
#include <stdio.h>
#include <conio.h>


void main(void)
{
	struct forma{
	 int tip;
	 int culoare;
	 float raza;
	 float suprafata;
	 float perimetru;
	} cerc = {2, 1, 5.0, 78.37, 31.42};

	printf("cerc.tip: %d\n", cerc.tip);
	printf("cerc.culoare: %d\n", cerc.culoare);
	printf("cerc.raza: %f\n", cerc.raza);
	printf("cerc.suprafata: %f\n", cerc.suprafata);
	printf("cerc.perimetru: %f\n", cerc.perimetru);

	// definim ulterior o alta variabila-structura de acelasi tip,
"forma", ca si "cerc"
	struct forma sfera = {3, 1, 5.}; // nu i-am initializat toate
campurile la declarare
	sfera.suprafata = 4. * 3.14 * sfera.raza * sfera.raza;
	printf("sfera.suprafata: %f\n", sfera.suprafata); // 314
	printf("sfera.perimetru: %f\n", sfera.perimetru); // camp
	//neinitilaizat
	_getch();
}
/*-------------------------------------------------------------*/
```

Exemplul 3.

```
/* Programul citeste datele pentru n persoane (nume, prenume, data
nasterii, codul numeric personal), apoi le afiseaza.*/
// rezerva spatiu pe heap-ul static local al functiei main() pt.
MAX=20 structuri, memorate ca elementele unui tablou
// In functii accesul la campuri se face cu operatorul "->" (pointer
la structura)
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>

#define MAX 20

struct data_calend {
	int ziua; //	tip 1  elem.1
	char luna[11]; // tip2  elem.2
	int anul;//		...
};

struct date_pers {
	char nume[16];
	char prenume[20];
	long cod;
	struct data_calend data_nast;  // struct. imbricata
};

void CitDatePers(struct date_pers *); // pointer la structura
void AfisDatePers(struct date_pers *);


void main(void){
struct date_pers dp[MAX];  // dp e un tabel de structuri
int i, n;
	printf("\nNumar angajati : ");
	scanf("%d", &n);
	if(n < 0)  {
		printf("\n Numar invalid !");_getch();
		return;
	}
	printf("\n Introduceti datele personale :");
	for(i=0;i<n;i++) {
		printf("\n Persoana %d:", i+1);
		CitDatePers(&dp[i]);
	}

	printf("\ntest: dp[0].data_nast.ziua = %d\n",
	dp[0].data_nast.ziua);// test

	printf("\n\n Persoane introduse: %d\n", n);
	for(i=0;i<n;i++) {
		printf("\t");
		AfisDatePers(&dp[i]);
	}
	_getch();
}
```

```
void CitDatePers(struct date_pers *p) // la apel p <-- &dp[i]
{
	printf("\nNumele: ");
	scanf("%s", p->nume);
	printf("\nPrenumele: ");
	scanf("%s", p->prenume);
	printf("\nCod: ");
	scanf("%ld", &p->cod);
	printf("\nData nasterii: ");
	printf("\n\tZiua: ");
	scanf("%d", &(p->data_nast).ziua);
	printf("\n\tLuna: ");
	scanf("%s", (p->data_nast).luna);
	printf("\n\tAnul: ");
	scanf("%d", &(p->data_nast).anul);
}

void AfisDatePers(struct date_pers *p)
{
	printf("\n Numele: %s", p->nume);
	printf("\n Prenumele: %s", p->prenume);
	printf("\n Cod: %d", p->cod);
	printf("\n Data nasterii: ");
	printf("\n\t Ziua: %d", (p->data_nast).ziua);
	printf("\n\t Luna: %s", (p->data_nast).luna);
	printf("\n\t Anul: %d\n", (p->data_nast).anul);
}
/*------------------------------------------------------------*/
```

Exemplul 4.

```
/* Programul preia de la tastatura datele pentru n persoane (nume,
prenume, data nasterii, codul personal), apoi le afiseaza, folosind
alocarea dinamica pentru cele n structuri de date de acelasi tip.*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <conio.h>

struct data_calend {
	int ziua;
	char luna[11];  // tablou rezervat pt. max. 11 caractere
	int anul;
};

struct date_pers {
	char nume[16];
	char prenume[20];
	long cod;  // 4
	struct data_calend data_nast;
};

void CitDatePers(struct date_pers *); // parametru formal de tip
pointer la structura
void AfisDatePers(struct date_pers *);

void main(void)
{
struct date_pers *dp; // declararea pointerului dp
int i, n;
```

```
	printf("\nNumar angajati : ");
	scanf("%d", &n);
	if(n < 0)  {
		printf("\n Numar invalid (negativ) !");
		_getch();
		return;
	}

printf("lungime data_calend %d\n", sizeof(struct data_calend)); // 20
printf("lungime date_pers %d\n", sizeof(struct date_pers));//60octeti

	if (!(dp = new struct date_pers [n])) { //initializare ptr. dp
			printf("Alocare nereusita!");
			_getch();
			return;
	}

	printf("\n Introduceti datele personale :");
	for(i=0;i<n;i++) {
		printf("\n Persoana %d:", i+1);
		CitDatePers(dp+i);
	}

	printf("\n\n Persoane introduse: ");
	for(i=0;i<n;i++) {
		printf("\n\t"); // printf("\n\t dp+i = %d", dp+i);
		AfisDatePers(dp+i);
	}
	delete [] dp; //eliberarea memoriei alocate
		_getch();
}

void CitDatePers(struct date_pers *p)
{
	printf("\nNumele: ");
	scanf("%s", p->nume); // campul nume e un tablou
	printf("\nPrenumele: ");
	scanf("%s", p->prenume);
	printf("\nCod: ");
	scanf("%d", &p->cod);
	printf("\nData nasterii: ");
	printf("\n\tZiua: ");
	scanf("%d", &(p->data_nast).ziua);
	printf("\n\tLuna: ");
	scanf("%s", (p->data_nast).luna);
	printf("\n\tAnul: ");
	scanf("%d", &(p->data_nast).anul);
}

void AfisDatePers(struct date_pers *p)
{
	printf("\n Numele: %s", p->nume);
	printf("\n Prenumele: %s", p->prenume);
	printf("\n Cod: %d", p->cod);
	printf("\n Data nasterii: ");
	printf("\n\t Ziua: %d", (p->data_nast).ziua);
	printf("\n\t Luna: %s", (p->data_nast).luna);
	printf("\n\t Anul: %d\n", (p->data_nast).anul);
}
```

4. Întrebări:

1. Cum se numesc elementele unei structuri ?
2. Cum se face accesul la elementele unei structuri ?
3. Ce sunt structurile imbricate?
4. Cum se initializeaza campurile unei structuri?

5. Teme:

1. Să se scrie un program C/C++, care folosind o structură numita *student,* avand campurile {nume, prenume, ţara de origine, grupa, anul naşterii}, să determine numărul de studenţi străini din grupă (grupa de *MAX* studenti) şi să afişeze toate datele acestora. Datele pentru studenţii din grupa se citesc de la intrarea standard, până la întâlnirea numelui AAA.
2. Să se scrie un program C/C++, în care folosind câte o funcţie se transferă ca parametru o variabilă de tip structură de date, ca valoare şi, respectiv, prin adresă folosind pointeri. În funcţia *main()* iniţializaţi câmpurile structurii cu date citite de la tastatură. În ambele funcţii afişaţi datele din structură folosind un mesaj adecvat.
3. Să se scrie un program C/C++, în care o funcţie returnează o structură de date adecvată. În acest fel vor fi returnate mai multe valori. Afişaţi rezultatul, valorile iniţiale transferate funcţiei (puteţi realiza orice operaţie în cadrul acelei funcţii), cu mesaje adecvate.
4. Să se scrie o aplicaţie C/C++, care utilizând o structură de tip *angajat* să afişeze toate datele persoanelor cu ocupaţia inginer, dintr-o întreprindere (nume, prenume, ocupaţia, data naşterii, secţia în care lucrează).
5. Să se scrie un program care citeşte datele personale pentru *n* persoane (nume, prenume, data naşterii, codul numeric personal, data angajării) şi apoi le afişează în ordinea datei angajării.
6. Să se scrie un program care afişează numele, prenumele şi media studentului cu cele mai bune rezultate din grupă în urma sesiunii de iarnă. Folosiţi pentru aceasta un tablou de structuri, de un tip numit *student,* alocarea dinamica, şi o funcţie de care returnează înregistrarea student cu media cea mai mare.
7. Să se scrie o aplicaţie C/C++, care alocă dinamic memorie pentru datele a *n* studenţi, (nume, prenume si gen), citeşte datele pentru fiecare din aceştia, afişează numărul studentelor si eliberează memoria alocată.
8. Declaraţi un nou tip de date *o_struct*, care să conţină o variabilă de tip întreg, una de tip caracter şi un şir de 256 de caractere. Definiţi in main() o variabilă statică de tip *o_struct*, căreia să-i iniţializaţi variabilele cu date citite de la intrarea standard. Declaraţi apoi un pointer de tip *o_struct*, numit *po_struct*, pe care să-l definiţi prin alocarea dinamică a unei zone de memorie care să conţină un articol de tip *o_struct*. Iniţializaţi câmpurile structurii de date folosind pointerul *po_struct*. Afisati toate campurile si eliberaţi zona de memorie alocată.
9. Scrieti o aplicaţie C/C++, care alocă dinamic memorie pentru memorarea datelor a *n* produse, folosind o structură *Produs,* cu câmpurile *denumire, pret, cantitate,* citeşte datele pentru fiecare dintre produse si afişează produsul din care avem cel mai mult pe stoc. În final va elibera memoria alocată.
10. Să se definească o structură cu numele *Masina,* care are câmpurile *producator, anul fabrictiei, capacitatea_cilindric* si *culoare*. Să se aloce dinamic memorie pentru n date de tip Maşina şi să se citească informaţiile pentru acestea. Să se afişeze inregistrarile maşinilor de culoare roşie, fabricate după anul 2010.

5'. Homework:

1. Develop a C/C++ application considering an adequate data structure named *student* having the fields: *name, surname, country, group* and *birth_year*. Count all the non-Romanian students from the group (*MAX* students in the group). The effective fields will be introduced from the keyboard generating an array of structures. A name AAA (upper or lower case) will finish the introduction process.
2. Develop a C/C++ application considering an adequate data structure that will be transferred by value as a parameter to a function, and then by address, using pointers to other function. Please initialise the fields of the structure within the *main()* function with data from the keyboard. In both functions, display the field's values with an adequate message.
3. Develop a C/C++ application considering an adequate data structure that will be returned by a function. In this way more values can be returned. Display the results, the initial values transferred to the function (doing whatever operation inside the function) using adequate messages.
4. Using included structures, *data_calend* with fields *day, month, year* and *personal_data* with fields *name, surname, occupation, code, department, birth_date* and *empl_date* of type *data_calend*, generate an array of structures of type *personal_data*, containing couple of employees (max. 20), reading their data from the keyboard. Considering "engineer", "teacher", "student" and "manager" as possible values for the field *occupation*, display all engineer's records.
5. Using the previous array of structures, generate a list of records being sorted in ascending order by their *code*, and in descending order by the *empl_date*.
6. Develop a C/C++ program that displays the name, surname, and media of the student with the best results in the group after the winter exams. Define a user-type table of structures named *student*, using dynamic memory allocation and a function that will return the record of the best student.
7. Write a C/C++ application that allocates dynamically memory for the data of *n* students (surname, name, gendre), reading from the keyboard all the required info, the program displays the number of female students and frees up the allocated memory.
8. Declare a structure named *a_structure* that contains as fields one integer- and one character-type variable and an array of 256 characters. Define in the main function a variable of type *a_structure* and initialize it's fielsd with data read from the keyboard. Declare a pointer, named *pa_structure* and initialize-it by allocating memory for a single variable of type *a_structure.* Use this pointer define all the fields of your variable with data read from the keyboard. Display all the fields of the structure, then free up the allocated memory.
9. Write a C/C++ application that allocates the necessary amount of memory for storing *n* products, using a structure named *Product* having the fields: *name, price, quantity*. After reading from the keyboard each product's data, display the item that has the biggest stock value. Free up the allocated memory.
10. Define a structure named *Car* that contains the following variables: *producer, production_year, cylinder_volume* and *colour*. Store in a newly allocated zone of memory the data for *n* cars. Display the records for the red cars produced after 2010.