

12. Structuri. Structuri imbricate. Pointeri la structuri. Alte tipuri utilizator.

(Data structures. Imbricate structures. Pointers to structures. Other user data-types).

1. Obiective:

- Înțelegerea noțiunii de date definite de utilizator(user-datatypes)
- Înțelegerea modalității de lucru cu structuri: declarare, inițializări, acces la câmpuri
- Scrierea de programe folosind structuri simple și imbricate
- Exersarea accesului cu pointeri la structuri de date
- Declararea, utilizarea datelor utilizator diferite de cele de tip struct.
- Scrierea de programe folosind alte date utilizator.

1'. Objectives:

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

2. Rezumat:

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

```
struct [Nume_struct] {  
    lista_declarații;  
}[nume1,nume2,...,numen];
```

unde:

- *Nume_struct*, este un nou tip de date, tip definit de utilizator, cu construcția *struct*;
- *lista_declarații*, este o listă prin care se declară componentele unei structuri, putând conține elemente de forma *tip_i element_i*, unde *tip_i* este un tip admis de limbajul C inclusiv o nouă structură (în acest ultim caz avem *structuri imbricate*);
- *nume₁, ..., nume_n* este o listă de variabile/obiecte de tip *Nume_struct*, putând lipsi la definirea structurii, caz în care este obligatoriu să fie precizat *Nume_struct*.

Dacă avem *Nume_struct* și nu avem *nume_i* atunci putem defini ulterior alte variabile/obiecte de tip *Nume_struct* cu declarația:

```
struct Nume_struct nume_obj; //C/C++  
sau  
Nume_struct nume_obj; //C++
```

unde *Nume_struct* reprezintă un nou tip de dată, un tip utilizator.

Elementele unei structuri se numesc *câmpuri sau attribute*, iar grupările de elemente *variabile* de tip structura, sau *inregistrari*, sau *obiecte*.

Printre elementele unei structuri se pot găsi și alte structuri (inclusiv pointeri către structura însăși), care sunt referite din exterior spre interior:

Accesul la elementele unei structuri se face precizând numele variabilei/obiectului de tip structură, de exemplu *angajat*, și câmpul care ne interesează (inclusiv dacă e o structură imbricată), 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 și printr-un pointer la structură, folosind operatorul *->* (săgeată).

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

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

Structurile pot fi transferate ca și parametri funcțiilor.

O funcție 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 și orice pointer către orice alt tip de variabilă. Pointerul va conține adresa primei componente a structurii la care pointează.

Pointerii la structuri sunt utili când dorim să transmitem o structură unei funcții, prin intermediul lor putând să transmitem doar adresa structurii.

În corpul funcției accesul la câmpurile structurii se face prin intermediul pointerului *p*, astfel:

<i>(*p).zi</i>	<i>sau</i>	<i>p ->zi</i>
<i>(*p).luna</i>	<i>sau</i>	<i>p ->luna</i>
<i>(*p).an</i>	<i>sau</i>	<i>p ->an</i>

Câmpuri de biți (Bit fields)

Un *câmp de biți* este un câmp al unei structuri care cuprinde unul sau mai mulți biți adiacenți. Cu ajutorul câmpurilor de biți se pot *accesa prin nume*, unul sau mai mulți biți dintr-un octet sau cuvânt oferind o granularitate perfect adaptabilă la periferice inteligente (ca ceasul CMOS, controlere de comunicație, etc.).

O structură care conține câmpuri de biți se declară astfel:

```
struct [Nume_struct] {  
    tip [câmp1] : lungime1;  
    ...  
    tip [câmpn] : lungimen;  
} [nume1, nume2, ..., numem];
```

Exemplu:

```
struct oct_cda_82C54 {  
    unsigned char BCD : 1; // LSB  
    unsigned char ModDeLucru : 3;  
    unsigned char RW_LowHigh : 2;  
    unsigned char SlctChannel : 2;  
} oct_cda_timer1, oct_cda_timer2;
```

unde :

- *tip* este un cuvânt cheie, de obicei *int/unsigned (char, int)*, care înseamnă *lungime_i* reprezintă numărul de biți pentru câmpul respectiv și este o constantă cu valori între 0 și o valoare nu mai mare de dimensiunea cuvântului calculator (8 biți pt. *char*, 16 pt. *short int* și 32 pt. *int/long* pentru mediul Microsoft VC++1y/2z).

Reguli:

- Câmpurile se alocă de la primul câmp către ultimul începând cu biții de ordin inferior ai cuvântului (LSB)
- Un câmp de biți trebuie să se poată alocă într-un cuvânt calculator. Dacă un câmp nu se poate alocă în cuvântul curent, atunci compilatorul îl va alocă în cuvântul următor. Mai multe câmpuri de biți, dacă încap, vor fi păstrate în același cuvânt calculator. (De ex. listei de câmpuri cu lungimile 1, 3, 2, deja doi octeți). Dacă unul din câmpuri este *int* sau *unsigned int* câmpul de biți va fi alocat pe minim un cuvânt, adică patru octeți (bytes).
- Un câmp fără nume nu poate fi accesat. Acesta definește o zonă neutilizată dintr-un cuvânt.
- Lungimea în biți alocată pentru un câmp de biți o putem defini zero, dacă vrem ca următorul câmp de biți să fie alocat în următorul cuvânt-calculator.
- Compilatorul admite lungime 0 numai pentru câmpuri de biți fără nume. Un câmp fără nume nu poate fi accesat.
- Câmpurile de biți se pot referi folosind aceleași convenții ca și în cazul structurilor obișnuite: direct (prin operatorul *.*) sau indirect (prin operatorul *->*).

Observații:

- O structură poate îngloba câmpuri de biți împreună cu alte variabile de alte tipuri.
- Octetul fiind cea mai mică unitate de memorie adresabilă la calculatoarele compatibile IBM PC, adresa de memorie a unui câmp de biți nu se poate obține cu ajutorul operatorului de adresare *&*, deci nu se poate aplica la un câmp de biți.
- Câmpurile de biți sunt folosite când lucrăm cu multe date booleene, permițând folosirea eficientă a memoriei, 8 date booleene pe octet. Prelucrarea datelor pe biți se poate face și cu ajutorul operatorilor logici pe biți, care duc însă în general la un efort mai mare în programare comparativ cu câmpurile de biți (necesită instrucțiuni suplimentare cum ar fi deplasări, setări, și/sau mascări de biți).

Reuniuni (Unions)

Reuniunea este un tip de dată definit de utilizator, asemănător ca și construcție cu structura, dar în loc de cuvântul cheie *struct* este folosit cuvântul cheie *union* și la definirea unei variabile de acest tip nu se alocă memorie pentru toate câmpurile conținute de reuniune, ci doar memoria necesară câmpului cu dimensiunea cea mai mare. Putem spune că reuniunea este ca o structură în care toate câmpurile sunt stocate la aceeași adresă.

Exemplu:

```
union A {  
    int x;  
    long y;  
    double r;  
    char c;  
} u;
```

unde *A* este un nou tip de dată de tip reuniune, iar *u* este o variabilă/obiect de tip reuniune *A*.

Accesul la elementele *x*, *y*, *r*, *c* ale reuniunii se face ca și la structuri folosind operatorul *.* (*punct*) sau operatorul *->* (*săgeată*).

Asignări de nume pentru tipuri de date

În limbajul C se poate atribui un nou nume unui tip deja existent, fie el predefinit în limbaj sau definit de utilizator, cu o construcție de forma:

```
typedef tip nou_nume_tip;
```

unde:

- *tip*, este tipul existent;
- *nou_nume_tip*, este noul nume, care poate fi utilizat în continuare pentru a declara date de același tip.

Enumerări (Enumeration, enum)

Tipul *enumerare* permite programatorului să definească o listă de *constante întregi* cu nume, în vederea folosirii de nume sugestive pentru valori numerice.

Tipul enumerare se declară printr-un format asemănător cu cel utilizat în cadrul structurilor și anume:

```
enum [Nume_enum] {  
    numei [=consti],  
    ...  
} [en1, en2, ..., enn];
```

unde:

- *Nume_enum*, este numele noului tip de date utilizator introdus prin această declarație
- *nume_i*, sunt nume care se vor utiliza în locul valorilor numerice și au valori implicite: prima are valoarea 0, iar restul au valoarea constantei precedente plus 1 (*nume_i* va avea valoarea *i*);
- *const_i* este constantă de inițializare a unui element; dacă este prezentă atunci:

$$\text{nume}_i = \text{const}_i, \text{ nume}_{i+1} = \text{const}_i + 1, \dots$$

de exemplu: `enum Months {ian = 1, feb, mar,, dec}; //feb = 2`

- *en₁, en₂, ..., en_n*, sunt variabile *enum* de noul tip *nume_enum*.

În limbajul C, unei variabile enumerare i se poate atribui o valoare întreagă și se pot face operații cu ele. În limbajul C++, unei variabile enumerare i se pot atribui numai constante de enumerare.

Tipul enumerare se folosește pentru variabile care pot lua un număr redus de valori întregi prin asocierea unor nume sugestive fiecărei valori.

3. Exemple:

Exemplul 1.

```
/* Citeste de la consolă elementele unei structuri Data_calend și  
afiseaza data citita în alt format : zi/luna/an. */  
#define _CRT_SECURE_NO_WARNINGS  
#include <stdio.h>  
#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
const char* tab[] = { "ian", "feb", "mar", "apr", "mai", "iun",
"iul", "aug", "sep", "oct", "noi", "dec" };

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

int main(void) {
    struct Data_calend dc = {}; //initializare obiect
    printf("\n Data initiala este :\n %d/%p/%d", dc.ziua, dc.luna,
dc.anul);
    //struct Data_calend udc; //declarare obiect - permis
    //printf("\n Data initiala este :\n %d/%p/%d", udc.ziua,
udc.luna, udc.anul); -nepermis
    int i, luna = 0; // luna e in 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_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_exit("\nLuna incorecta !");
    printf("\n\t anul : ");
    scanf("%d", &dc.anul);
    if (dc.anul <= 0) puts_exit("\nAnul incorect (negativ)!");
    printf("\n Data introdusa este :\n %d/%d/%d", dc.ziua, luna,
dc.anul);
}

void puts_exit(const char* ptr) {
    puts(ptr); // printf(ptr); -- de asemenea e OK
    exit(0);
}

/*-----*/

```

Exemplul 2.

/* Programul declara local in functia main() un nou tip de date, Forma, si defineste o variabilă de acest tip, cerc. Se acceseaza campurile structurii prin metoda "nume calificat" */

```

#include <stdio.h>

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

```

```

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

// definim ulterior o alta variabila-structura de acelasi tip,
"forma", ca si "cerc"
struct Forma sfera = { 3, 1, 5.f }; // nu i-am initializat
toate campurile la declarare
sfera.suprafata = 4 * 3.14f * sfera.raza * sfera.raza;
printf("\nInfo sfera:\n");
printf("sfera.tip: %d\n", sfera.tip);
printf("sfera.culoare: %d\n", sfera.culoare);
printf("sfera.raza: %g\n", sfera.raza);
printf("sfera.suprafata: %g\n", sfera.suprafata); // 314
printf("sfera.perimetru: %g\n", sfera.perimetru); // camp
neinitilaizat
return 0;
}
/* Rulare:
cerc.tip: 2
cerc.culoare: 1
cerc.raza: 5
cerc.suprafata: 78.37
cerc.perimetru: 31.42
sfera.suprafata: 314
sfera.perimetru: 0
*/
/*-----*/

```

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>

#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 *);

```

```

int 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 !");
        return 1;
    }
    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]);
    }
    return 0;
}

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. Se

sorteaza dupa un camp, respectiv doua campuri si se afiseaza - Se foloseste abordarea prin ADT - Abstract Data Type*/

```
//DatePers.h

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

struct Date_pers {
    char nume[16];
    char prenume[20];
    long cod;
    struct Data_calend data_nast;
    void citDatePers(Date_pers*); //methods inside struct
    void afisDatePers(Date_pers*);
};

void afisDatePers(Date_pers* p) {
    cout << "\n Numele: " << p->nume;
    cout << "\n Prenumele: " << p->prenume;
    cout << "\n Cod: " << p->cod;
    cout << "\n Data nasterii: ";
    cout << "\n\t Ziua: " << (p->data_nast).ziua;
    cout << "\n\t Luna: " << (p->data_nast).luna;
    cout << "\n\t Anul: " << (p->data_nast).anul;
}

void citDatePers(Date_pers* p) {
    cout << "\nNumele: ";
    cin >> p->nume;
    cout << "\nPrenumele: ";
    cin >> p->prenume;
    cout << "\nCod: ";
    cin >> p->cod;
    cout << "\nData nasterii: ";
    cout << "\n\tZiua: ";
    cin >> (p->data_nast).ziua;
    cout << "\n\tLuna: ";
    cin >> (p->data_nast).luna;
    cout << "\n\tAnul: ";
    cin >> (p->data_nast).anul;
}

//main()

#include <iostream>
using namespace std;
#include "DatePers.h"
int comp_cod(const void* a, const void* b);
int comp_nume_an(const void* a, const void* b);

int main() {
    struct Date_pers* dp; // declararea pointerului dp
    int i, n;
    cout << "\nNumar angajati : ";
    cin >> n;
    if (n <= 0) { cout << "\n Numar invalid (negativ sau zero) !"
    << endl; exit(1); }
}
```



```

//cout <<"lungime Data_calend: \n"<<sizeof(Data_calend)<<endl;
//cout <<"lungime Date_pers: \n"<< sizeof(Date_pers)<<endl;
if (!(dp = new Date_pers[n])) { //initializare ptr. dp
    cout << "Alocare nereusita!"; exit(1);    }
cout << "\n Introduceti datele personale :";
for (i = 0; i < n; i++) {
    cout << "\n Persoana:" << i+1;
    citDatePers(dp + i);
}
cout << "\nPersoane introduse: ";
for (i = 0; i < n; i++) afisDatePers(dp + i);
cout << "\n\nPersoane sortate dupa cod: ";
qsort(dp, n, sizeof(Date_pers), comp_cod);
for (i = 0; i < n; i++) afisDatePers(dp + i);
cout << "\n\nPersoane sortate dupa nume si an: ";
qsort(dp, n, sizeof(Date_pers), comp_nume_an);
for (i = 0; i < n; i++) afisDatePers(dp + i);
delete[] dp; //eliberarea memoriei alocate
}

int comp_cod(const void* a, const void* b) {
    Date_pers* pa = (Date_pers*)a;
    Date_pers* pb = (Date_pers*)b;
    return (pb->cod - pa->cod); //descendent
}
int comp_nume_an(const void* a, const void* b) {
    int fl_nume;
    Date_pers* pa = (Date_pers*)a;
    Date_pers* pb = (Date_pers*)b;
    if((fl_nume=strcmp(pa->nume, pb->nume))==0)
        return ((pa->data_nast).anul - (pb->data_nast).anul);
    //ascendent_nume si an
    return fl_nume;
}

```

Exemplul 5.

/* Programul preia de la tastatura valori reale pe care le stocheaza intr-un tablou dimensional si foloseste o structura pentru a returna sumele valorilor pozitive, respectiv negative din tablou */

```

#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#define MAX 20

struct Sume {
    float sumaPoz;
    float sumaNeg;
};

Sume calculSume(float [], int);

int main()
{
    int n;
    float tab[MAX];
    printf("\n Cate valori doriti sa cititi? ");
    scanf("%d", &n);
    if(n<=0 || n>20)
    {
        printf("\n Valoare gresita pentru numarul de valori");
    }
}

```

```

        return 1;
    }
    printf("\nIntroduceti valorile:\n");
    for (int i = 0; i < n; i++)
    {
        printf("tab[%d]=", i);
        scanf("%f", &tab[i]);
    }
    Sume rez = calculSume(tab, n);
    printf("Suma elementelor pozitive este %.2f si a elementelor negative este %.2f", rez.sumaPoz, rez.sumaNeg);
}

Sume calculSume(float x[], int n) {
    Sume s = {};
    for (int i = 0; i < n; i++){
        if (x[i] >= 0) s.sumaPoz += x[i];
        else s.sumaNeg += x[i];
    }
    return s;
}

```

Câmpuri de biți (bit fields)

Programul folosește structura *SYSTEMTIME* a cărei declarație este prezentată în continuare. Pentru a putea accesa structura mai sus menționată trebuie inclus fișierul header *Windows.h*.

```

typedef struct _SYSTEMTIME {
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME;

```

Pentru a putea accesa structura de mai sus trebuie inclus fișierul header *Windows.h*.

Exemplu bitfields

```

#include <iostream>
using namespace std;
#include <Windows.h>

const char* getWeekday(unsigned short); //traduce nr.-ul zilei in text

struct Date {
    unsigned short nWeekDay : 3;    // 0..7   (3 biti)
    unsigned short nMonthDay : 5;    // 0..31  (5 biti)
    unsigned short nMonth : 4;       // 0..15  (4 biti)
    unsigned short nYear : 11;       // 0..2047 (11 biti)
};

int main() {
    //declararea unui pointer de tip SYSTEMTIME
    Date date;
    SYSTEMTIME* p_st = new SYSTEMTIME;
    //popularea elementelor structurii SYSTEMTIME
    GetSystemTime(p_st);
}

```

```

//GetLocalTime(p_st); // Exista si o functie pt. ora locala
//afisarea datelor stocate in SYSTEMTIME
cout << "Anul curent: " << p_st->wYear << endl;
cout << "Luna curenta: " << p_st->wMonth << endl;
cout << "Ziua curenta: " << p_st->wDay << endl;
cout << "Ora curenta (GMT): " << p_st->wHour << endl;
cout << "Minutul curent: " << p_st->wMinute << endl;
cout << "Numarul curent de secunde: " << p_st->wSecond << endl;
cout << "Numarul curent de milisecunde: " << p_st->
wMilliseconds << endl;
printf("\nOra mai poate fi afisata in formatul:
%02d:%02d:%02d", p_st->wHour, p_st->wMinute, p_st->wSecond);
//copierea datelor din SYSTEMTIME in Date
date.nYear = p_st->wYear;
date.nMonth = p_st->wMonth;
date.nMonthDay = p_st->wDay;
date.nWeekDay = p_st->wDayOfWeek;
//afisarea datelor din Date
cout<<"\nData afisata cu valorile din Date:\n";
cout<<date.nMonthDay<< "(" << date.nWeekDay << ")." << date.nMonth << ".
" << date.nYear << endl;
//afisarea datelor din Date cu ziua ca si text
cout << getWeekday(date.nWeekDay) << ", " << date.nMonthDay << "." <<
date.nMonth << "." << date.nYear << endl;
delete p_st;
return 0;
}

const char* getWeekday(unsigned short nWeekday) {
const char* tsc[] = { "Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday" };
// tabloul de pointeri catre sirurile de caractere.-e este
local in fc.-tie, persista dupa ret.
return tsc[nWeekday]; // sau return *(tsc + nWeekday);
};
/*
Anul curent: 2020
Luna curenta: 12
Ziua curenta: 14
Ora curenta (GMT): 14
Minutul curent: 39
Numarul curent de secunde: 37
Numarul curent de milisecunde: 522

Ora mai poate fi afisata in formatul: 14:39:37
Data afisata cu valorile din Date:
14(1).12.2020
Monday, 14.12.2020
Press any key to continue . . .*/

```

Reuniuni

Exemplul R1

```

#include <iostream>
using namespace std;

int main(void)
{
// reuniune anonima
union {
int data1;

```

```

        float data2;
};

    data1 = 3;
    cout << "Valoarea din campul data1 este: " << data1;

    data2 = 1.2345;
    cout << "\nValoarea din campul data2 este: " << data2;

    cout << "\nInca o data Valoarea din campul data1 este: " <<
data1 << endl;
    return 0;
}
/*
Valoarea din campul data1 este: 3
Valoarea din campul data2 este: 1.2345
Inca o data Valoarea din campul data1 este: 1067320345
Press any key to continue . . .
*/

```

Exemplul R2

```

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string.h>

using namespace std;

int main(void)
{
    union {
        long l;
        double d;
        char s[4];
    } vr;

    vr.l = 100000;
    cout << vr.l << " ";

    vr.d = 123.2342;
    cout << vr.d << " ";

    strcpy(vr.s, "hi\n");
    cout << vr.s;
    return 0;
}
/*
100000 123.234 hi
*/

```

Exemplul R3

```

#include <stdio.h>

int main(void) {
    union EmployeeDates {
        int days_worked;

        struct Date {
            int month;
            int day;
            int year;
        } last_day;
    } emp_info;
}

```

```

union Numbers {
    int a;
    float b;
    long c;
    double d; // cel mai mare camp
} value;

printf("Dimensiunea reuniunii EmployeeDates este de: %d octeti.\n",
sizeof(emp_info));
printf("Dimensiunea reuniunii Numbers este de: %d octeti.\n",
sizeof(value));
return 0 ;
}
/*Dimensiunea reuniunii EmployeeDates este de: 12 octeti.
Dimensiunea reuniunii Numbers este de: 8 octeti.
*/

```

Enumerari

Exemplul E1

```

#include <stdio.h>

int main(void){
enum Weekdays { Luni=1, Marti, Miercuri, Joi, Vineri, Sambata };

printf("%d %d %d %d %d %d\n", Luni, Marti, Miercuri, Joi, Vineri,
Sambata);
return 0;
}
/*1 2 3 4 5 6
*/

```

Exemplul E2

```

#include <stdio.h>

int main(void){
enum Weekdays {
    Luni = 10,
    Marti = 20,
    Miercuri = 30,
    Joi = 40,
    Vineri = 50,
    Sambata = 60
};

printf("%d %d %d %d %d %d\n", Luni, Marti, Miercuri, Joi,
Vineri, Sambata);
return 0;
}
/*10 20 30 40 50 60
*/

```

Exemplul E3

```

//programul face translatia între formatul prescurtat și cel lung al
lunilor
#include <iostream>
using namespace std;

enum Month { Ian=1, Feb, Mar, Apr, Mai, Iun, Iul, Aug, Sep, Oct, Noi,
Dec };

```

```

const char* MonthStr(Month month);

int main(void) {
    cout << "\nLuna " << Aug << " este: " << MonthStr(Aug) << '\n';
    cout << "\nLuna " << Dec << " este: " << MonthStr(Dec) << '\n';
    return 0;
}

const char* MonthStr(Month month)
{
    switch (month) {
        case Ian:    return "Ianuarie";
        case Feb:    return "Februarie";
        case Mar:    return "Martie";
        case Apr:    return "Aprilie";
        case Mai:    return "Mai";
        case Iun:    return "Iunie";
        case Iul:    return "Iulie";
        case Aug:    return "August";
        case Sep:    return "Septembrie";
        case Oct:    return "Octombrie";
        case Noi:    return "Noiembrie";
        case Dec:    return "Decembrie";
        default:     return " ";
    }
}

```

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. Ce înțelegeți printr-un câmp de biți?
6. Cum se pot referi câmpurile de biți?
7. Care sunt diferențele dintre structuri și reuniuni?
8. Ce înțelegeți prin tipul enumerare?
9. Cum se poate asigna un nume unui tip de date?

5. Teme:

1. Să se scrie un program C/C++, care folosind o structură numita *Student*, având campurile {nume, prenume, țara de origine, grupa, anul nașterii}, să determine numărul de studenți străini din grupă (grupa de *MAX* studenti) și să afișeze toate datele acestora. Datele pentru studenții 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 funcție, se transferă ca parametru o variabilă de tip structură de date prin valoare și respectiv prin adresă, folosind pointeri. În funcția *main()* inițializați câmpurile structurii cu date citite de la tastatură. În ambele funcții afișați datele din structură folosind un mesaj adecvat.
3. Să se scrie un program C/C++, în care o funcție returnează o structură de date adecvată. În acest fel vor fi returnate mai multe valori. Afișați rezultatul, valorile inițiale transferate funcției (puteți realiza orice operație în cadrul acelei funcții), cu mesaje adecvate.
4. Să se scrie o aplicație C/C++, care utilizând o structură de tip *Angajat*, să afișeze toate datele persoanelor cu ocupația inginer dintr-o întreprindere (nume, prenume, ocupația, data nașterii, secția în care lucrează).

5. Folosind structura de la exemplul precedent , să se scrie un program care citește datele personale pentru n persoane (nume, prenume, data nașterii, codul numeric personal, data angajării) și apoi le afișează în ordinea datei angajării.
6. Să se scrie un program care afișează numele, prenumele și media studentului cu cele mai bune rezultate din grupă în urma sesiunii de iarnă. Folosiți pentru aceasta un tablou de structuri, de un tip numit *Student*, alocarea dinamica, și o funcție care returnează înregistrarea student cu media cea mai mare.
7. Să se scrie o aplicație C/C++, care alocă dinamic memorie pentru datele a n studenți, (nume, prenume și gen), citește datele pentru fiecare din aceștia, afișează numărul studentelor și eliberează memoria alocată.
8. Declarați un nou tip de date *O_struct*, care să conțină o variabilă de tip întreg, una de tip caracter și un șir de 256 de caractere. Definiți în *main()* o variabilă statică de tip *O_struct*, căreia să-i inițializați câmpurile cu date citite de la intrarea standard. Declarați apoi un pointer de tip *O_struct*, numit *po_struct*, care va fi definit prin alocarea dinamică a unei zone de memorie care să conțină un articol de tip *O_struct*. Inițializați câmpurile structurii de date folosind pointerul *po_struct*. Afișați toate câmpurile și eliberați zona de memorie alocată.
9. Scrieți o aplicație C/C++, care alocă dinamic memorie pentru memorarea datelor a n produse, folosind o structură *Produs*, cu câmpurile *denumire*, *pret*, *cantitate*, citește datele pentru fiecare dintre produse și afișează 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 fabricției*, *capacitatea_cilindrică* și *culoare*. Să se aloce dinamic memorie pentru n date de tip *Mașina* și să se citească informațiile pentru acestea. Să se afișeze înregistrările mașinilor de culoare roșie, fabricate după anul 2010.
11. Folosind structura de date union denumită *grup* compusă din diferite câmpuri (int, long, double, char etc.). Scrieți o aplicație C/C++ care va inițializa un element de tipul *grup* de la tastatură. Este posibil să afișăm în același timp toate câmpurile folosind pointeri sau nume calificate? Afișați ceea ce este posibil și dimensiunea elementului union. Realizați aceeași operație considerând o simplă structură struct.
12. Definiți o dată de tip enumerare enum *Lumina_Alba* care va avea în componență culorile de bază (Roșu, Portocaliu, Galben, Verde, Albastru, Indigo și Violet). Inițializați câteva variabile de tip *Lumină_Alba*. Urmăriți să generați culori secundare cu ajutorul operațiilor secifce enum prin combinații ale culorilor de bază. Traduceți printr-un mecanism folosind enumerările, numele culorilor în limba Franceză, Engleză sau Germană și afișați toate culorile, menționând numele inițial, noul nume și valoarea asociată.

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 fields 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.
11. Write a C/C++ application that defines a union called *group* that contains various fields (int, log, double, char, etc.). Read from the keyboard the data associated to a *group* variable. Is it possible to display all the fields in the union using pointers or fully qualified field-names? Display the accessible information and the dimension of the union variable. Perform the same operations considering a regular structure instead of the union.
12. Define an enumeration called *White_Light* that will contain the basic colors (red, orange, yellow, green, blue, indigo and violet). Initialize a few variables of type *White_Light* and try to generate some secondary colors by combining the basic colors mentioned above. Use an enumeration based mechanism for translating the color names into French, English or German. Display all the colors mentioning the original- and the translated names and the associated value.