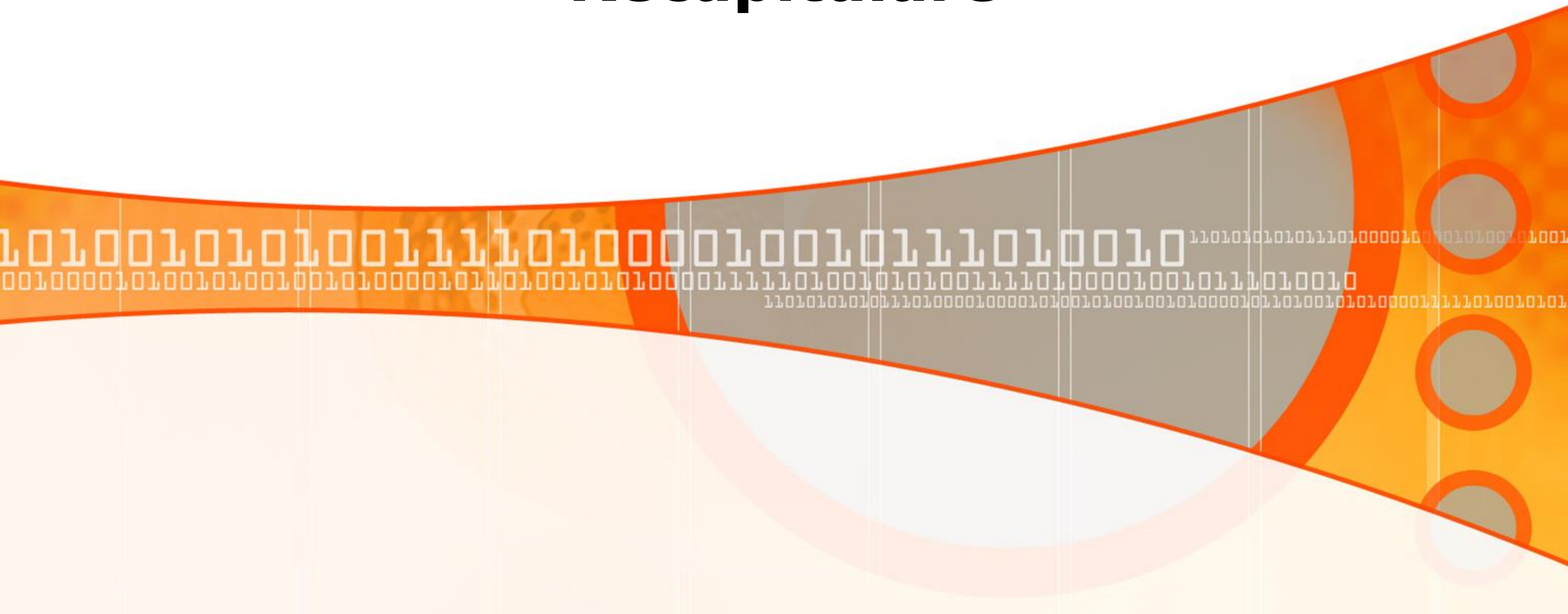
STRUCTURI DE DATE

Recapitulare



EVALUARE

- **SEMESTRU: 4 puncte**
  - **Testare cunostinte calculator/scris/oral: 3 puncte**
  - **Proiect elaborat si sustinut individual: 1 punct; cerinte publicate pe acs.ase.ro/data-structures**

EVALUARE

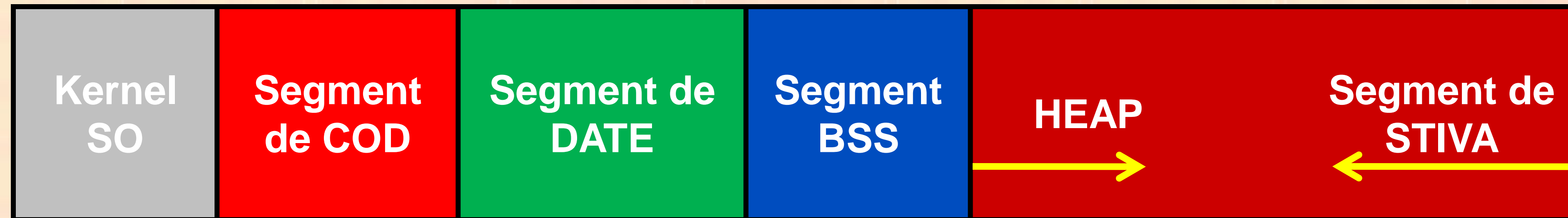
- **EXAMEN: 6 puncte**
  - **Test cunostinte (pe calculator): 1 punct;**
  - **Test PRACTIC (pe calculator): 5 puncte;**
  - **Punctaje acordate DOAR pentru aplicatii "duse" pana in faza de executie: compilare, exemple de test (inclusiv valori "extreme" pentru validarea datelor de I/O);**
  - **Acumulare minim 50% (3 puncte) din punctajul maxim posibil al examenului.**

BIBLIOGRAFIE

- Ion Ivan, Marius Popa, Paul Pocatilu (coord.) - “Structuri de date”, vol. 1, vol. 2, 2009
- Ion Smeureanu – “Programarea in limbajul C/C++”, Editura CISON, 2001
- Bjarne Strastroup – The Creator of C++, “The C++ Programming Language” – 3rd Edition, Editura Addison-Wesley
- http://www.acs.ase.ro
- http://www.itcsolutions.eu

MEMORIE

Organizarea memoriei la executia unui proces



Adrese mici

Adrese mari

- **Segment de COD**

- Instructiuni executabile (binare);

- Read-Only – nu pot fi scrise secvente binare, deci nu poate fi o bresa de securitate (generare eroare acces memorie);

- Partajat intre utilizatori care executa concurent codul binar al aplicatiei.

MEMORIE

• **Segment de DATE**:

- Date alocate static (clasa de memorie) si date globale initializate cu valori in codul aplicatiei;

- Fiecare proces contine propriul segment de date;

- Nu este segment executabil.

• **Segment BSS (Block Started by Symbol)**:

- Date alocate static (clasa de memorie) si date globale neinitializate prin codul aplicatiei (implicit au valori nule);

- Nu este segment executabil.

MEMORIE

• **Segment STACK**:

- Variabile locale – definite in functii;

- Transfer parametri in functii;

- Localizata la capatul memoriei accesibile => alocare descrescatoare a adreselor;

- Management stack segment – registrul **SP (Stack Pointer** procesor de 16 biti) / **ESP (Extended Stack Pointer** procesor de 32 biti) / **RSP (Register Stack Pointer** procesor 64 biti);

- Stocare date la nivel de cuvant calculator – unitate de date de lungime fixa folosita de procesor conform setului de instructiuni ale acestuia. Registrii procesorului sunt lungime cuvant calculator (8, 16, 24, 32 sau 64 biti).

- **Segment HEAP**:

- Alocare memorie pe durata de executie a codului binar;

- Alocare gestionata de sistemul de operare;

- Zone de memorie gestionate prin variabile de tip pointer.

MEMORIE

- **HEAP**

- Memorie alocata dinamic;
- Accesata prin pointeri si are un continut anonim;
- Dimensiune pana la pointerul break;
- Posibilitate de crestere a dimensiunii prin mutarea pointerului spre adrese mai mari;

- **STIVA**

– Utilizata pentru variabilele locale, valori temporare, argumente functii, adrese de return.

MEMORIE

Clase de memorie ale variabilelor:

• **AUTOMATICE** (specificator **auto)**:

- Locale pentru blocul de instructiuni in care se definesc variabilele;
- Persistente pana la terminarea blocului de instructiuni in care se definesc;
- Zone de memorie distincte pentru cod recursiv sau multithreading;
- Stocate in segmentul de stiva;

Exemple:

```
auto a = 7;
auto b = "VariabileAuto";
```

Clase de memorie ale variabilelor:

• **REGISTRU** (specificator **register)**:

- Specificator utilizat doar pentru variabile locale si parametri ai functiilor;
- Persistente pana la terminarea blocului de instructiuni in care se definesc (similar specificatorului auto);
- Decizia compilatorului de incarcare a unui registru cu continut variabila;
- Utile pentru operatii frecvente din punctul de vedere al reducerii timpului de acces si executie;

Exemplu:

```
register int vreg;
int d;
d = 8;
vreg = d;
```

MEMORIE

Clase de memorie ale variabilelor:

• **EXTERNE** (specificator **extern**):

- Utilizate pentru variabile declarate in mai multe fisiere sursa;
- Memorie alocata inainte de executia functiei main; persistenta pana la terminare executiei programului;
- Definite in bloc de instructiuni cu accesibilitate in cadrul blocului; altfel, accesibila la nivel de fisier sursa;

Exemplu:

Fisierul 1

```
extern int i;
void f() {
        i++;
}
```

Fisierul 2

```
int i = 0;
extern void f();
void g() {
        f();
        printf("%d\n", i);
}
```

MEMORIE

Clase de memorie ale variabilelor:

• **STATICE** (specificator **static**):

- Implicit pentru variabilele definite in afara tuturor blocurilor;
- Alocate la inceperea executiei programului si dezalocate la terminare executiei;
- Declararea intr-o functie asigura persistenta continutului intre apeluri;

Exemplu:

```
int f() {
  static int x = 0;
  x++;
  return x;
}

void main() {
  int j;
  for (j = 0; j < 10; j++) {
    printf("Rezultat functie f: %d\n", f());
  }
}
```

MEMORIE

Sursa C/C++

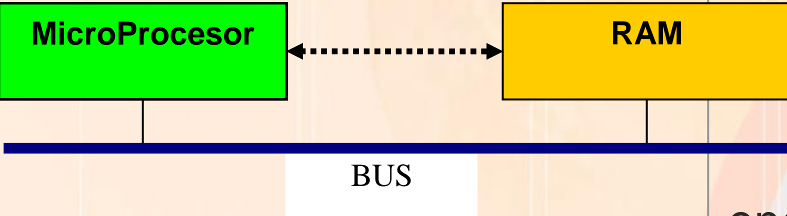
```
#include<stdio.h>

void main()
{
        char a = 7, b = 9;
        short int c;
        c = a+b;
}
```

Reprezentare ASM

```
.model small
.stack 16
.data
        a db 7
        b db 9
        c dw ?

.code
start:
        mov AX, @data
        mov DS, AX

        mov AL,a
        add AL,b
        mov c,AX

        mov AX, 4C00h
        int 21h
end start
```

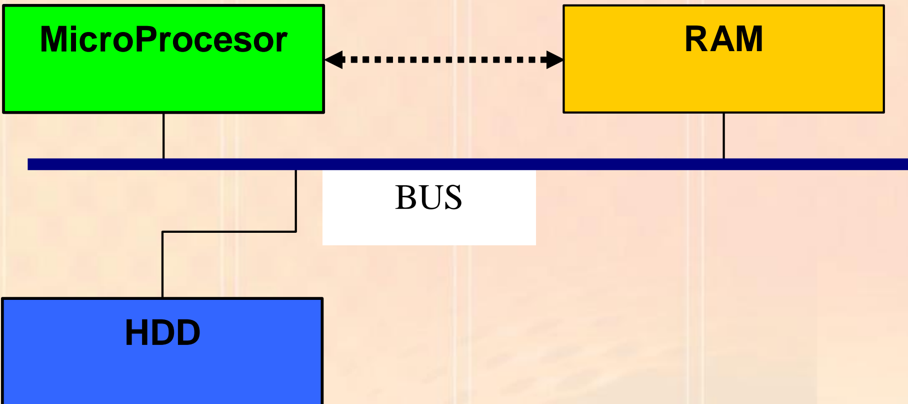
Cod Masina

```
B8 02 00 8E D8
A0 00 00 02 06
01 00 A3 02 00
B8 00 4C CD 21
00 00 00…..00 00
07 09
```

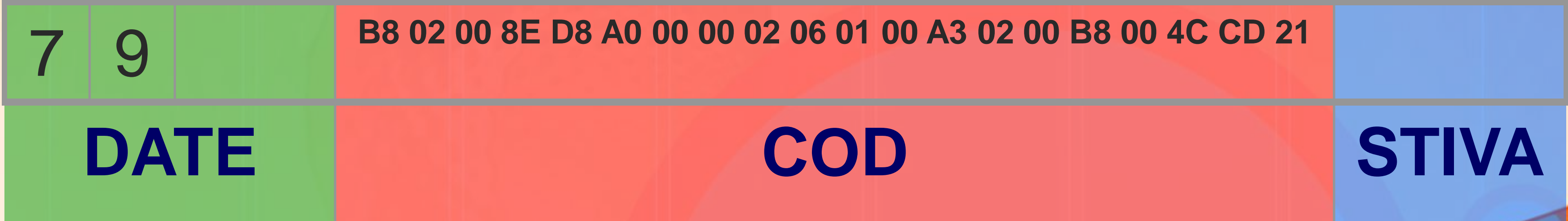


MEMORIE

Sursa C/C++




```
#include<stdio.h>

void main()
{
        char a = 7, b = 9;
        short int c;
        c = a+b;
}
```



Tipuri de date

Sursa definirii:

- Fundamentale, definite in cadrul limbajului;
- Definite de utilizator (programator/dezvoltator etc); exemple: structuri articol, clase de obiecte, structuri pe biti, uniuni etc.

Natura continutului:

- Simple, corespunzatoare tipului de date;
- Masive;
- Pointeri.

Tipuri de date

Descriere tipuri fundamentale:.

Denumire	Explicatie	Dimensiune in bytes	Interval valori posibile
char	Caracter sau intreg de valoare mica.	1 byte	cu semn: -128 la 127 fara semn: 0 la 255
short int (short)	Intreg short.	2 bytes	cu semn: -32768 la 32767 fara semn: 0 la 65535
int	Intreg.	4 bytes	cu semn: -2147483648 la 2147483647 fara semn: 0 la 4294967295
long int (long)	Intreg long.	4 bytes	cu semn: -2147483648 la 2147483647 fara semn: 0 to 4294967295
float	Real virgula mobila precizie simpla.	4 bytes	+/- 3.4e +/- 38
double	Real virgula mobila precizie dubla.	8 bytes	+/- 1.7e +/- 308
long double	Real virgula mobila precizie extinsa.	8 bytes / **10 bytes** / 16 bytes (functie de compilator)	

Reprezentare interna tipuri reale (lungimi zone in biti):

Denumire	Semn	Exponent	Mantisa	Total biti	Bias exponent
float	1	8	23	32	127
double	1	11	52	64	1023
long double	1	15	64	80	16383

POINTERI

- date numerice utilizate pentru a gestiona valori reprezentand adrese;
- dimensiune data de arhitectura procesorului
- definire:

*tip_data * nume_pointer;*

- initializare*:*

nume_pointer = & nume_variabila;

- utilizare*:*

*nume_variabila = * nume_pointer;*

Exemple:

- i n t * p i ; // pointer la int
- c h a r ** p p c ; // pointer la pointer de char
- i n t * a p [1 0]; // vector de 10 pointeri la int

Valoarea 0 pentru un pointer este o valoare nula. Aceasta este asociata cu simbolul

```
#define NULL 0
```

sau cu constanta

```
const int NULL = 0;
```

Aritmetica pointerilor:

- pentru un pointer de tip T*, operatorii --/++ asigura deplasarea inapoi/inainte cu sizeof(T) octeti;
- pentru un pointer de tip T* pt, expresia pt + k sau pt – k este echivalenta cu deplasarea peste k * sizeof(T) octeti;
- diferenta dintre 2 pointeri din interiorul aceluiasi sir de valori reprezinta numarul de elemente dintre cele doua adrese;
- adunarea dintre 2 pointeri nu este acceptata;

Pointeri constanti

Exemplu:

```
int * const p;                  // pointer constant la int
int const *pint;                // pointer la int constant
const int *pint2;               // pointer la int constant
const int * const pint2;        // pointer constant la int constant
```

Utilizare:

```
char* strcpy(char*p, const  char*q);
```

Alocare dinamica memorie

- operatorul ***new*** sau ***new [ ]***;
- rezerva memorie in ***HEAP***

Dezalocare memorie

- operatorul ***delete*** sau ***delete[ ]***;
- elibereaza memoria rezervata in ***HEAP***

Caracteristici

• secventa de cod sursa cu caracter general si repetitiv;

• primesc parametri de intrare si returneaza rezultate;

• imbricarea nu este permisa in C/C++;

• transferul parametrilor de intrare: valoare, adresa, variabile globale;

• parametri copiati in zone de memorie organizate ca stive;

• rezultatul returnat: tip de retur, argumente transmise prin adresa.

FUNCTII

Declararea si construirea unei functii:

TipRetur DenFunctie([ListaParametriFormali]); **Declarare antet functie**

Standard/Utilizator

Implicit int/void

Masiv NU!

Identificator functie

```
TipRetur DenFunctie([ListaParametriFormali]){
    // corp functie
}
```

- **declaratii locale**
- **instructiuni**
- **apeluri subprograme**
- **instructiune *return***

Parametrii formali sub forma

[tip$_i$ p$_i$[,…]]

Exemplu functie:

Sursa C/C++

```
#include<stdio.h>

double Suma1(float x, float y){
		double s;
		s = x + y;
		return s;
}
```

```
#include<stdio.h>

void Suma2(float x, float y, float *z){
		*z = x + y;
}
```

Apel subprograme C/C++

```
…
float a = 1.2, b = 4.7, c;
…
c = Suma1(a, b);
…
```

```
…
float a = 1.2, b = 4.7, c;
…
Suma2(a, b, &c);
…
```

POINTERI LA FUNCTII

- definire:

  tip_return (den_pointer) (lista_parametri);*

- initializare*:*

  den_pointer = den_functie;

- apel functie prin pointer*:*

  den_pointer (lista_parametri);

POINTERI LA FUNCTII

- f l o a t (*f p)(int *); // pointer la functie ce primeste un pointer la ***int*** si ce returneaza un ***float***

- i n t * f (c h a r *); // functie ce primeste ***char**** si returneaza un pointer la ***int***

- i n t * (*f p [5]) (c h a r *); // vector de 5 pointeri la functii ce primesc ***char**** si returneaza un pointer la ***int***

PREPROCESARE

- Etapa ce precede compilarea
- Bazata pe simboluri definite prin #
- NU reprezintă instrucţiuni executabile
- Determina compilarea condiţionata a unor instrucţiuni
- Substituire simbolica
- Tipul enumerativ
- Macrodefinitii

PREPROCESARE

Substituire simbolica:

- bazata pe directiva *#define*

```
#define NMAX 1000
#define then
#define BEGIN {
#define END }

void main()
BEGIN
int vb = 10;
int vector[NMAX];
if(vb < NMAX) then printf("mai mic");
else printf("mai mare");
END
```

Substituire simbolica:

- valabilitate simbol:
  - sfarsit sursa;
  - redefinire simbol;
  - invalidare simbol:

```
#define NMAX 1000
….
#define NMAX 10
…
#undef NMAX
```

Tipul enumerativ:

enum denumire {lista simboluri} lista variabile

- valorile sunt in secventa
- se poate preciza explicit valoarea fiecarui simbol

enum rechizite {carte , caiet , creion = 4, pix = 6, creta}

PREPROCESARE

Macrodefinitii:

#define nume_macro(lista simboluri) expresie

Exemplu:

    #define PATRAT(X) X*X
    #define ABS(X) (X) < 0 ? – (X) : (X)

Sursa C/C++

```
...
int x=PATRAT(3);
int y=PATRAT(3+2);
...
```

PREPROCESARE

Macrodefinitii generatoare de functii:

```
#define SUMA_GEN(TIP) TIP suma(TIP vb2, TIP vb2) \
        { return vb1 + vb2; }
```

Compilare conditionata:

```
#if expresie_1
        secventa_1
#elif expresie_2
        secventa_2
…
#else
        secventa_n
#endif
```

PREPROCESARE

Compilare conditionata:

```
#ifdef nume_macro
        …
#else
        …
#endif
```

sau

```
#ifndef nume_macro
        …
#endif
```

PREPROCESARE

Operatorii # si ##:

- sunt utilizati impreuna cu #define
- operatorul # (*de insiruire*) transforma argumentul intr-un sir cu “”;

```
#define macro1(s) # s
```

- operatorul ## (*de inserare*) concateneaza 2 elemente

```
#define macro2(s1, s2)  s1 ## s2
```