Programare Procedurala

Liviu Dinu
ldinu@fmi.unibuc.ro

Bibliografie obligatorie

- Brian Kernighan, Dennis Ritchie. Limbajul C, Ed. Teora, 2003 (trad. Ionut Utescu)

- Bibliografie suplimentara - alte cursuri de C

Introducere

- Elemente esentiale ale limbajului
- Prezentare formala, mai putin riguroasa
- Scop: scrierea cat mai rapida a unui program
- Neajuns: nu vom descrie complet niciun element al limbajului

Introducere(2)

- Invatare prin exemple.
- Ex. 1:

```
#include <stdio.h>
main()
{
printf("Salut, cum te simti azi?  \n");
}
```

Observatie:

- Orice program in C incepe sa se execute la intalnirea functiei *main*
- Instructiunile unei functii sunt inchise intre acolade { }
- Secventa \n reprezinta *rand nou;* alte secvente: \t (tab), \b (backspace), \\ (backslash), etc.

Fahrenheit-Celsius
$^{o}C=(5/9)(^{O}F-32)$:

```
 0   -17
 1    -6
 2    4
 3    15
 4    26
 ……..
 280  137
300    148
```

```
#include <stdio.h>
/*afiseaza tabelul alaturat
fahrenheit-celsius*/
main()
{
int  fahr, cel;
int prim, ultim, pas;
prim=0;
ultim=300;
pas=20;
fahr=prim;
        while(fahr<=ultim)
        {
        cel=5*(fahr-32)/9;
        printf("%d \t %d \n", fahr, cel);
        fahr=fahr+pas;
        }
}
```

printf(“%3d %6d \n”, fahr, cel);

```
#include <stdio.h>
/*afiseaza tabelul alaturat*/
main()
{
float  fahr, cel;
float prim, ultim, pas;
prim=0;
ultim=300;
pas=20;
fahr=prim;
      while(fahr<=ultim)
      {
      cel=(5.0/9.0)*(fahr-32.0);
      printf(“%3.0f   %6.1f \n”,
      fahr, cel);
      fahr=fahr+pas;
      }
}
```

For

- Acelasi lucru putea fi scris si cu for:

```
#include <stdio.h>
/*afiseaza tabelul alaturat*/
main()
{
int fahr
for (fahr=0;fahr<=300;fahr=fahr+20)
      printf("%3d   %6.1f \n", fahr, (5.0/9.0)*(fahr-32));
}
```

For and while

- Formal, instructiunea for are sintaxa:

for(*expr1;expr2;expr3*)

instructiune

- Echivalenta cu:

expr1;

while(*expr2*)

{*instructiune*

expr3;

}

for(*initializare;conditie;incrementare*) *instructiune*

- *initializare* si *incrementare* sunt atribuiri sau apeluri de functie (cel mai adesea), *conditie* expresie relationala
- Oricare din cele 3 expresii poate lipsi (nu si ;)
- **Q1: Ce se intampla daca lipseste *expr1*, *expr2* sau *expr3*?**

if (expresie)
instructiune
else instructiune

```
/*cautbin: cauta x in
    v[0]<=v[1]<=...<=v[n-
    1]*/
int cautbin(int x,int v[], int n)
{
int prim, ultim, mijl;
prim=0;
ultim=n-1;
while (prim<=ultim)
      {
      mijl=(prim+ultim)/2;
      if (x<v[mijl])
            ultim=mijl-1;
            else if (x>v[mijl])
                  prim=mijl+1;
            else /* s-a gasit*/
            return mijl;
            }
      return -1; /*nu s-a gasit*/
}
```

```
switch (expresie) {
case expr-const: instructiuni
case expr-const: instructiuni
default: instructiuni
}
```

```
#include <stdio.h>
main() /* numara cifre, spatii, altele*/
{
int c, i, nalb, nalte, ncifra[10];
nalb=nalte=0;
for (i=0;i<10;i++)
ncifra[i]=0;
while ((c=getchar())!=EOF)
        {
        switch (c ) {
        case '0': case'1': case'2': case'3':
        case'4': case'5': case'6': case'7':
        case'8': case'9':
        ncifra[c-'0']++;
        break;
        case ' ':
        case '\n':
        case '\t':
        nalb++;
        break;
        default: naltul++;
        break;
        }
    }
printf(“cifre =”);
for (i=0; i<10;i++)
printf(“%d”, ncifra[i]);
printf(“, spatiu alb = %d, altul = %d\n”,
nalb, nalte);
return 0;
}
```

```
do
   instructiune
while (expresie);
```

- Exemplu:

```
void itoa (int n, char s[])
{
int i, semn;
if ((semn=n)<0)
     n=-n;
i=0;
do{
s[i++]=n%10+’0’;
} while ((n/=10)>0);
if (semn<0)
     s[i++]=‘-’;
s[i]=‘\0’;
}
```

Break and *continue*

- Break: pentru a inchide un *case* intr-un *switch* sau pentru a iesi imediat dintr-o bucla
- *Continue*: forteaza trecerea la urmatoarea iteratie a buclei.

Exemplu:

```
#include <stdio.h>
void main(void)
{
char s[80], *sir;
int spatiu;
printf("introduceti un sir ");
gets(s);
sir=s;
for(spatiu=0; *sir; sir++){
      if (*sir!=' ') continue;
      spatiu++;
}
printf("%d spatii \n", spatiu);
}
```

Matrice si siruri

- Toate matricele au 0 ca indice pentru primul element
- Exemplu:

```
void(main)
{
int x[100];
int t;
for(t=0;t<100;++t) x[t]=t;
}
```

- C nu controleaza limitele unei matrice
- Programatorul controleaza limitele acolo unde exista!
- Matricele unidimensionale sunt liste de informatii de acelasi tip stocate in zone contigue

Pointer la matrice

- Un pointer la primul element al matricei:

```
int*p
int proba[9];
p=proba;
```

- Acest program atribuie

lui p adresa primului element din *proba*

- In C nu se poate transmite o matrice intreaga ca argument al unei functii.

Urmatorul cod introduce adresa lui i in func1():

```
void main(void)
{
int i[10];
func1(i);
…
}
```

Func1() poate fi declarata astfel:

- void func1(int *x){ } //pointer
- void func1(int x[9]){ } //matrice cu dimensiune
- void func1(int x[]) { }//matrice fara dimensiune

Siruri

- In C un sir este definit ca o matrice de caractere care se termina cu un caracter null.
- Un null este specificat ca ‘\0’ si are valoarea 0.
- Matricele de tip caracter trebuie declarate cu un caracter mai mult decat cel mai lung sir pe care il vor contine

Functii de manevrare a sirurilor

- strcpy(s1,s2) copiaza s1 in s2
- Strcat(s1,s2) concateneaza s2 la sfarsitul lui s1
- Strlen(s1) returneaza lungimea lui s1
- Strcmp(s1,s2) 0 daca s1 si s2 sunt identice, negativ daca s1<s2, pozitiv altfel
- Strchr(s1,c) pointer la prima aparitie a lui c in s1
- strstr(s1,s2) pointer la prima aparitie a lui s2 in s1

Exemplu

```
#include <stdio.h>
#include <string.h>
void main(void)
{
char s1[0], s2[80];
gets(s1);
gets(s2);
printf("lungimi: %d  %d",
    strlen(s1), strlen(s2));
if(!strcmp(s1,s2)) printf("siruri
    egale\n");
strcat(s1,s2);
printf("%s", s1);
strcpy(s1,"acesta este un test. \n");
printf("%s\n",s1);
if(strchr("hello",'e')) printf("e este in
    hello\n");
if(strstr("la revedere", "la")) printf("am
    gasit la");

}
```

Matrice multidimensionale

- Declarare exemplu:

  int d[10][20];

- Accesare, exemplu

  d[1][2];

- Cand o matrice este utilizata ca argument al unei functii, este transmis doar un pointer catre primul element al matricei.
- Trebuie insa definit neaparat numarul de coloane ale matricei!
- In general, trebuie definite toate dimensiunile mai putin cea din extremitatea stanga
- Exemplu:

  void func1(int x[][10]){ }

Matrice de siruri

- Similar unei matrice obisnuite:

char matrice_siruri[10][80] ;

- Pentru a acces un sir individual:

```
gets(matrice_siruri[2]);
```

apeleaza functia gets prntru al treilea sir din matrice.

Initializarea matricelor

- C permite initializarea matricelor in aclasi timp cu declararea lor.
- Exemplu:

  int i[5]={1, 2, 3, 4, 5};
- Matricele de caractere care contin siruri permit o initializare prescurtata:
- Exemplu:

  char str[15]=“Buna dimineata”;

  Este similar cu a scrie:

  char sir[15]={B’, ‘u’, ‘n’, ‘a’, ‘ ‘, ‘d’, ‘i’,… };

Initializarea matricelor multidimensionala

- Similar cu cele unidimensionale.
- Exemplu:
- int patrat[5][2]=

  {1,1,

  2,4,

  3,9,

  4,16,

  5,25

  };

- Initializarea matricelor fara marime:
- Exemplu:

  char e1[]="eroare de citire\n";

  char e2[]="eroare de scriere\n";

Rezultat:

printf("%s are marimea %d", e2, sizeof e2);

Va afisa:

eroare de scriere are marimea 18

I/O la consola

- Cele mai simple functii I/O pentru consola sunt getchar(), care citeste un caracter de la tastatura, si putchar, care scrie un caracter e ecran.
- getchar() asteapta sa fie apasata o tasta si returneaza valoarea sa. Tasta apasata are ecou imediat pe ecran.
- Putchar() scrie un caracter pe ecran in dreptul cursorului.
- Fisierul antet pentru ele este <stdio.h>

Exemplu

```
#include<stdio.h>
#include<ctype.h>
void main(void)
{
char ch;
printf(“introduceti un text.\n”);
do
{
ch=getchar();
if(islower(ch)) ch=toupper(ch);
else ch=tolower(ch);
putchar(ch);
}while (ch!=‘.’);
}
```

- Prototipurile functiilor:

- int getchar(void);
- void putchar(int c);

- Alternative la getchar():
  - getch(): asteapta apasarea unei taste dupa care returneaza imediat; nu are ecou pe ecran.
  - getche(): identica cu getch() dar caracterul apare pe ecran.

  Sunt definite in conio.h

Citirea si scrierea sirurilor

- Functia gets() citeste un sir de caractere introduse de la tastatura si le plaseaza la adresa indicata de argumentul sau.
- Caracterele se citesc pana cand apare un caracter linie noua; acesta nu face parte din sir.
- gets() nu poate fi folosit pentru a returna caracterul linie noua.

- Functia puts() scrie pe ecran argumentul sau sir urmat de o linie noua.
- Valoarea returnata de puts() este rareori urmarita; se presupune ca atunci cand scrii la consola nu ai erori.

printf si scanf(). Specificatori de format

- %c caracter;
- %d nr. intregi in baza 10 cu semn
- %i idem
- %e notatie stiintifica
- %E notatie stiintifica
- %f numar zecimal in virgula mobila
- %o nr. in octal fara semn
- %s sir de caractere
- %u nr. intregi fara semn
- %x nr. in hexa fara semn
- %X nr in hexa fara semn (litere mari)
- %p afiseaza un pointer
- %% afiseaza %.

Specificatori pt. marimea minima a campului

- Un intreg plasat intre % si codul pentru format.
- Umple iesirea cu spatii pentru a se asigura ca ajunge la o anumita lungime minima
- Se poate pune un 0 inaintea specificatorulu pentru marimea campului pentru a se completa cu 0.

- Specificatorii de precizie
  - Specificatorii de precizie urmeaza specificatorului de camp minim.
  - Consta dintr-un punct urmat de un intreg.

- Alinierea iesirilor
  - Implicit toate sunt aliniate la dreapta.
  - Putem forta ca ele sa fie la stanga, prin inserarea unui – imediat dupa %: %-10.2 f va alinia la stanga un numar in virgula mobilancu doua zecimale intr-un camp de 10 caractere.

I/O cu fisiere

- Sistemul de fisiere din C este proiectat sa lucreze cu o mare varietate de echipamente: terminale, drivere de disc si drivere de unitate de banda.
- Chiar daca echipamentele difera, sistemul de fisiere din C le transforma pe fiecare intr-un instrument logic numit stream.
- Doua tipuri de stream:
  - Text
  - binar

Streamuri

- Stream text
  - Un stream text este o secventa de caractere. Standardul ANSI C permite (dar nu impune) ca un stream de tip text sa fie organizat in linii ce se termina cu un caracter linie noua.
- Stream binar
  - Un stream binar este o secventa de octeti intr-o corespondenta biunivoca cu cei de la echipamentul extern- nu apar transformari de caractere

Fisiere

- In C un fisier poate sa fie orice, de la un fisier pe disc pana la un terminal sau o imprimanta.
- Un stream este asociat cu un fisier printr-o operatie specifica de deschidere.
- O data deschis un fisier se poate realiza un schimb de informatii intre el si program.
- Printr-o operatie de inchidere realizam disocierea fisierului de stream.

Functii ale sistemului de fisiere ANSI C

- fopen() deschide fisier
- Fclose() inchide fisier
- Putc() scrie caracter in fisier
- Fputc() idem
- Getc() citeste caracter din fisier
- Fgetc() idem
- Fseek() cauta un octet in fisier
- Fprintf() actioneaza in fisiere la fel ca printf() la consola
- Fscanf() similar
- Feof() returneaza adevarat daca s-a ajuns la finalul fisierului
- Ferror() adevarat daca a aparut vreo eroare
- Rewind() readuce indicatorul de pozitie al fisieruli la inceput
- Remove() sterge un fisier
- Fflush() goleste un fisier

Pointerul fisierului

- Pointerul fisierului este legatura dintre fisier si sistemul I/O
- Pentru a citi/scrie fisiere programul trebuie sa foloseasca pointeri pentru ele.

```
FILE *p;
```

Deschiderea unui fisier

- Funcita fopen() deschide un stream pentru a fi folosit si il asociaza unui fisier.
- Prototip:

FILE *fopen(const char *numefisier, const char*mod);

Exemplu:

```
FILE *fp;
Fp=fopen(“test”,”w”);
Altfel:
FILE *p;
if((fp=fopen(“test,”w”))==NULL {
printf(“Nu pot deschide”);
exit(1);
}
```

Observatii

- Daca se foloseste fopen() pentru a se deschide un fisier pt. scriere, orice fisier care exista deja cu acel nume va fi sters si va fi inceput un nou fisier. Daca nu exista un fisier, va fi creat unul. Daca se doreste sa adaugam la sfarsitul unui fisier, folosim modul “a”.
- Pot fi deschise fisiere pt. operatii citire/scriere; in acest caz fisierul nu va fi sters daca exista.daca nu exista va fi creat.
- Pot fi deschise FOPEN_MAX fisiere la un moment dat (>=8).

Moduri permise

- r deschide fisier pt. citire
- w creeaza fisier pt. scris
- a adauga intr-un fisier text
- rb deschide un fisier binar pt. citire
- wb creeaza un fisier binar pt. scriere
- ab adauga intr-un fisier binar
- r+ deschide un fisier text pentru citit/scris
- w+ creeaza un fisier tip text pentru citit/scris
- a+ adauga in sau creeaza un fisier text pentru citit/scris
- r+b deschide un text in binar pentru citit/scris
- w+b creeaza un fisier de tip binar pentru citit/scris
- a+b adauga sau creeaza un fisier de tip binar pentru citit/scris.

Operatii

- Inchiderea unui fisier: fclose()
  - Prototip:
    int fclose(FILE *fp);
- Scrierea unui caracter:
  - Prototip:
    Int putc(int ch, FILE *fp);
- Citirea unui caracter:
  - Prototip:
    int getc(FILE *fp);
    Getc returneaza EOF cand se ajunge la sfarsit de fisier

Dar returneaza EOF si daca a intalnit o eroare. Pentru a vedea exact ce s-a intamplat trebuie folosit ferror()

Exemplu

```
#include<stdio.h>
#include<stdlib.h>
void main(int argc, char *argv[])
{
      FILE *fp;
      char ch;
      if (argc!=2){
        printf(“nu ati introdus numele fisierului\n”);
              exit(1);
      }
      if(fp=fopen(argv[1], “w”))==NULL {
         printf(“nu pot deschide fisierul\n”);
              exit(1);
      }
      do{
      ch=getchar();
      putc(ch,fp);
      } while (ch!=‘.’);
      fclose(fp);
}
```

```
#include<stdio.h>
#include<stdlib.h>
void main(int argc, char *argv[])
{
      FILE *fp;
      char ch;
      if (argc!=2){
        printf(“nu ati introdus numele fisierului\n”);
              exit(1);
      }
      if(fp=fopen(argv[1], “r”))==NULL {
         printf(“nu pot deschide fisierul\n”);
              exit(1);
      }
      ch=getc(fp);
      while(ch!=EOF){
      putchar(ch);
      ch=getc(fp);
}
      fclose(fp);
}
```

- fputs si fgets –similare cu putc() si getc();
- rewind()
  - Readuce indicatorul de pozitie al fisierului la inceput, indicatorul fiind specificat ca argument.
  - Prototip:

    void rewind(FILE *fp);

Exemplu

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main(void)
{
      char sir[80]
      FILE *fp;
        if((fp=fopen(“test,”w”))==NULL {
        printf(“Nu pot deschide”);
        exit(1);
        }
        do{
        printf(“introduceti un sir\n”);
        gets(sir);
        strcat(sir, “\n”);
        fputs(sir,fp);
        } while(*sir!=‘\n’);
rewind(fp);
while(!feof(fp)){
fgets(sir,79,fp);
printf(sir);
}
}
```

Pointeri

- Un pointer este o variabila care contine o adresa din memorie.
- Aceasta adresa este de obicei localizarea in memorie a unui obiect –de obicei o alta variabila
- Daca o variabila contine adresa altei variabile se prima se spune ca este pointer (indica pe) la cea de a doua
- Pointerii sunt considerati impreuna cu instructiunea *goto* ca fiind o **modalitate excelenta de a crea programe imposibil de inteles.**

Variabile de tip pointer

- O declarare de pointer consta dintr-un *tip de baza*, un * si numele variabilei:

  *tip * nume;*

- Tipul de baza al pointerului defineste tipul de variabila catre care indica acesta.
- Toata aritmtica pointerilor este creata relativ la tipul de baza

Operatori pentru pointeri

- Exista doi operatori specifici pentru pointeri: & si *.
- & este un operator unar care returneaza adresa din memorie a operandului sau.
  - Exemplu:

    m=&numara;

  introduce in m adresa variabilei numara.
- Aceasta adresa este locatia interna din calculator a variabilei
- Instructiunea precedenta inseamna *m* primeste adresa lui *numara*
- Daca numara are valoarea 100 si se afla la adresa 2000, dupa atribuirea precedenta m va avea valoarea 2000.

*

- * este complementarul lui &
- Este un operator unar care returneaza valoarea inregistrata la adresa care ii urmeaza.
- Daca *m* contine adresa din memorie a variabilei *numara*,

  q=*m;

  plaseaza valoarea din *numara* in *q*
- q va avea valoarea 100
- Intructiunea precedenta poate fi citita : *q primeste valoarea de la adresa m*

Pointeri

- Variabilele de tip pointer trebuie sa indice intotdeauna tipul corect de date.
- Cand declaram un pointer ca fiind de tipul int, compilatorul intelege ca adresa pe care o contine memoreaza o variabila de tip int.
- Urmatorul exemplu este corect sintactic, dar nu incorect semantic:

```
void main(void)
{
float x,y;
int *p;
p=&x;  //determina p sa indice catre un float
y=*p
}
```

- Nu va atribui valoarea din x lui y. deoarece p este declarat ca un pointer de tip int, vor fi transferati in ydoar 2 octeti din memorie, nu toti 8 care formeaza un float.

Expresii cu pointeri

- Atribuirea pentru pointeri:

```
#include<stdio.h>
void main(void)
{
int x;
int *p1, *p2;
p1=&x;
p2=p1;
printf("%p",p2); /* afiseaza adresa
    lui x, nu valoarea sa*/
}
```

- Aritmetica pointerilor
- Exista doua operatii:
  - Adunarea si scaderea

Sa luam in pointer de tip intreg cu valoarea 2000. Pp ca intregii au 2 octeti

Dupa expresia:

p1++;

p1 va contine 2002, nu 2001.

Motivul este ca de cate ori va fi incrementat p1 va indica spre urmatorul intreg.

Analog pentr scadere:

p1--;

Va da lui p1 valoarea 1998.

- p1=p1+12;

va face ca p1 sa indice al 12-lea element de acelasi tip cu p1.

- Doi pointeri pot fi scazuti pentru cate elemente de acelasi tip ii separa

Compararea pointerilor

- Doi pointeri pot fi comparati intr-o expresie relationala:

if (p<q) printf(“p indica o memorie mai mica decat q”);

Initializarea pointerilor

- Dupa ce este declarat un pointer dar inainte de a i se atribui o valoare el poate sa contina o valoare necunoscuta.

Functii de alocare dinamica

- Alocarea dinamica este caracteristica prin care un progrma poate obtine memorie in timpul rularii.
- Nucleul sistemului de alocare din C consta din functiile malloc() si free().
- Se gasesc in fisierul antet stdlib.h
- Prototip:

```
void *malloc(size_t numar_octeti);
```

- Dupa un apel reusit, malloc() returneaza un pointer spre primul octet al regiunii de memorie alocate in memoria libera.

Alocare dinamica

- Exemplu:

char *p;

p=malloc(1000); //preia 1000 de octeti

Dupa alocare, p indica spre primul din cei 1000 de octeti.

- Spatiu pentru 50 de intregi:

p=malloc(50*sizeof(int));

Pentru a trata erorile folosim urmatorul fragment de cod:

```
if(!(p=malloc(1000))){
printf(“depasire”);
exit(1);
}
```

Probleme ale pointerilor

- Pointerul neinitializat:

```
void main(void)
{
int x, *p;
x=10;
*p=10;
}
```

Rezultat: se va pune valoarea 10 la o adresa oarecare, nceunoscuta.

Probleme ale pointerilor(2)

```
#include <stdio.h>
void main(void)
{
int x, *p;
x=10;
p=x;
printf(“%d”, *p);
}
```

Rezultatul nu va consta in afisarea valorii 10, ci o valoare necunoscuta, cea care se afla la adresa 10.

Atribuirea corecta:

```
P=&x;
```

Structuri

- O structura este un grup de variabile unite sub acelasi nume, ce pune la dispozitie un mod convenabil de pastrare a informatiilor legate intr ele.
- Variabilele care fac parte din structura sunt denumite membri ai structurii
- In general toti membrii structurii au legatura logica
- Exemplu:

```
struct adrese
    {
    Char nume[30];
    Char strada[40];
    Char oras[20];
    Char judet[3];
    unsigned long int cod;
    };
```

Structuri(2)

- Pentru a declara o structura, folosim sintaxa:

struct adrese adr_inform;

- Putem renunta la cuvantul cheie struct.
- Accesul la membrii structurii se face prin folosirea operatorului .

adr_inform.cod=12345;

- Putem folosi gets:

```
gets(adr_inform.nume);
```

Atribuire in structuri

- Informatia constinuta intr-o structura poate fi atribuita unei alte structuri printr-o singua operatiune de atribuire:

```
#include <stdio.h>
void main(void)
{
struct{
int a;
int b;} x,y;
x.a=10;
y=x //atribuie o structura alteia
printf(“%d”, y.a);
}
```

Structuri

- O declaratie stuct defineste un tip.
- O declaratie de structura care nu este urmata de o lista de variabile nu aloca spatiu; ea descrie practic un sablon
- O structura poate fi initializata plasand dupa definitia ei o lista de valori de initializare pentu membri:
- Exemplu:

```
struct punct {
int x;
int y;
};
…
struct punct max={320,200};
```

Structuri imbricate

- O structura poate fi initializata printr-o atribuire sau prin apelarea unei functii care intoarce o structura de tipul corespunzator
- Structurile pot fi imbricate.
- Exemplu dreptunghi:

```
struct drept{
struct punct p1;
struct punct p2;
}
…
struct drept ecran;
ecran.pt1.x
```

se refera la coordonata x a punctului p1.

Operatii cu Structuri

- Singurele operatii legale cu o structura sunt copierea sau atribuirea sa gandita ca o unitate, citirea acesteia cu operatorul & si accesarea membrilor sai.
- Copierea si atribuirea include si transmiterea de argumente functiilor si returnarea de valori din functii
- Structurile nu pot fi comparate

Structuri si functii

- 3 abordari:
  - Transmiterea componentelor separat
  - Transmiterea unei intregi structuri
  - Transmiterea unui pointer catre aceasta

- Fiecare abordare are propriile avantaje si dezavanatje

- Functia creeazapunct primeste 2 intregi si va returna o structura de tip punct:

```
struct point creeazapunct(int x, int y)
{
struct punct temp;
temp.x=x;
temp.y=y;
return temp;
}
```

Obs.: nu exista conflict intre numele argumentului si membrul cu acelasi nume

Exemplu

- Structura creeazapunct poate fi folosita pentru a initializa dinamic orice structura sau pentru a furniza unei functii argumente de tip structura:

Exemplu:

```
struct drept ecran;
struct punct mijloc;
struct punct creeazapunct(int,int);
…
ecran.pt1=creeazapunct(0,0);
ecran.pt2=creeazapunct(XMAX,YMAX);
mijloc=creeazapunct((ecran.pt1.x+ecran.pt2.x)/2),
    (ecran.pt1.y+ecran.pt2.y)/2);
```

Operatii

- Exemplu:

```
struct adunapuncte
    (struct punct p1, struct
    punct p2)
{
p1.x+=p2.x;
p1.y+=p2.y
return p1;
}
```

- Argumentele si valoarea returnata sunt structuri
- Se evidentiaza faptul ca parametrii de tip structura se transmit prin valoare

- Functia ptindrept testeaza daca un punct se afla in interiorul unui dreptunghi
- Exemplu:

```
int ptindrept (struct punct p, struct drept d)
{
return p.x>=d.pt1.x && p.x< d.pt2.x &&
p.y>=d.pt1.y && p.y <d.pt2.y;
}
```

Se presupune ca dreptunghiul are coordonatele lui pt1 mai mici decat cele ale lui pt2

- Functia urmatoare aduce un dreptunghi la “forma canonica”

```
#define min (a,b)  ((a)<(b) ? (a): (b))
#define max (a,b) ((a)>(b) ? (a): (b))

struct drept canondrept(struct drept d)
{
struct drept temp;
temp.pt1.x=min(d.pt1.x, d.pt2.x);
temp.pt1.y=min(d.pt1.y, d.pt2.y);
temp.pt2.x=max(d.pt1.x, d.pt2.x);
temp.pt2.y=max(d.pt1.y, d.pt2.y);
return temp;
}
```

- Daca o structura voluminoasa trebuie transmisa unei functii, este in general mai eficient sa transmitem un pointer decat sa copiem intreaga structura.
- Exemplu:

```
struct punct **pp;
```

Precizeaza ca pp este un pointer la o structura punct de tipul struct.

*pp este structura, iar (*pp).x si (*pp).y sunt membrii

Putem folosi pointerul pp astfel:

```
struct punct origine, *pp;
…
pp=&origine;
printf("originea este (%d,%d)\n", (*pp).x, (*pp).y);
```

- Operatorul . Este mai puternic decat operatorul *.
- *pp.x este echivalenta cu *(pp.x) si este ilegala deoarece x nu este pointer.
- Notatie alternativa:

p->membru

Face referire la membrul structurii p.

Notatii alternative:

d.pt1.x

dp->pt1.x

(d.pt1).x

(dp->pt1).x

- Operatorii . si -> se afla in varful ierarhiei precedentei.
- Exemplu:

```
struct{
int lung;
char *str;
}*p;
```

*p->str preia obiectul spre care indica str;

*p->str++ incrementeaza pointerul str dupa accesarea obiectului spre care indica acesta

- ++p->lung incrementeaza variabila lung, nu variabila p
- Expresia (++p)->lung incrementeaza variabila p inainte de a-l accesa pe lung
- Expresia (p++)->lung incrementeaza variabila p dupa accesare

Multumesc!