Programarea consultaţiilor organizate de
Facultatea de Matematică şi Informatică
pentru pregătirea concursului de admitere 2016

5 Martie 2016
9:00 -10:15

Pentru elevii de liceu

5 *Martie* 2016	$9 - 10^{15}$	*Subprograme. Mecanisme de transfer prin intermediul parametrilor*	*Lect. dr. Prejmerean Vasile*
	$10^{30} - 11^{45}$	*Algoritmi elementari*	*Conf. dr. Czibula Istvan*

http://www.cs.ubbcluj.ro/~per/Web_Page/Cursuri.htm

Universitatea Babeş-Bolyai, Cluj-Napoca

Facultatea de Matematică şi Informatică

Home	Cursuri	Proiecte	Cv	Calendar	Lucrari Lic. Dis. GrI	Contact

Cursuri 2015-2016

Semestrul I	. . .	. . .
Semestrul II	. . .	. . .
Lectii Admitere (Concursul *Mate-Info* ~ 16 Apr. 2016)	5 martie 2016, 9:00 - 10:15 sala 2/Et.I	*Subprograme. Mecanisme de transfer prin intermediul parametri*.

1. Subprograme definite de utilizator

1.1. Subprograme.

Mecanisme de transfer prin intermediul parametrilor.

- **Subprograme** (*Pascal*) :
  - proceduri ***Procedure***,
  - functii [***Function***].

- Parametri (tipuri/clasificari),
- Vizibilitate,
- Parametri de tip *Functie*, *Procedura*,
- Apelul recursiv (subpr. recursive),
- Definire simultană (*Forward*).

Subalgoritmi: rezolvă o anumită subproblemă

Apel:

[*Cheamă*] *Nume_Subalgoritm (Lista_parametri_actuali);*

Def.:

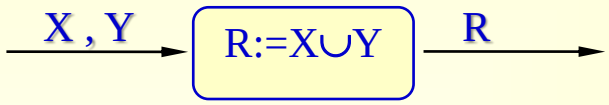
Subalgoritmul *Nume_Subalgoritm (Lista_parametri_formali)* ***Este :*** *{Antet}*

. . . *{ Corp subalgoritm }*

Sf_Subalgoritm. *{ sau* ***Sf_****Nume_Subalgoritm. }*

Parametri de Intrare → Expresii
Parametri de Iesire → Variabile

Exemplu: *pentru trei mulţimi date A, B şi C calculăm $A \cup B$, $A \cup C$ şi $B \cup C$*



Algoritmul Reuniuni Este :

Date A,B,C;

Cheamă ***Reuniune*** *(A,B,R1);*

Cheamă ***Reuniune*** *(A,C,R2);*

Cheamă ***Reuniune*** *(B,C,R3);*

Rezultate R1;

Rezultate R2;

Rezultate R3;

Sf_Algoritm.

Subalgoritmul Reuniune (X,Y, R) Este :

R:=X;

Pentru fiecare $y \in Y$ Execută

Dacă ***$y \notin X$*** *Atunci $R:=R \cup \{y\}$*

Sf_Dacă

Sf_Pentru

Sf_Reuniune.

Subalgoritmul Reuninune determină mulţimea $R = X \cup Y$ astfel :

$R := X \cup (Y \setminus X)$, adică depune în reuniune mai întâi elementele din mulţimea X, la care apoi mai adaugă acele elemente din Y care nu ***aparţin*** *lui X.*

Functii: ≈ *Subalgoritm + Val. functiei*

Apelul unei funcţii:
se face scriind într-o expresie numele funcţiei urmat de lista parametrilor actuali.

... Nume_Funcţie (Listă_parametri_actuali) ... { → *expr.* → *instr.* }

Def.:

Funcţia *Nume_Funcţie (Listă_parametri_formali)* ***Este*** : *{ Antetul funcţiei }*

. . .

Nume_Funcţie := Expresie; *{ Corpul funcţiei }*

. . .

Sf_Funcţie. *{ sau* ***Sf*** *_Nume_funcţie.}*

Parametri de Intrare → Expresii
Parametri de iesire → Variabile

Exemple: *Există & Apart*

```
Funcţia  Există  ( b, A, n, p )  Este :
        p:=1;
        Cât_Timp (p<=n) şi (b<>ap) Execută  p:=p+1  Sf_Cât_Timp;
        Există  :=  (p<=n)
Sf_Există.
```

```
Funcţia  Apart (b,A) Este :
        p:=1;                  {Card(A)=|A|}
        Cât_Timp (p<=Card(A)) şi (b<>A[p]) Execută  p:=p+1
        Sf_Cât_Timp;
        Apart  :=   (p<=Card(A))
Sf_Apart.
```

```
Funcţia  Card (A) Este :
          Card := a0
Sf_Card.
```

Dacă $y \notin X$ *Atunci* $R:=R\cup\{y\}$ *Sf_Dacă;*

Dacă Not ***Apart (y,X)*** *Atunci* $R:=R\cup\{y\}$ *Sf_Dacă;*

Exemplu: *determină maximul dintr-un şir X cu n componente .*

```
Funcţia  Max (X,k)  Este :
    Dacă k=1 Atunci       Max:=x1                          {Consistenţa}
         Altfel   Dacă  Max(X,k-1) < xk   Atunci Max:= xk
                                          Altfel   Max:= Max(X,k-1)
                  Sf_Dacă
    Sf_Dacă
Sf_Max.
```

```
Apelul:
    Max(X,n)
```

Exemplu: *decide dacă b aparţine primelor k elemente din şirul A .*

```
Funcţia Apart (b,A,k) Este :                    { ak  Propr. }
    Apart := (k>0)   şi  ( Apart(b,A,k-1)  Sau  (b=ak)  )
Sf_Apart.
```

```
Apelul:
    Apart(b,A,Card(A))
```

Subprograme Pascal : Procedure, Function :

<Def_subprogram> ::= <Def_funcţie> | <Def_procedură>

<Def_funcţie> ::= <Antet_funcţie> ; <Bloc>

<Def_procedură> ::= <Antet_procedură> ; <Bloc>

<Antet_funcţie> ::= Function <Nume> [(L_p_f)] : <Tip_f>

<Antet_procedură> ::= Procedure <Nume> [(L_p_f)]

Real, Integer, Byte, Boolean, Char, String, …

Apel:

P: *<Nume> [(Lista_parametri_actuali)];*

F: *… <Nume> [(Listă_parametri_actuali)] …* { → expr. → instr. }

Parametri :

<L_p_f.> ::= <spf> { ; <spf> }

<spf> ::= <sp_val> | <sp_var> | <p_functie> | <p_procedura>

<sp_val> ::= <lista_id> : <id_tip>

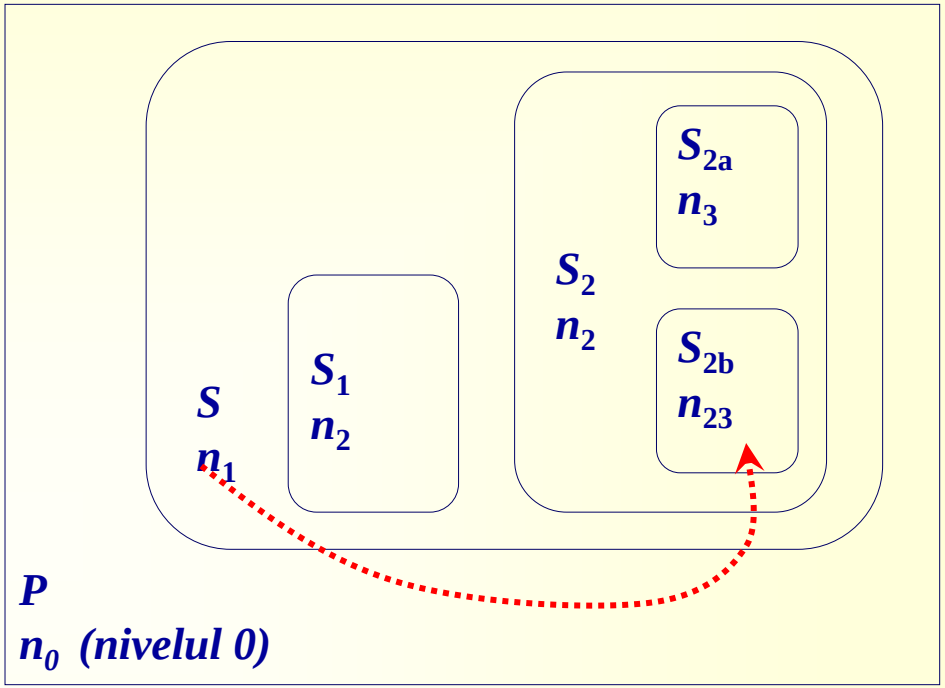
<sp_var> ::= Var <lista_id> : <id_tip>

Procedure Reuniune (A,B:Multime; Var C:Multime);

Vizibilitate :

Domeniu de vizibilitate:

- **Var.** ***locale***
- **Var.** ~~***globale***~~ **(!)**
- Proceduri,
- Functii,
- …




```
Function Apart (b:Integer; A:Multime) : Boolean;     {Apartine b multimii A?}
Var  i,n : Integer;              { Locale }
Begin
            i:=1;  n:=A[0];
    While (i<=n) And (b<>A[i]) Do i:=i+1;
    Apart :=  i<=n                                   {Apartine, daca i<=n }
End;
```

```
Program  AuB_AuC_BuC;
Type    Multime = Array[0..100] Of Integer;
Procedure Citeste  (Var A:Multime);  { Citeste o multime de la tastatura }
Var  c,p,n : Integer;        s : String;
Begin
    Write (' Dati elementele multimii : '); Readln (s);  n:=0;
    While s<>'' Do Begin
        n:=n+1;  Val(s,A[n],p);
        If  p>0  Then Val(Copy(s,1,p-1),A[n],c) Else p:=Length(s);   Delete (s,1,p)
    End;
    A[0]:=n;          { n=Card(A)→ A[0] }
End;
Procedure Tipareste (A:Multime);                    { Tipareste pe ecran o multime A }
Var  n,i : Integer;
Begin
    n:=A[0]; Write (' {');
    For i:=1 To n Do Write (A[i],',');
    Writeln (Chr(8),'}')                          { Rescrie peste ultima virgula, acolada }
End;
```

```
Function  Apart (b:Integer; A:Multime) : Boolean;          {Apartine b multimii A?}
Var  i,n : Integer;
Begin       i:=1; n:=A[0];
    While (i<=n) And (b<>A[i]) Do i:=i+1;
    Apart :=  i<=n                          {Apartine, daca i<=n }
End;
Procedure Reuniune (A,B:Multime; Var C:Multime);    { C := A ∪ B }
Var  i,p,q,r : Integer;
Begin                  C:=A;   p:=A[0]; q:=B[0];              r:=C[0];
    For i:=1 To q Do
       If  Not  Apart(B[i],A)   Then      Begin
                r:=r+1;  C[r]:=B[i]      End       C[0]:=r
End;
Var     A, B, C,   AuB, AuC, BuC : Multime;
Begin                                                          { Modulul principal }
    Citeste (A); Citeste (B); Citeste (C);
    Reuniune(A,B,AuB);  Tipareste (AuB);
    Reuniune(A,C,AuC);  Tipareste (AuC);
    Reuniune(B,C,BuC);  Tipareste (BuC);                              Readln
End.
```

Parametri de tip Function, Procedure :

Ex.: *Grafic (f,a,b); Grafic (g,c,d); …*
Integrala (f,a,b,SumeR); Integrala (g,c,s,Trapeze); …
Radac (f,a,b,eps,Coardei); Radac (g,c,d,0.01,Tangentei); …

```
Program pfp;          {$F+}.
Type   Functie    = Function   ( x : Real ) : Real ;
   Procedura  = Procedure  (   . . .   ) ;
Function  <nume_subpr>  ( ... f:Functie;  ...  p:Procedura; ... ) : ...
Procedure       …
Begin                  … f …          … p …                End;
```

Apelul:

```
Function  f1 (…):…                Procedure metoda1(…);…
Function  f2 (…):…                Procedure metoda2(…);…

. . .  <nume_subpr>  ( ... f1, ... ,metoda1, ... ) ...
. . .  <nume_subpr>  ( ... f2, ... ,metoda2, ... ) ...
```

```
Program  Parametri_Functie_Procedura;      {$F+}
Type     Sir       = Array  [1..30] Of Integer;
         Functie   = Function (x:Integer)   : Integer;
         FctBool   = Function (a,b:Integer) : Boolean;
         Procedura = Procedure(Var Z:Sir; n:Integer);
Function f (x:Integer) : Integer;  Begin  f:=Sqr(x+1)-x  End;
Function g (x:Integer) : Integer;  Begin  g:=Sqr(x-1)+x  End;
Function      Cresc(a,b:Integer):Boolean; Begin      Cresc:=a<b End;
Function DesCresc(a,b:Integer):Boolean; Begin  DesCresc:=a>b End;
Procedure Ordonez  (Var X:Sir; n:Integer;  Relatie : FctBool );
Var      Ordonat : Boolean; k, i, v : Integer;
Begin                    k:=1;
  Repeat  Ordonat:=True;
        For i:=1 To n-k Do
           If Not Relatie(X[i],X[i+1]) Then    Begin
                              v:=X[i]; X[i]:=X[i+1]; X[i+1]:=v;
              Ordonat:=False                      End;
        k:=k+1
  Until Ordonat;
End;
```

```
Procedure OrdCresc    (Var A:Sir; n:Integer);
Begin     Ordonez     (    A, n, Cresc     )  End;

Procedure OrdDesCresc (Var A:Sir; n:Integer);
Begin     Ordonez     (    A, n, DesCresc  )  End;

Procedure Gen_Ord_Tip (Var A:Sir; n:Integer;    r : Functie;
                                                Ordonare : Procedura );
Var  i:Integer;
Begin
    For i:=1 To n Do A[i]:=r(i);                          {Generare}
    Ordonare (A,n);                                       {Ordonare}
    For i:=1 To n-1 Do Write (A[i],','); Writeln(A[n])  { Tiparire }
End;

Var      X, Y : Sir;

Begin
   Gen_Ord_Tip (X,10,  f,  OrdCresc );                    { Prel X }
   Gen_Ord_Tip (Y,11,  g,  OrdDesCresc );    Readln  { Prel Y }
End.
```

Apelul recursiv :

Este permis apelul recursiv (un subprogram se autoapeleaza). Se ofera solutii simple plecand de la (avand) definitii simple.

Ex.: *Problema turnurior din Hanoi , n!=n*(n-1)! sau 1* (Consistenta!)

Recursivitate→ Iteratie !

Ex.: *"Fiind date două numere naturale de maxim 30 de cifre, sub forma a două şiruri de caractere, să se determine şirul cifrelor sumei"* :

Formulă recursivă pentru a aduna numărul *A* de *n* cifre cu numărul *B* de *m* cifre :

$$Ad(A,m,B,n,Tr) = \begin{cases} Ad(B,n,A,m,Tr) & \text{Dacă } m > n & (1) \\ \text{`` } & m=0,\ n=0,\ t=0 & (2) \\ \text{'1'} & m=0,\ n=0,\ t=1 & (3) \\ Ad(A,m,B,n-1,Tr(\text{'0'}+B[n]))\text{✦}Cifra(\text{`0'},B[n]) & m=0,\ n>0 & (4) \\ Ad(A,m-1,B,n-1,Tr(a[m],B[n]))\text{✦}Cifra(a[m],b[n]) & m>0,\ n>0 & (5) \end{cases}$$

```
Program   Adunare_Numere_siruri_cifre_baza_10;    { Adun_Str[n], n<=30 }
Const     n=31;
Type      Numar = String[n];
Function Adun (a, b : Numar) : Numar;        { Adun := A+B }
  Function Ad (m, n, t : Byte) : Numar;        { Aduna primele m resp. n  cifre }
  Var c:Char;
    Function Tr (a:Char):Byte;        { Tr:=a+b[n] Div 10 ;  a = 0 sau a[m] }
    Var s : Byte;
    Begin    Tr:=0;{  c:=a+b[n] Mod 10    }
      s:= Ord(a)+Ord(b[n])-$60+t;   { s=suma cifrelor 0..19 }
      If s>9 Then Begin Tr:=1; s:=s-10 End;
      c:= Chr(s Or $30);                  { c=caracterul sumei resturilor }
    End;
  Begin
  If n=0 Then  If  t=0  Then  Ad:=''                                  { Cazul (2) }
             Else       Ad:='1'                                       {       (3) }
          Else If m=0   Then Ad:=Ad(m  ,n-1,Tr( '0'))+c               {       (4) }
       Else Ad:=Ad(m-1,n-1,Tr(a[m]))+c                                 {       (5) }
  End;
```

```
Begin
  If Length(a)>Length(b)
       Then Adun := Adun (b,a)            { Cazul (1),    m>n }
       Else Adun := Ad (Length(a),Length(b),0)  { Initial Transportul = 0 }
End;   {Adun }

Function  Sp(a:Numar) : Numar;                  { Completeaza la stanga cu spatii }

Begin
       If Length(a)<n            Then  Sp:=Sp(' '+a)
                          Else  Sp:=a
End;

Var       a, b   : Numar;

Begin
  Write  (' Dati a : ');    Readln (a) ;
  Write  (' Dati b : ');    Readln (b) ;
  Writeln(Sp(a)+'  + ');                  {  Tipareste :       1234      +          }
  Writeln(Sp(b));                         {                     123                 }
  Write  (Sp(Adun (a,b)));                {                    1357                 }
  Readln
End.
```

Definire simultană (Forward) :

Function / Procedure S1 *(. . .) ;* **Forward**; *{ Antetul lui S1 }*

...

Function / Procedure S2 *(. . .) ;* *{ Subprogramul S2 apelează S1 }*

Begin ... S1... End;

Function / Procedure S1 *(. . .);* *{ Subprogramul S1 apelează S2 }*

Begin ... S2... End;...

" Ce functie (procedură) se va scrie prima **?** "

Ex. Pentru a compara două numere (a<b, a>b, a=b) reprezentate prin şirul (caracterelor) cifrelor, vom utiliza trei funcţii simultane (mutuale):

a<b → b>a,

a=b → *Not* a<b şi *Not* a>b,

a>b → NrCif(*a*) > NrCif(*b*) sau

NrCif(*a*) = NrCif(*b*) → Cat(*a*)>Cat(*b*) sau Cat(*a*)=Cat(*b*) şi Rest(*a*)>Rest(*b*) {/10}.

```
Program   Comparare_Numere_Str_cifre_baza_10;          { < , > , = ,  n<=10 }
Const     n=10;
Type      Numar = String[n];
Function  MaiMare(a,b : Numar) : Boolean;   Forward;
Function  MaiMic (a,b : Numar) : Boolean;
Begin     MaiMic:=MaiMare(b,a)   End;
Function  Egale (a,b : Numar) : Boolean;
Begin     Egale := Not MaiMic(a,b) And Not MaiMare(a,b) End;
Function  MaiMare (a,b:Numar) : Boolean;    Var m,n:Byte;
Begin                          m:=Length(a); n:=Length(b);
        MaiMare:=(m>n) Or                                          { |A| > |B| }
                       (m=n) And (m>0) And                         {    =      }
       (MaiMare(Copy(a,1,m-1),Copy(b,1,n-1)) Or                    { Cat(A) > Cat(B) }
      (Egale  (Copy(a,1,m-1),Copy(b,1,n-1)) And (a[m]>b[n])))      { Am > Bn }
End;
Var       a,b   : String;
Begin
  Write  (' Dati a : ');    Readln (a) ;          Write  (' Dati b : ');    Readln (b) ;
  If MaiMic   (a,b) Then Write (' a < b ')   Else
  If MaiMare(a,b) Then Write (' a > b ')   Else
  If Egale    (a,b) Then Write (' a = b ')   Else                    Write (' a ? b ');  Readln
End.
```

1.3. Funcţii

O **funcţie** este formată dintr-un ***antet*** şi un ***bloc*** (***corp***). Ea poate fi apelată dacă a fost definită în întregime sau doar antetul său.

Antetul unei funcţii are următorul format:

Tip Nume (Listă_parametri_formali)

unde:

- ***Tip*** este tipul valorilor funcţiei (codomeniul);
- ***Nume*** este un identificator (literă urmată eventual de litere sau cifre);
- ***Listă_parametri_formali*** conţine *parametrii formali* separaţi prin ‘,’.

Exemplu:

```
int Min (int a, int b)
{       if  (a<b)  return a;  else  return b; }
```

Obs. ***Prototipul*** unei funcţii este ***antetul*** acesteia urmat de ‘;’ .

Corpul unei funcţii are următoarea structură:

```
{
        Declaraţii
        Instrucţiuni
}
```

Exemple:

```
int Cmmdc (int a, int b)                          //  Cmmdc(a,b)
{
    if (b= =0) return a;
          else   return   Cmmdc(b,a % b);     //  Cmmdc(b,a  Mod  b);
}
int cmmdc (int a, int b)                          //  cmmdc(a,b)
{   int    rest;
    do {rest=a%b;
          a=b;
          b=rest;   }
    while (rest!=0);                  //  rest ≠  0;    sau    while (rest) ;
    return a;
}
```

2.4. Apelul unei funcţii

Instrucţiunea de apel a unei funcţii este un caz particular al instrucţiunii expresie:

Nume_funcţie (Listă_parametri_actuali);

O funcţie poate fi apelată şi în cadrul unei expresii dintr-o instrucţiune:

... Nume_funcţie (Listă_parametri_ actuali) ... ;

… 2.4. Apelul unei funcţii

O funcţie poate fi utilizată doar dacă a fost definită, sau cel puţin a fost declarat ***prototipul*** (***antet*** **;**) ei într-una din următoarele moduri:

a) Tip_funcţie Nume_funcţie (Lista_parametrilor_formali);

b) Tip_funcţie Nume_funcţie (Lista_tipurilor_parametrilor_formali);

c) Tip_funcţie Nume_funcţie (void); // nu sunt parametri formali

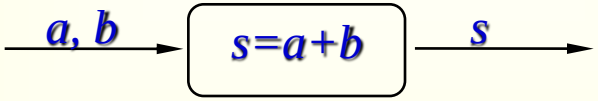
d) Tip_funcţie Nume_funcţie ( ); // nu se fac verificările de tip

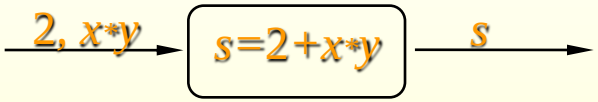
… 2.4. Apelul unei funcţii

Apelul implicit pentru *variabile simple* (*de bază*) este ***prin valoare***, iar pentru *tablouri* ***prin referinţă***. ***Apelul prin referinţă*** se obţine prin intermediul variabilelor de tip ***pointer*** sau a operatorului de adresă **&**.

Parametri de intrare → *Expresii*

Parametri de iesire → *Variabile*






```
void suma (int a, int b, int& s)
{
      s=a+b;
}
```

```
{     …
      suma (2, x*y, s);
      …
}
```

Revenirea dintr-o funcţie se poate realiza fie prin instrucţiunea ***return***, fie automat după ultima instrucţiune a funcţiei (situaţie în care nu se returnează nicio valoare):

return [*expresie*] ;

fiind returnată valoarea ***expresiei*** (dacă există).


```
void citeste (int& a)
{
      cin >> a;
}
```

```
int citeste (void)
{     int a;
      cin >> a;  return a;
}
```

```
void tipareste (int a)
{
      cout << a;
}
```

Exemplul 1:

```
#include <graphics.h>   #include <math.h>
int    u1,v1, u2,v2;       float  a, b,  c, d ;
int  u (float x)   { return ((x-a)/(b-a)*(u2-u1)+u1); }
int  v (float y)   { return ((y-d)/(c-d)*(v2-v1)+v1); }
void InitGraf(void)         { int Gd = DETECT, Gm;  initgraph(&Gd, &Gm, "c:\\Bc\\Bgi"); }
void ViewPort(int x1,int y1,int x2,int y2)              {u1=x1; v1=y1;
                                                  u2=x2; v2=y2;   /*rectangle(u1,v1,u2,v2);*/   }
void Window(float x1,float y1,float x2,float y2)      { a=x1; d=y1; b=x2; c=y2; }
void Rectangle(float x1,float y1,float x2,float y2)    { rectangle(u(x1),v(y1),u(x2),v(y2)); }
void Bar(float x1,float y1,float x2,float y2)           { bar(u(x1),v(y1),u(x2),v(y2)); }
void Linie(float x1,float y1,float x2,float y2)         { line(u(x1),v(y1),u(x2),v(y2)); }
void Muta(float x,float y)                    { moveto(u(x),v(y)); }
void Trag(float x,float y)                    { lineto(u(x),v(y)); }
void Rot(float &x,float &y, float x0, float y0, float Alfa)  {      float xp;
      xp=(x-x0)*cos(Alfa)-(y-y0)*sin(Alfa)+x0;
      y =(x-x0)*sin(Alfa)+(y-y0)*cos(Alfa)+y0;
      x = xp;
}
```

Exemplul 2:

```
#include <iostream.h>;
#include <conio.h>;
int Sf (int& f, int k)
{ int p=0;
  while (!(f%k)) { f/=k; p++; }
  return p;
}
main ()  {                                                clrscr();
      int    n;    int f2=0; int Uc=1;
      cout << " n : ";  cin >> n;
      for (int i=2; i<=n; i++)     { int f=i;
              f2+=Sf(f,2); f2-=Sf(f,5); Uc=Uc*f%10; }
      cout << " Uc= " << Uc*((f2&=3,int(f2?f2*1.4:3))<<1)%10;
      getch();
}
```

Exemplul 3: // Calc. A∪B, A∩B \\

```
#include <iostream.h>;     #include <conio.h>;
int Card(int A[])  {    return A[0];  }
int Apart (int x, int A[])
{   for (int i=1; i<=Card(A); i++) if (x==A[i]) return 1; return 0; }
void n (int A[], int B[], int C[])
{ C[0]=0; for (int i=1; i<=Card(A); i++) if (Apart(A[i],B)) C[++C[0]]=A[i];  }
void u (int A[], int B[], int C[])
{ int  i;  for (i=0; i<=Card(B); i++) C[i]=B[i];
  for (i=1; i<=Card(A); i++) if (!Apart(A[i],B)) C[++C[0]]=A[i];  }
void Tip (char *Mult, int A[])
{  int i; cout << Mult << '{' ;
   for (i=1; i<=Card(A); i++) cout << A[i] << ",";   cout << "\b}" << endl;  }
void main (void)
{                                                          clrscr();
  int A[]={5, 1,3,5,7,9};      Tip ("  A  : ",A);
  int B []={5, 1,2,3,4,5};     Tip ("  B  : ",B);
  int AuB[10];  u (A,B,AuB);   Tip (" AuB = ",AuB);
  int AnB[10];  n (A,B,AnB);   Tip (" AnB = ",AnB);          getche();
}
```

2.4.1. Operatorul de adresă (&)

Acest operator (&) se poate utiliza şi pentru a defini un tip *referinţă* printr-o declaraţie de forma *tip* & (asemănător cu o construcţie de forma *tip* *, pentru *pointer*).

Cu ajutorul acestui operator putem:

- redenumi o variabilă,
- **realiza un apel prin referinţă,**
- să declarăm o variabilă de referinţă astfel:

```
tip &  parametru_formal      // par. ref. (adresă)
```

```
tip &  nume_var_ref ;        // var. de tip referinţă
```

Exemplul 4: *// Apel prin Referinţă*

```
#include <iostream.h>;
void suma (int x, int y,  int *  z)   { *z  = ++x * ++y; }        // x,y → z
void Suma (int x,i nt y,  int &z)   {   z  = ++x * ++y; }        // x,y → z
void main (void)
{
   int x,y, z;
   cout << " Dati x,y : ";   cin  >> x >> y;
   suma(x,y,&z);
                        cout << "(x+1)*(y+1)=" << z << endl;
   Suma(x,y, z);
                        cout << "(x+1)*(y+1)=" << z << endl;      // mai simplu!
}
```

Recomandare: *Top* ⇩ *Down* *Mixta*

Pas: http://www.cs.ubbcluj.ro/~per/Fp%20-_-/

Cpp: http://www.cs.ubbcluj.ro/~per/C++%20Oop/

Succes!

… 5 Martie 2016 ~ 9:00-10:15 -_-