Concepte fundamentale ale limbajelor de programare

Transmiterea parametrilor. Subprograme generice

Curs 06

conf. dr. ing. Ciprian-Bogdan Chirila

Universitatea Politehnica Timisoara
Departamentul de Calculatoare si Tehnologia Informatiei

March 27, 2023

Cuprins

Transmiterea parametrilor

- folosită la comunicarea între subunitățile de program
- transfer de informație
- activat prin apelul de subprogram
  - procedură
  - funcție
  - subprograme
  - subrutină
- folosite pentru:
  - date
  - tipuri
  - alte subprograme

Cuprins

Mecanismul de bază

- în declarația unui subprogram specificăm
  - o lista de parametri formali în C, C++, C Sharp, Java
  - se cheama argumente fictive în Fortran
- acești parametri formali înlocuiesc
  - informația actuală din momentul apelului
  - pentru textul unui subprogram
- corespondența dintre parametrii actuali și formali este realizată în ordinea lor listată
  - în definiția subprogramului
  - în lista de argumente a apelului

Ce putem transmite

- diferite mecanisme de apel pentru
  - transmisiile de date
  - transmisiile de subprograme
- subprograme generice
  - descrierea de subprograme generice și parametrizate
  - instantierea de subprograme cu tipuri
  - exemple pentru Ada și C++

Transmiterea datelor ca parametri

- transmiterea prin adresă sau prin referintă
- transmiterea prin copiere
- transmiterea prin nume

Cuprins

Transmiterea parametrilor prin referintă sau adresă

- adresa argumentelor este transmisă subprogramului apelant
- orice acces la parametrul formal înseamnă un acces la locatia de memorie a cărei adresă este transmisă
- este un acces direct la parametrul actual

Exemplu

```
var z:t;
-----
procedure p(x:t);
-----
begin
 -----
 x:=3;
end;
-----
z:=5;
p(z);
p(z+2); //-> eroare
-----
```

Transmiterea parametrilor prin referintă

- argumentul
  - trebuie să fie o variabilă
  - trebuie să aibă o adresă
- transmiterea unei expresii ca argument va genera o eroare de compilare în majoritatea limbajelor de programare
- de exemplu `p(z+2); //-> eroare`
- mecanismul permite transmiterea datelor în ambele direcții:
  - de la apelant la subprogram
    - prin mecanismul de apelare
  - de la subprogram la apelant
    - prin modificarea valorilor apelantului

Cuprins

Transmiterea parametrilor prin copiere

- parametrii formali se comportă ca niște variabile locale
- orice modificări
  - ramân vizibile doar local, în subprogram
  - sunt invizibile în exterior
- depinzând de
  - valoarea inițiala a parametrilor formali
  - daca își utilizează sau nu valoarea lor finala avem
    - transmitere de valoare
    - transmitere de rezultat
    - transmitere de valoare si rezultat

Transmiterea de valoare

- înainte de apel
- valoarea parametrilor actuali este copiată în parametri formali
- devin valorile lor initiale
- modificările aplicate parametrilor formali
  - rămân invizibile din exterior
  - sunt aplicate doar parametrilor actuali
- parametri actuali rămân nemodificați dupa apel

Exemplu de transmitere de valoare

```
var z:t;
-------
procedure p:(x:t);
 var a:t;
begin
 a:=x-1;
 -------
 x:=1;
end;
z:=5;
-------
p(z);
p(z-5);
-------
```

Transmiterea de valoare

- parametrul actual poate fi o expresie
- mecanismul permite transmisia într-o singură direcție
  - de la apelant la subprogram

Transmiterea de rezultat

- valoarea parametrului actual nu afectează parametrul formal
- parametrul actual rămâne neinițializat dupa initierea apelului
- la ieșirea din functie valoarea finala a parametrului formal este copiata în parametrul actual
- parametrul actual își schimbă valoarea dupa apel

Exemplu de transmitere de rezultat

```
var z:t;
--------
procedure p:(x:t);
 -----
begin
 -----
 x:=3;
end;
-----
z:=5;
p(z);
-----
```

Transmiterea de rezultat

- parametrul actual trebuie să fie o variabilă
- mecanismul de transfer permite transferul de date într-o singură direcție
  - de la subprogram la apelant

Transmiterea de valoare și rezultat

- se comportă ca ambele mecanisme
  - transmiterea de valoare
  - transmiterea de rezultat
- argumentul actual este copiat în parametrul formal ca valoare inițiala
- la ieșirea din funcție valoarea parametrului formal va fi copiată în argumentul actual
- argumentul actual trebuie să fie o variabilă

Transmiterea de valoare și rezultat

- din punct de vedere al transferului de date se comportă ca și transmiterea prin referință
  - permite transmisia de date în ambele sensuri
- diferența constă în
  - mecanismul de transmitere de adresă modifică direct argumentele actuale în timpul execuției subrutinei
  - mecanismul de transmitere de valoare și rezultat menține argumentele nemodificate în timpul execuției subrutinei

Exemplu de transmitere de valoare și rezultat

```
var z:integer;
-------
procedure p:(x,y:integer);
begin
 x:=2*x;
 y:=2*y;
end;
-------
z:=3;
p(z,z);
-------
```

Transmiterea de valoare și rezultat

- procedura p dublează cele două valori transmise
- comportamentul este corect și rezultatul este cel așteptat la ambele mecanisme
  - transmiterea de adresă
  - transmiterea de valoare și rezultat
- cu excepția cazului când aceeași variabilă este pusă pe ambele poziții
- rezultatul este
  - 12 în cazul transmisiei prin referință
  - 6 în cazul transmisiei de valoare și rezultat

Cuprins

Transmiterea parametrilor prin nume

- este similară transmisiei de adresă unde
  - locația referențiată este parametrul actual
- în transmiterea de nume
  - locația referită rezultă prin înlocuirea textuală a numelor parametrilor formali cu numele parametrilor actuali

Exemplu de transmitere de parametri prin nume

```
var x,y,i:integer;
 t:array[1..100] of integer;
------------------
procedure p(a,b:integer);
 var m:integer;
begin
 m:=a;
 a:=b;
 b:=m;
end;
```

Exemplu de transmitere de parametri prin nume

- în cazul unui apel `p(x,y);`
- secvența executată este următoarea

```
m:=x;
x:=y;
y:=m;
```

- efectul este cel așteptat
- în special pentru variabile scalare

Exemplu de transmitere de parametri prin nume

- nu este aceeasi situație pentru un tablou
- `i:=3; t[i]=50;`
- apelul `p(i,t[i]);` va executa secvența

```
m:=i;
i:=t[i];
t[i]:=m;
```

- `i=50`, `t[3]` rămâne 50, dar `t[50]` devine 3 !!!
- folosind transmiterea prin adresă efectul ar fi de interschimbare al valorilor `i=50` și `t[3]=3`

Transmiterea parametrilor prin nume

- În concluzie la mecanismul de transmitere prin nume
  - argumentul poate fi orice expresie
  - expresia este evaluată ori de cate ori parametrul formal este accesat în timpul execuției procedurii

Cuprins

Exemplu de transmitere a parametrilor prin context de nume

```
var x:integer;
-------------------
procedure p(a:integer);
var x:integer;
begin
x:=2;
write(a); --> aici se va afisa 1
write(x); --> aici se va afisa 2
end;
-------------------
x:=1;
p(x);
-------------------
```

Parametrul actual evaluat în contextul apelului

- în ce context este evaluat parametrul actual?
- parametrul actual este evaluat în contextul apelului
- `write(a)` va afișa 1 deoarece a este înlocuit cu `x` care este global
- `write(x)` va afișa 2 deoarece x este o variabilă locală atribuită cu valoarea 2

Parametrul actual evaluat în contextul subprogramului

- `write(a);` ar afișa 2 deoarece a înlocuieste pe `x` care este evaluat în subprogram referindu-se la variabila locală `x`
- acest mecanism de transmitere este cunoscut ca transmitere prin text

Cuprins

Transmiterea parametrilor în diferite limbaje de programare

- Fortran
  - transmitere prin adresă
- Lisp, C, Algol 68
  - transmitere prin valoare
  - adresa pointerului poate fi transmisă ca o valoare
- C
  - când sunt transmise tablourile este transmisă adresa primului element
  - astfel solutia evită copierea pe stivă în zona de parametri a intregului tablou

Transmiterea parametrilor în diferite limbaje de programare

- la decizia de implementare a programatorului
  - Pascal
    - transmiterea prin valoare
    - transmiterea prin adresă
  - Algol 60
    - transmiterea prin nume
    - transmiterea prin valoare
  - Simula 67
    - transmiterea prin nume
    - transmiterea prin valoare
    - transmiterea prin adresă

Exemple

- Pascal:
  - `procedure p(a:integer; var x,y:real);`
  - `a` transmise prin valoare
  - `x,y` transmise prin adresă
  - `a` transmis prin valoare
- Ada:
  - `procedure p(a,b:in integer; x:in out boolean; z:out integer; c:character);`
  - `a`, `b`, `c` de tip `in` transmis prin valoare

Transmiterea parametrilor în Ada

- nu impune o anumită tehnică de implementare
- Pentru declarații cu `in`
  - Transmisie de valoare
- Pentru declarații cu `out`
  - Transmisie de rezultat sau prin adresă
- Pentru declarații cu `in out`
  - Transmisie de valoare și rezultat sau prin adresă

Transmiterea parametrilor in C++

- prin valoare
- prin referinta utilizand symbolul &

Transmiterea parametrilor in C++

```
#include<iostream>
using namespace std;
void f(int a, int &b)
{
 a+=5;
 b+=5;
}
int main()
{
 int x=10;
 int y=10;
 cout<<x<<" "<<y<<endl; // 10 10
 f(x,y);
 cout<<x<<" "<<y<<endl; // 10 15
 return 0;
}
```

Transmiterea parametrilor în C Sharp

- Pentru declarații fara specificator
  - Transmisie de valoare
- Pentru declarații cu specificator `in`
  - Transmisie de valoare, in regim readonly
- Pentru declarații cu specificator `out`
  - Transmisie de rezultat, fara a putea fi accesata valoarea initiala si nedefinita
- Pentru declarații cu specificator `ref`
  - Transmisie prin referinta

Transmiterea in C Sharp fara specificator

```
using System;
public class Program
{
 public static void f(int x)
 {
  x+=10;
  Console.WriteLine(x); //20
 }

 public static void Main()
 {
  int a=10;
  Console.WriteLine(a); //10
  f(a);
  Console.WriteLine(a); //10
 }
}
```

Transmiterea in C Sharp cu specificator in

```
using System;
public class Program
{
 public static void f(in int x)
 {
  // x=x+10 --wrong
  // x is readonly variable
  Console.WriteLine(x);
 }

 public static void Main()
 {
  f(10);
 }
}
```

Transmiterea in C Sharp cu specificator out

```
using System;
public class Program
{
 public static void f(out int x)
 {
  // Console.WriteLine(x);
  // x is unassigned, we can not use its value
  x=20;
  Console.WriteLine(x); //20
 }

 public static void Main()
 {
  int a=10;
  Console.WriteLine(a); //10
  f(out a);
  Console.WriteLine(a); //20
 }
```

Transmiterea in C Sharp cu specificator ref

```
using System;
public class Program
{
 public static void f(ref int x)
 {
  Console.WriteLine(x);
  x+=20;
  Console.WriteLine(x); //30
 }

 public static void Main()
 {
  int a=10;
  Console.WriteLine(a); //10
  f(ref a);
  Console.WriteLine(a); //30
 }
}
```

Parametri impliciti în Python

```
def f(x=10,y=20,z=30):
 print(x," ",y," ",z)

def main():
 f() # 10 20 30
 a=15
 b=25
 c=35
 f(a) # 15 20 30
 f(a,b) # 15 25 30
 f(a,b,c) # 15 25 35

main()
```

Ordinea parametrilor în Python

```
def f(x,y,z):
 print(x," ",y," ",z)

def main():
 a=15
 b=25
 c=35
 f(z=a, x=c, y=b) # 35 25 15

main()
```

Transmisia parametrilor în Python pentru argumente imuabile/neschimbatoare

```
def f(x):
 print("x=",x," id=",id(x)) # x=5 id=10746600
 x=40
 print("x=",x," id=",id(x)) # x=40 id=10747656

def main():
 x=5
 print("x=",x," id=",id(x)) # x=5 id=10746600
 f(x)

main()
```

Transmisia parametrilor în Python pentru argumente imuabile/neschimbatoare

- daca transmitem argumente imuabile cum ar fi intregii, string-urile sau tuplurile transmisia se face prin referinta
- se vede clar adresa variabilei `x` inainte si dupa apel
- referinta obiectelor este transmisa iar aceste obiecte nu pot fi modificate pentru ca sunt imuable/neschimbatoare
- in exemplu se vede cum asignarea unei alte valori 40 a dus la alocarea unei noi adrese de memorie pentru variabila `x`
- variabila originala `x` a ramas intacta

Transmisia parametrilor în Python pentru valori mutabile/schimbatoare

```
def no_side_effects(cities):
    print(cities) # ["Timisoara", "Arad", "Oradea", "Resita"]
    cities = cities + ["Sibiu", "Brasov"] # a new list is created
    print(cities) # ["Timisoara", "Arad", "Oradea", "Resita", "Sibiu
locations = ["Timisoara", "Arad", "Oradea", "Resita"]
no_side_effects(locations)
print(locations) # ["Timisoara", "Arad", "Oradea", "Resita"]
```

Transmisia parametrilor în Python pentru valori mutabile/schimbatoare

```
def with_side_effects(cities):
    print(cities) # ["Timisoara", "Arad", "Oradea", "Resita"]
    cities += ["Sibiu", "Brasov"] # the += operator reuses the old l
    print(cities) # ["Timisoara", "Arad", "Oradea", "Resita", "Sibiu
locations = ["Timisoara", "Arad", "Oradea", "Resita"]
with_side_effects(locations)
print(locations) # ["Timisoara", "Arad", "Oradea", "Resita", "Sibiu"
```

Transmisia parametrilor în Python pentru valori mutabile/schimbatoare

```
def with_side_effects(cities):
    print(cities) # ["Timisoara", "Arad", "Oradea", "Resita"]
    cities += ["Sibiu", "Brasov"]
    print(cities) # ["Timisoara", "Arad", "Oradea", "Resita", "Sibiu
locations = ["Timisoara", "Arad", "Oradea", "Resita"]
with_side_effects(locations[:]) # on a copy of the list
print(locations) # ["Timisoara", "Arad", "Oradea", "Resita"]
```

Cuprins

Transmiterea de subprograme ca parametru

- posibilă în mai multe limbaje de programare
  - Fortran, Pascal, C, C++, Lisp
- programul va executa diferite calcule în funcție de subprogramul trimis
- în Turbo Pascal
  - parametri de tip subprogram
  - funcții, proceduri

Exemplu subprogram dat ca parametru în Pascal

```
type fnt=function(x:integer):real;

precedure tab(f:fnt;j,i:integer);
 var a:integer;
begin
 for a:=j to i do
  writeln(a,f(a));
end;

{$F+}
function f1(x:integer):real;
begin
 f1:=2*3.14*x;
end;
```

Exemplu subprogram dat ca parametru în Pascal

```
function fact(x:integer):real;
 var f:real; i:integer;
begin
 f:=1.0;
 for i:=1 to x do
  f:=f*i;
 fact:=f;
end;
{$F-}
----------------------------
tab(f1,-10,10);
tab(fact,0,10);
----------------------------
```

Exemplu subprogram dat ca parametru în Fortran

```
      SUBROUTINE TAB(F,I,J)
      REAL F
      INTEGER J,I,A
      DO 1 A=J,I
      WRITE(*,2) A,F(A)
2     FORMAT(5X,I4,F10.3)
1     CONTINUE
      RETURN
      END
      REAL FUNCTION F1(X)
      INTEGER X
      F1=2*3.14*X
      RETURN
      END
```

Exemplu subprogram dat ca parametru în Fortran

```
REAL FUNCTION FACT(X)
  INTEGER X,I
  REAL F
  F=1.
  DO 1 I=1,X
  F=F*I
1 CONTINUE
  FACT=F
  RETURN
  END
```

Exemplu subprogram dat ca parametru în Fortran

```
C MAIN PROGRAM
  EXTERNAL F1,FACT
  REAL F1,FACT
  CALL TAB(F1,-10,10)
  CALL TAB(FACT,0,10)
  STOP
  END
```

Exemplu de subprograme date ca parametru în C

```
void tab(double (*f)(int),int j,int i)
{
 for(;j<i;j++)
  printf("%d %f\n",j,(*f)(j));
}

double f1(int x)
{
 return 2*3.14*x;
}
```

Exemplu de subprograme date ca parametru în C

```
double fact(int x)
{
 double f=1; int i;
 for(i=1;i<=x;i++)
  f*=i;
 return f;
}
----
tab(f1,-10,10);
tab(fact,0,10);
----
```

Exemplu de subprograme date ca parametru în Lisp

```
(DEFUN tab1(f j i)
 (PRINT (LIST f (FUNCALL f j)))
 (COND ((= j i) NIL)
  (T (tab1 f(+ j 1) i))))

(DEFUN tab(f j i)
 (COND ((> j i) NIL)
  (T (tab1 f j i))))

(DEFUN f1(x)
 (* 2 3.14 x))
```

Exemplu de subprograme date ca parametru în Lisp

```
(DEFUN fact(x)
 (COND ((ZEROP x) 1.0)
  (T (*(FLOAT x)(fact (- x 1))))))

(tab 'f1 -10 10)
(tab 'fact 0 10)
```

Cuprins

Subprograme generice în Ada

Subprograme generice în Ada

```
generic
 type tip_el is private;
 type vec is array (integer range< >) of tip_el;
 zero:tip_el;
 with function "+"(x,y:tip_el)
return tip_el;

function apply(v:vec) return tip_el is
 rez:tip_el:=zero;
begin
 for i in v'first..v'last loop
  rez:=rez+v[i];
 end loop;
 return rez;
end apply;
```

Subprograme generice în Ada

```
type v_int is array(integer range< >) of integer;
type v_real is array(integer range < >) of real;
function sum is new apply(integer,v_int,0,"+");
function prod is new apply(real,v_real,1,"*");
```

Subprograme generice în Ada

```
function ad_inv(x,y:integer) return integer is
begin
 if y=0 then
  return 0;
 else
  return x+1/y;
 end if
end ad_inv;
----
function s_inv is new apply(integer, v_int, 0, ad_inv);
```

Subprograme generice în C++

```
template <class T> void sort(T *array, int size)
--------
void main()
{
 int arrayofint[10]={---};
 double arrayofdouble[20]={---};
 ------
 // type instantiation and function calls
 sort(arrayofint,10);
 sort(arrayofdouble,20);
}
```

Subprograme generice în C++

```
//template definition
template <class T> void sort(T *array,int size)
{
 register int i,j;
 T temp;
 for(i=1;i<size;i++)
 {
  for(j=size-1;j>=i;j--)
  {
   if(array[j-1]>array[j])
   {
    temp=array[j-1];
    array[j-1]=array[j];
    array[j]=temp;
   }
  }
 }
}
```

Bibliografie

1. Brian Kernighan, Dennis Ritchie, C Programming Language, second edition, Prentice Hall, 1978.
2. Carlo Ghezzi, Mehdi Jarayeri – Programming Languages, John Wiley, 1987.
3. Horia Ciocarlie – Universul limbajelor de programare, editia 2-a, editura Orizonturi Universitare, Timisoara, 2013.