Fundamental Concepts of Programming Languages

Parameter transmission. Generic subprograms
Lecture 06

conf. dr. ing. Ciprian-Bogdan Chirila

University Politehnica Timisoara
Department of Computing and Information Technology

Parameter transmission

- Used for communication between program subunits
- Information transfer
- Enabled by the subprogram call
  - Procedure
  - Function
  - Subroutine
- Used for
  - Data
  - Types
  - Other subprograms

The basic mechanism

- In declaring a subprogram we specify
  - a list of formal parameters in C
  - fictive arguments in Fortran
- These formal parameters replace
  - the actual information set at call time
  - for the subprogram text
- Correspondence between actual and formal parameters is done in the listed order
  - of the subprogram definition
  - of the call arguments

To discuss next

- Different call mechanisms for
  - data transmission
  - subprogram transmission
- Generic subprograms
  - generalized and parameterized subprogram description
  - subprogram instantiation with types
  - e.g. for Ada and C++

Transmitting data as parameters

- Transmitting by address or by reference
- Transmitting by copying
- Transmitting by name

Lecture outline

1. Parameter transmission by reference (address)
2. Parameter transmission by copying
3. Parameter transmission by name
4. Parameter transmissions in different PLs
5. Transmitting subprograms as parameters
6. Generic subprograms

Parameter transmission by reference (address)

- The arguments address is passed to the called subprogram
- Any access to the formal parameter means an access to the memory location whose address was transmitted
- It is a direct access to the actual parameter

Example

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

Parameter transmission by reference (address)

- The argument
  - must be a variable
  - must have an address
- Transmitting an expression as argument will issue a compiling error in most PLs
- e.g.: p(z+2); -> ERROR
- The mechanism allows data transmission in both ways:
  - must be a variable
    - By the call mechanism
  - From the subprogram to the caller
    - By modifying the callers values

Lecture outline

1. Parameter transmission by reference (address)
2. Parameter transmission by copying
3. Parameter transmission by name
4. Parameter transmissions in different PLs
5. Transmitting subprograms as parameters
6. Generic subprograms

Parameter transmission by copying

- The formal parameter acts as a local variable
- Any modifications
  - will remain visible only locally, in the subprogram
  - will be invisible to the outside
- Depending on
  - formal parameter initial value
  - using or not its final value, we may have
    - Value transmission
    - Result transmission
    - Value and result transmission

Value transmission

- Before the call
- The value of the actual parameter is copied into the formal parameter
- It becomes its initial value
- Modifications applied on the formal parameter
  - Remain invisible from the outside
  - Are applied only to the formal parameter
- The actual parameters remains untouched after the call

Value transmission example

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

Value transmission

- The actual parameter can be any expression
- The mechanism allows transmission only in one way
  - From the caller to the subprogram
- The actual parameters remains untouched after the call

Result transmission

- The value of the actual parameter does not affect the formal parameter
- The actual parameter remains uninitialized after the call initiation
- At return the final value of the formal parameter is copied into the actual parameter
- The actual parameter changes its value after the call

Result transmission example

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

Result transmission

- The actual parameter must be a variable
- The transfer mechanism allows data transfer in one way from the subprogram to the caller

Value and result transmission

- Behaves like both
  - Value transmission
  - Result transmission
- The actual argument is copied into the formal parameter as its initial value
- At return the formal parameter value will be copied into the actual argument
- The actual argument must be a variable

Value and result transmission

- From the data transfer point of view behaves like reference transmission
  - Allows data transmission in both ways
- The difference is
  - The address transmission modifies directly the actual argument during the subroutine execution
  - The value and result transmission keeps the argument value unmodified during subroutine execution

Value and result transmission example

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

Value and result transmission

- Procedure p doubles the two transmitted values
- The behavior is correct and the result is the expected one in both
  - Address transmission
  - Value and result transmission
- Except the case when the same variable is set on the two positions
- The result is
  - 12 in the case of address transmission
  - 6 in the case of value and return transmission

Lecture outline

1 Parameter transmission by reference (address)

2 Parameter transmission by copying

3 Parameter transmission by name

4 Parameter transmissions in different PLs

5 Transmitting subprograms as parameters

6 Generic subprograms

Parameter transmission by name

- Is similar to the address transmission where
  - The referred location is the actual parameter
- In name transmission
  - The referred location results from the textual replacement of the formal parameter name with the actual parameter name

Parameter transmission by name example

```
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;
```

Parameter transmission by name example

- In case of a call p(x,y);
- The executed sequence is:
  m:=x
  x:=y
  y:=m
- The effect is the expected one
- Especially for scalar variables

Parameter transmission by name example

- it is not the same situation for an array
- i:=3; t[i]=50;
- The call p(i,t[i]); will execute the sequence:
  m:=i;
  i:=t[i];
  t[i]:=m;
- i=50, t[3] stays 50, but t[50] becomes 3!!!
- Using transmission by address the effect would be the arguments value exchange i=50 and t[3]=3

Parameter transmission by name example

- In conclusion in name transmission
  - The argument can be any expression
  - The expression is evaluated as many times as the formal parameter is accessed during procedure execution

Parameter transmission by name context example

```
var x:integer;
--------------------
procedure p(a:integer);
        var x:integer;
begin
        x:=2;
        write(a); --> here will print 1
        write(x); --> here will print 2
end;
--------------------
x:=1;
p(x);
--------------------
```

Actual parameter evaluated in the call context

- In which context is evaluated the actual parameter?
- The actual parameter is evaluated in the call context
- write(a) will print 1 since a is replace with x which is global
- write(x) will print 2 since x is a local variable assigned with value 2

Actual parameter evaluated in the subprogram context

- write(a); would print 2 because a replaces x which is evaluated in the subprogram denoting the local x
- This transmission is known as transmission by text

Lecture outline

1. Parameter transmission by reference (address)
2. Parameter transmission by copying
3. Parameter transmission by name
4. Parameter transmissions in different PLs
5. Transmitting subprograms as parameters
6. Generic subprograms

Parameter transmissions in different PLs

- Fortran
  - Transmission by address
- Lisp, C, Algol 68
  - Transmission by value
  - The pointer address can be transmitted as a value
- C
  - When arrays are transmitted the address of the first element is transmitted
  - Thus the solution avoids copying on the stack parameter memory zone the whole array

Parameter transmissions in different PLs

- At programmers choice
  - Pascal
    - Transmission by value
    - Transmission by address
  - Algol 60
    - Transmission by name
    - Transmission by value
  - Simula 67
    - Transmission by name
    - Transmission by value
    - Transmission by address

Parameter transmissions in different PLs

Ada

- does not impose a certain implementation technique
- declared as **in**
  - Transmitted by value
- declared as **out**
  - Transmission by result or transmission by address
- declared as **in out**
  - Transmission of value and result or transmission by address

Examples

- Pascal:
  - procedure p(a:integer; var x,y:real);
  - x,y transmitted by address
  - a transmitted by value
- Ada:
  - procedure p(a,b:in integer; x:in out boolean; z:out integer; c:character);
  - a,b,c of type **in** transmitted by value

Lecture outline

1. Parameter transmission by reference (address)
2. Parameter transmission by copying
3. Parameter transmission by name
4. Parameter transmissions in different PLs
5. Transmitting subprograms as parameters
6. Generic subprograms

Transmitting subprograms as parameters

- Possible in several PLs
  - Fortran, Pascal, C, Lisp
- The program will perform different computations depending on the sent subprogram
- In Turbo Pascal
  - Subprogram type parameters
  - functions, procedures

Subprograms as parameters
Pascal example

```
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;
```

Subprograms as parameters Pascal example

```
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);
----------------------------
```

Subprograms as parameters Fortran example

```
   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
```

Subprograms as parameters Fortran example

```
      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
```

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

Subprograms as parameters C example

```
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;
}
```

Subprograms as parameters C example

```
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);
----
```

Subprograms as parameters Lisp example

```
(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))
```

Subprograms as parameters Lisp example

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

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

Lecture outline

1. Parameter transmission by reference (address)
2. Parameter transmission by copying
3. Parameter transmission by name
4. Parameter transmissions in different PLs
5. Transmitting subprograms as parameters
6. Generic subprograms

Generic subprograms in 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;
```

Generic subprograms in 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,"*");
```

Generic subprograms in 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);
```

Generic subprograms in 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);
}
```

Generic subprograms in 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;
                           }
                    }
        }
}
```

Bibliography

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.