SUBPROGRAM

Topics:

- Definitions of subprogram
- general subprogram characteristics
- parameters
- Functions and procedures
- Design issues for subprogram
- Parameter passing method
- Model for parameter passing
- Overload subprogram
- Type checking & referencing environment
- Subprogram passed method
- Generic subprogram

Definitions :

Subprogram :

A program separate from the main program that executes aseries
Of operations that occures multiple times during the machine cycle.

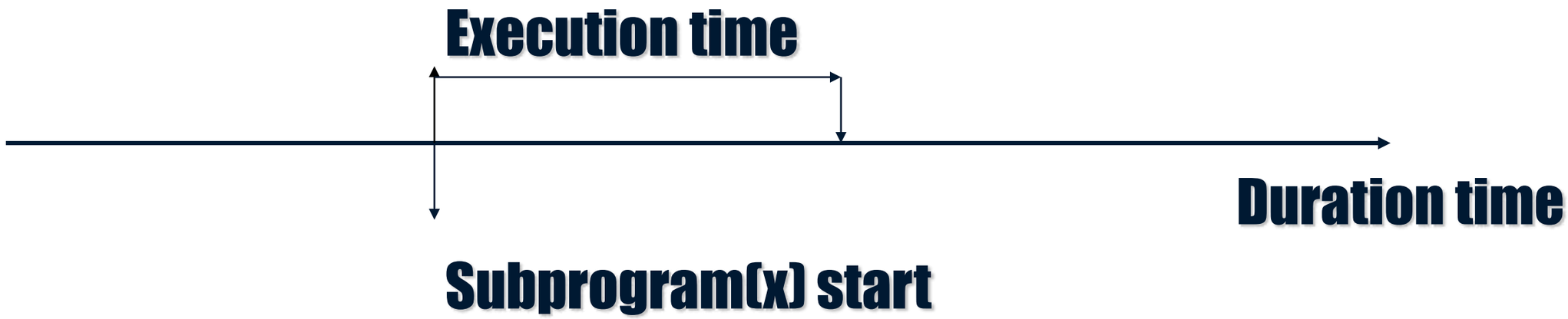
Subprogram : describes the interface to and the actions of the subprogram abstraction .

Subprogram call : is the explicit request that the called subprograms be executed .

Process abstraction are presented in programming languages by subprograms

What does it mean for a subproram to be active ?

Subprogram : is to be **active** if after having been called ,it has begun execution but has not yet completed that execution



Definitions :

A Subprogram header :

is the first line of the definition ,serves several purposes. First is specifies that the following syntactic unit is a subprogram's kind is often accomplished with a special word .

The header contains some keyword signalin the beginning of asubproram, a name for the subprogram ,and Header it may optionally specify alist of parameters

e.g: (Fortran)

Subroutine Adder (parameters)

Definitions :

e.g: (C)

Void adder (parameters)

,would serve as the header of a SUBPROGRAM named **adder** ,where **void** indicates that it does not return a value

- **Prototype :**is a subprogram declaration providing Interface information

Definitions :

Parameter profile:

Describes the number ,order , and types of the parameters

****also called “signature”.**

General subprogram characteristics :

- Each subprogram has a single entry point
- The calling program unit is suspend during The execution of the called subprogram , which means that there is only one subprogram in execution at any time
- Control always return to the caller when the subprogram execution terminates

Parameters :

There are two ways that subprogram can gain access to the data that it is to process:

- Direct access to nonlocal variables
- Parameter passing

** Parameter passing is more flexible than direct access to nonlocal variables

Parameters : (count …)

- **Formal parameters :**

The parameters in the subprogram header

- **Actual parameters :**

alist of parameters that appear in subprogram call statement

- **Positional parameters:**

is a method for binding actual parameter to formal parameter, is done by position

** ((the first actual parameters is bound to the first formal parameter and so forth such parameters are called **positional parameters**))

e.g :

M = rect (a , b ,c) // subproram call

Float rect (int x , int y , int z)

Parameters : (count …)

Keyword parameters :

The name of the formal parameter to which an actual parameter is to be bound is specified with the actual parameter

Advantage:

That they can appear in any order in the actual parameter list

Disadvantage:

That the user of the subprogram must know the names of formal parameters

Subprogram example:

```
Int cube(int);                          ← prototype
Int main(){
        Int y=5;                        ← actual parameters
      Cout<<cube(y);                    ← Subprogram call
        Int x=3;
        Int cube (int x);               ← subprogram header
{                                       ↑
                                        formal parameter
return x*x;
}
}
```

The Two kinds of subprograms :

There are **two** distinct categories of subprograms:

- **Procedures**
- **Functions**

Procedures :

Procedures :

are collection of statements that define parameterized computations , these computations are enacted by single call statements.

Procedure are called **subroutines**

Procedures has Two parts :

The speification and the Body

The specification is begins with the keyword PROCEDURE and ends With the procedure name or parameter list

e.G : **PROCEDURE** a_test(a,b : in integer ; c:out Integer)

Procedures :

Procedure identifier Rules :

- The name must start with aletter or made out of letters , numbers and the underscore(_).
- the name is always case-sensetive (uppercase / lowercase names are identical).
- the name may not start with the dollar-sign ($).

Examples of procedures

```
Program example1;

                                  Specification
   procedure add;
        var
          x : integer ;
          y : integer ;                    BODY
        begin
          read ( x , y );
          write ( x + y );
        end;

BEGIN
    add;
END .
```

```
Program example2 ;

Var
    x : integer ;
    y : integer ;

    Procedure add ( a : integer ; b : integer
    );
          begin
            write ( a + b );
          end;

BEGIN
    x = 10 ;
    y = 5 ;
    add ( 10 , 5 ) ;
END.
```

Functions :

Functions : structurally resemble procedure, But are semantically modeled on mathematical functions (Functions are procedures which return value).

** Function are **called** by appearances of their name in expressions

e.g: (function header and call)

```
Void sort (int list[], int listlen);       // function header
…
Sort(scores,100);          // function call
```

Functions :

** In function body The return statement end the execution of the function and return the value of the expression enclosed In the braces as a result

**The function body can contain any number of return statement

Functions :

```
// function example :
Function minimum ( A,B : Integer) return integer is
Begin
If A<= B then
Return A ;
Else
Return B ;
End if ;
End minimum;
```

Design Issues For Functions :

There are **Two** design issues are specific to functions :

- **Functional side effect**

The evaluation of a function effect the the value of other operands in the same expression.

** (solution parameters to functios can have only in mode formal parameter)

- **Types of returned value**

most imperative programming languages restrict the types that can be returned by their functions .

- C allow any type to be returned by its function except arrays and functions “both of these can be handled by pointer type return values ”
- Ada language its function can return values of any type

QUESTIONS?

Design issues for subprogram :

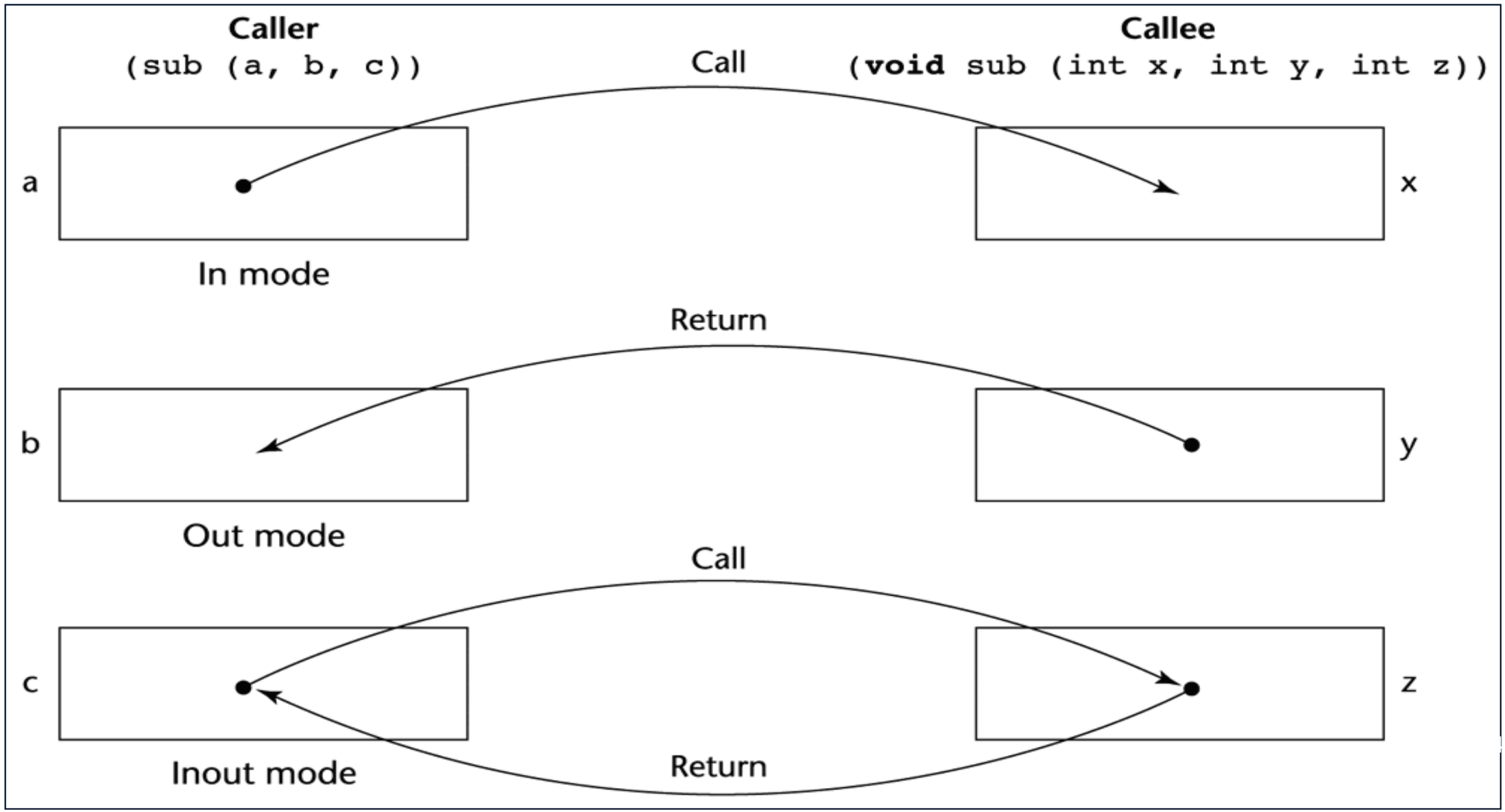
- What parameter passing method use ?
- Are the type of the actual parameter checked against the formal parameter ?
- Are local variables statically or dynamically allocated?
- If subprogram can be passed as parameters what is the referencing environment of such subprogram ?
- Can subprogram be overloaded ?
- Can subprogram be generic ?
- Is either separate or independent compilation possible ?

Parameter passing methods :

- Parameter passing methods are the ways in which parameters are transmitted to and/or from calling subprogram,
- **Formal parameters are characterized by one of three distinct semantics models :**

1. They can receive data from the corresponding actual parameter (in mode).
2. They can transmit data to the actual parameter (out mode).
3. They can do both (inout mode).

Graph for parameter passing methods :



Implementation models for parameter passing :

- Pass by value
- Pass by result
- Pass by value result
- Pass by reference
- Pass by name

Pass by value :

When parameter is passed by value ,the value of the actual parameter is used to initialize the correspondin formal parameter ,which then act as alocal variable in the subprogram

- **Advantage:** simple mechanism,actual parameter can be expression or values, (formal parameters are write protected) does not cause side effect
- **Disadvantage:** needs additional storage for formal parameters .

Pass by result :

- Formal parameter acts as local variable just before control returns to caller , value is passed to actual parameter , which must be a variable
- **problem #1:**

Actual parameter collision , such as call sub(p1,p1) may return different values to p1 depending on order of assignment

- **Problem #2:**

Address evaluation may be done at call or at return, for example list[index] if the index is changed by the subprogram ,either through global access or as formal parameter ,ten the address of list[index] will changed between the call and the return

Pass by value result :

- The value of the actual parameter is used to initialize the corresponding formal parameter ,the value of the formal parameter is transmitted back to the actual parameter

- Pass by value result is some times call **pass by copy**

- Pass by value result **share** with **pass by value** and pass by result the disadvantage of requiring multiple storage for parameters and time for copying values .

- Pass by value result **share** with **pass by result** the problems associated with the order in which actual parameters are assigned .

Pass by reference :

- In pass by reference method transmits an access path ,usually just an address , to the called subprogram

- The called subprogram is allowed to access the actual parameter in the calling program unit .

Pass by reference : (count …)

- **The advantage of Pass by reference :**

  Passing process is efficient , in term of both time and space , duplicate space is not required , nor is any copying required .

- **The disadvantage of Pass by reference :**

  1. access to the formal parameters will be slower ,because of the additional level of indirect addressing that is required

  2. if only one way communication to the called subprogram is required ,inadvertent and erroneous changes may be made to the actual parameter

  3. the aliases can be created

Pass by name :

- The actual parameter is textually substituted for the formal parameter in all its occurrences in the subprogram .

Advantage:

- More flexible than other methods.

Disadvantage :

- Relatively slow than other methods
- Very expensive

Parameter passing example:

```
Program ParamPassing;
var
i : integer;
a : array[1..2] of integer;

procedure p(x, y : integer);
begin
x := x + 1;
i := i + 1;
y := y + 1;
end; {p}

begin {ParamPassing}
a[1] := 1;
a[2] := 1;
i := 1;
p(a[i], a[i]);
writeln('a[1] = ', a[1]);
writeln('a[2] = ', a[2]);
end.
```

Pass by value:
a[1] = 1
a[2] = 1

Pass by result:
x and y have no initial values

Pass by value-result:
a[1] = 2
a[2] = 1

Pass by reference:
a[1] = 3
a[2] = 1

Pass by name:
a[1] = 2
a[2] = 2

Overload Subprogram

- **Overload Subprogram** is a subprogram that has the same name as another subprogram in the same referencing environment

- Overloaded subprograms that have default parameters can lead to ambiguous subprogram calls.

- Example: (C++)

```
void fun( float b = 0.0 );
void fun();
…
fun ( );
```

QUESTIONS?

Type checking parameter :

- Without type checking ,some typographical errors lead to program error difficult to diagnose because they are not detected by the compiler or the run time system.
- Early programming language ,such as FORTRAN 77 and the original version of c did not required parameter type checking ,most later language require it , the relatively recent language Perl, JavaScript , and PHP do not

Example of type checking parameter :

double sin(x)
  double x;
  { …. }

Using this method avoids type checking, thereby allowing calls such as:

Double value;
int count;
…
Value = sin (count);

To be legal, although they are never correct

Subprogram passed as parameters:

Some language allow nested subprogram **((allow subprogram to be a parameter for another subprogram))**

So in this case the following choices for referencing environment:

1. **Shallow binding :** the environment of the call statement that enacts the passed subprogram.
2. **Deep binding :** the environment of the definition of the passed subprogram
3. **Ad hoc binding :** the environment of the call statement that passed the subprogram as an actual parameter

Generic Subprogram

- A generic or polymorphic subprogram is one that takes parameters of different types on different activations.
- Overloaded subprograms provide ad hoc polymorphism .
- A subprogram that takes a generic parameter that is used in a type expression that describes the type of the parameters of the subprogram provides parametric polymorphism .

Advantages to using subprograms

- There are several advantages to using subprograms:
  - They help keep the code simple, and, thus, more readable;
  - They allow the programmer to use the same code as many times as needed throughout the program;
  - They allow the programmer to define needed functions; and,
  - They can be used in other programs

THANK YOU