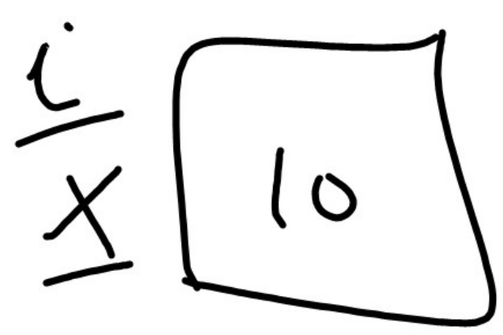
```
int i=3;
void f(x){
  {
    i++;
  }
  f(a[i]);
}
```

```
int i;

void f(int & x){
  ~
}

f(i);
```



Subprograms

Terminology

- Subprograms are also known as procedures, subroutines, functions
- Design issues apply also to methods
- Formal parameters: Variables in the function header
- Actual parameters: arguments in the function call statement

Keyword Parameters

- Association between formal and actual parameters
- As in Ada call statement
- **f(length=>my_length, list=>my_array, sum=>my_sum);**
- Parameters can be listed in any order
- Reduces errors that involve passing an actual parameter to the incorrect formal parameter

Parameters with Default Value

- Ada Example

```
function compute_pay(income: float;
exemptions: integer:=1; tax_rate: float)
return float;
```

- All actual parameters after an abscent actual parameter must be keyworded.

```
pay:=compute_pay(2000.0,tax_rate=>0.15)
```

- In C++ the default parameter must be last

```
float compute_pay(float income, float
tax_rate, int excemptions=1){…}
```

Function Design Issues

- What parameter passing method(s) are used?
- Are the types of actual parameters checked against the types of formal parameters?
- Can a subprogram appear inside another subprogram?
- If subprograms can be passed as parameters and subprograms can be nested, what is the referencing environment of a passed subprogram?
- Can subprograms be overloaded?
- Can subprograms be generic?

Parameter Passing Methods

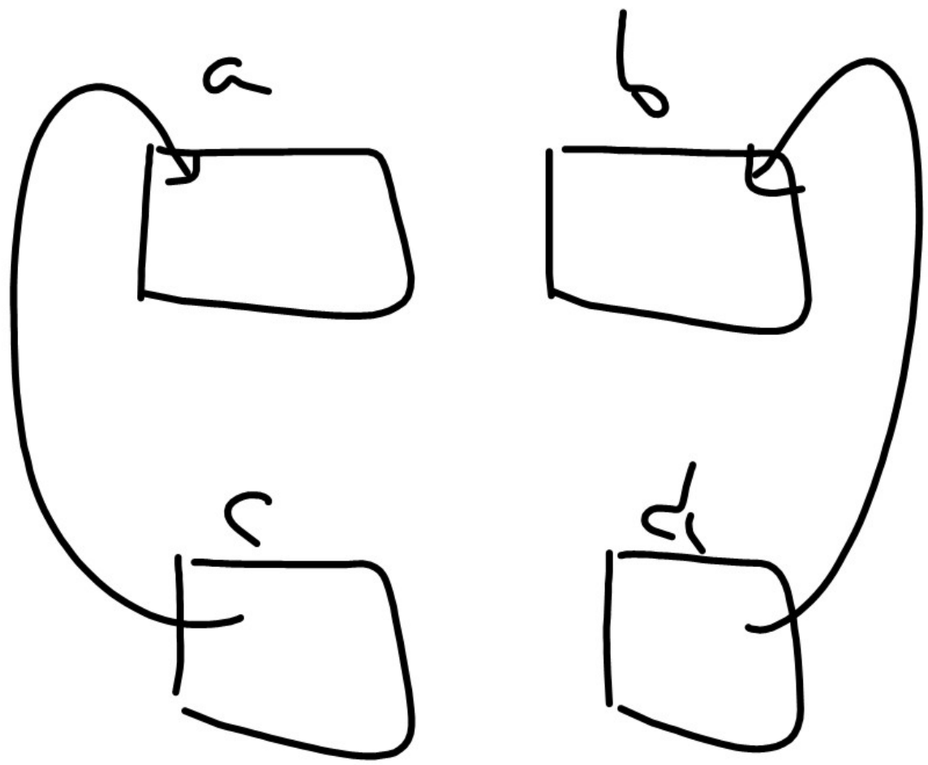
- Determines how data is transferred between actual and formal parameters
- There are 5 methods
  - Pass by value
  - Pass by result
  - Pass by value-result
  - Pass by reference
  - Pass by name

Pass by Value

- The formal parameter has an independent location from actual parameter and is stored on the stack
- Changes to formal parameter inside function do not affect formal parameter
- The value of the actual parameter is copied to corresponding formal parameter
- Copying occurs at beginning of function execution
- The only method in C

Example

```
void swap(int a, int b){
  int temp=a;
  a=b;
  b=temp;
}
swap(c,d);
```



Equivalent to:

a=c; b=d; temp=a; a=b; b=temp;

Evalution

- Data is stored twice in memory
- Extra execution time to perform data copying. Consider an array of 1000 large structures

Pass by Result

- The formal parameter has an independent location from actual parameter and is stored on the stack
- The value of the formal parameter is copied to corresponding actual parameter
- Copying occurs at end of function execution
- No copying occurs at beginning of function execution
- The actual parameter must be a variable
- Supported in C#

  public static void test(**out** int X){ … }

Ada Example

- procedure fact(x: **in** integer; y: **out** integer);
  var i: integer;
  begin
    y:=1;
    for i:=1 to x do y:=y*i;
  end;

value

result

24

...

fact(4,R);

write(R); 4 24

---

x=4; addr_R = &R;

y=1; for i=1 to x do y=y*i;

*addr_R = y; write(R);

Evaluation

- Memory and execution time problems are similar to pass by value
- Two additional problems that might reduce portability
  - Final value of actual parameter can be compiler/ interpreter dependent

```
procedure p(x: out integer, y: out integer);
...
p(a, a);
```

The final value of **a** can be the value of x or the value of y

- For the actual parameter A[i], the implementer has two choices to compute &A[i], before and after function execution

Pass by Value Result

- A combination of pass by value and pass by result
- Formal parameter also has independent location from formal parameter
- At the beginning of function execution value of actual parameter is copied to the corresponding formal parameter
- At the end of function execution value of formal parameter is copied to the corresponding actual parameter

Ada Example

```
procedure square(N: inout integer);
begin
   N:=N*N;
end;

...
i:=4;
square(i);
write(i);
```

```
i=4; N=i; addr_i=&i; N=N*N; *addr_i=N;
write(i);
```

Evaluation

- Memory problem is similar to value and result methods
- Execution time is worse since copying occurs twice
- Compiler/interpreter dependence is similar to pass by result

Pass by Reference

- No copying of data
- The formal parameter has no independent location. It refers to the same location of actual parameter
- Modifications of formal parameter are modifications to actual parameter
- Simulated in C using pointers
- Supported in C++ using **&**, in C# using **ref**, and in Pascal using **var**

C++ Example

```
void f(int &x){
    x=x+x;
}
int y=5;
f(y);
cout<<y; // 10
```

reference

value-result

```
y=5;
x=&y;
*x=*x + *x;
cout<<y;
```

Evaluation

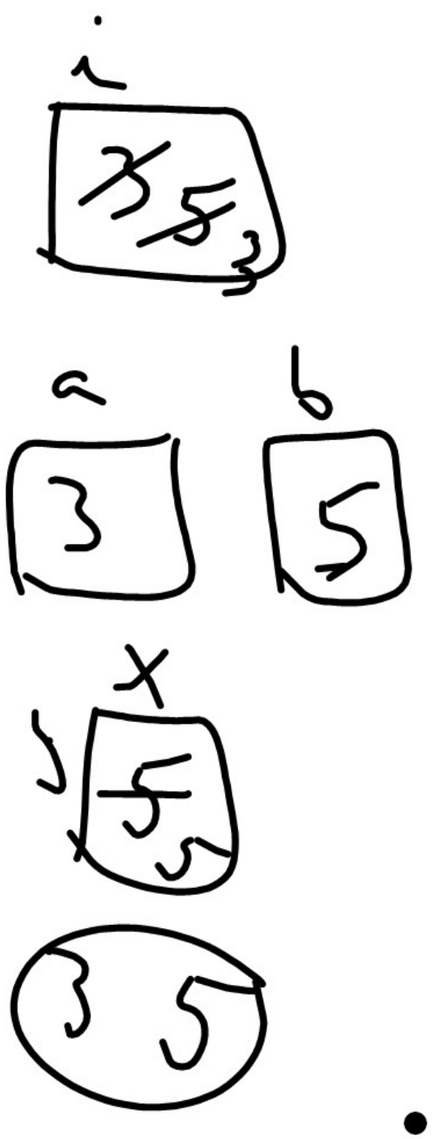
- Good for memory since data is stored only once
- Good for execution time since no copying occurs
- Aliases might be created
  - Makes the program more difficult to verify and hence reduces reliability
  - Negative for readability

Value-Result vs. Reference

- The two methods often produce the same results
- Languages that support value-result do not support reference. Example Ada
- Languages that support reference do not support value-result. Example C++, C#
- The two methods might produce different results when aliasing is involved

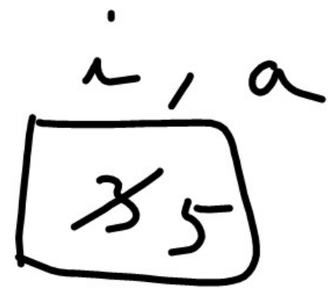
Example

```
int i=3;
void fun(int a, int b){
   i=b;
}
void main(){
   int x=5;
   fun(i, x);
   write(i,x);
}
```



- Using pass by value result:

```
addr_i = &i;
addr_x = &x;
a = i;
b = x;
i = b;
*addr_i = a;
*addr_x = b;
```



The value of i changes in fun from 3 to 5, but the copy back of the first formal parameter sets it back to 3

- Using pass by reference i remains 5

Pass by Name

- Actual parameter is often an expression
- Formal parameter has no independent location in memory
- No data copying
- The actual parameter is textually substituted for the corresponding formal parameter in all its occurrences in the subprogram before execution
- Found in Algol 60 and Scala (2004)
- www.scala-lang.org

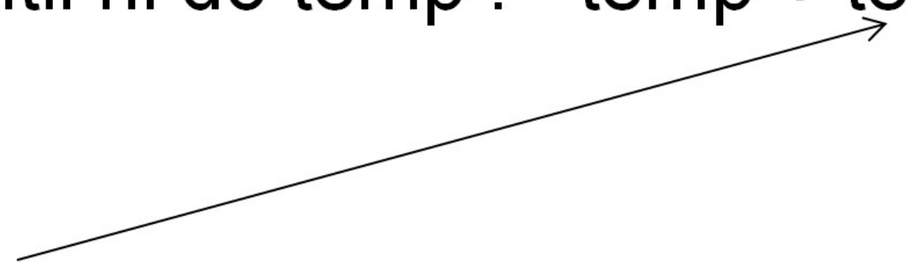
Example

```
int global;
int list[0..1];
void f(param){
    int param;
    param=3;
    global=global+1;
    param=5;
}
void main(){
    list[0]=1;
    list[2]=1;
    global=0;
    f(list[global]);
    write (list[0], list[2]);
}
```

- Using pass by name prints 3 5
- Using pass by reference prints 5 1

Pass by Name in Algol 60

```
begin
    integer i;
    real procedure sum (i, lo, hi, term);
        value lo, hi; // i, term by name by default
        integer i, lo, hi;
        real term;
    begin
        real temp;
        temp:=0;
        for i:= lo step 1 until hi do temp := temp + term;
        sum := temp;
    end;
    print (sum (i, 1, 100, 1/i))
end
```



C Macros

- Define short code patterns
- Work similar to pass by name
- Expanded by preprocessor
- C example

```
#define swap(a,b) temp=a; a=b; b=temp;
#define square(x,y,z) x*x+y*y+z*z

…
int X=3, Y=4, temp;
swap(X,Y);
k=square(i,j,k);
```

Name vs. Value

- By value always evaluates actual parameter before function executes and is called **eager** evaluation
- By name postpones evaluation of actual parameter until when only needed and is called **lazy** evaluation

Example

```
void f(X){
    …
    if (cond1) write(X);
    …
}
```

…

f(g(a)*g(b)*10 – h(a*b)); //expensive computation

- Eager evaluation performs unnecessary computation if cond1 is false
- Lazy evaluation would avoid the expensive evaluation if cond1 is false

Example

```
void f(X){
   if (cond1) write(X);
   if (cond2) Y=X;
   if (cond3) Z=X-1;
   …
}
…
f(g(a)*g(b)*10 – h(a*b)); //expensive computation
```

Which would be more efficient eager or lazy evaluation?

Passing Methods in Major Languages

- Fortran 95 and Ada provide in, out, inout passing methods
- C++ provide pass by value and pass by reference
- Java, all parameters are passed by value. Since objects can be accessed through reference variable, objects in effect are passed by reference. An object reference actual parameter cannot change after executing a function but the referenced object can change. Scalars cannot be passed by reference

Passing Methods in Major Languages

- C# default is pass by value. Pass by reference is provided by preceding both actual and formal parameters with ref.

  void sumer(ref int oldsum, int newone){…}

  …

  summer(ref sum, newvalue);

- PHP is similar to C# except that either the actual parameter or the formal parameter can specify pass by reference by preceding the parameter(s) with &.