Pass-by-reference

- The caller must copy the **reference** (i.e., address) of the actual parameter to the agreed location for the parameter
- The callee must use the parameter **assuming** that the parameter contains an **address** of a variable
- Example of pass-by-value in C++: click here

  Remark:

  - did you see the difference in the **language** on how to specify "pass-by-value" and "pass-by-reference" ?
  - Just in case that you did not notice: find the (rather insignificant in size) "&" in

    ```
    void func(int& x)
    ```

  - You will see that this little "&" will have a dramatic impact on the result because the **address** is passed !

  NOTE: get a copy, compile it and **run** it to see the effect !
  The variable **i** in main() **is incremented** when func() returns !

- Here is what happens in pass-by-reference exposed in Assembler code:

```
Assume the following agreement: parameter passed in D0
                                parameter passed by reference

C++ code:                              Assembler code:
---------                              --------------

main()                          main:
{                                      MOVE.L  #i, D0   // pass addr
   int i;                                                 // of var i

   func(i);  // pass addr              BSR     func ------+
             // of var i               ....               |
}                                                       |
                                                        |
void func( int& x )             func:              <-----+
{                                      // Assume addr is passed
   x = x + 1;                          // So D0 = address of param
}                                      // We have to go get the value...

                                       MOVE.L  D0, A0
                                       MOVE.L  (A0), D1  // Now D1 = x

                                       ADD.L  #1, D1     // D1 = x + 1

                                       // Don't stop yet...
                                       // Program says: put x+1 in x !

                                       MOVE.L  D1, (A0)  // Update !!!
```

```
RTS
```

- You can clearly see in the assembler code why the variable **i** in main() **WILL BE** incremented !

  Assembler programming can help you understand "mysterious" concepts in computer science...

- So: if you pass a parameter by reference, the callee function will **know** the location of the variable and can **update** the variable if it sees fit (depends on what the callee function will do)...

- Example of pass-by-value: sum of squares again...

```
// func(x,y) computes x^2 + y^2
int func(int& x, int y)   <---- x is passed by reference, y by value
{
    x = x + 1;
    y = y + 1;

    return(x*x + y*y);
}

main()
{
   int a, b, c;
   int i, j, k;

   c = func(a,b);
   k = func(i,j);
}
```

  - First agree on where to pass the parameters:
    - param 1: in D0, **by reference**
    - param 2: in D1, by value
  - Next agree on where to pass the return value back:
    - return value: in D7
  - Now we can write the program...

```
main:
      MOVE.L #a, D0     // Pass param 1 by ref
      MOVE.L b, D1      // Pass param 2 by value
      BSR    func

      MOVE.L D7, c      // c = func(a,b)

      MOVE.L #i, D0     // Pass param 1 by ref
      MOVE.L j, D1      // Pass param 2 by value
      BSR    func

      MOVE.L D7, k      // k = func(i,j)
      ....
      ....

func:
      MOVEA.L D0, A0    // D0 = address of variable x
      MOVE.L  (A0), D2  // Now D2 = value of variable x
      ADD.L   #1, D2    // Computed x+1
      MOVE.L  D2, (A0)  // Stores: x = x + 1

      MOVEA.L D1, A0    // D1 = address of variable y
```

```
MOVE.L  (A0), D2  // Now D2 = value of variable y
ADD.L   #1, D2    // Computed y+1
MOVE.L  D2, (A0)  // Stores: y = y + 1

MOVEA.L D0, A0    // D0 = address of variable x
MOVE.L  (A0), D2  // Now D2 = value of variable x
MULS    D2, D2    // D2 = x*x

MOVEA.L D1, A0    // D0 = address of variable y
MOVE.L  (A0), D3  // Now D3 = value of variable y
MULS    D3, D3    // D2 = y*y

ADD.L   D3, D2     // x*x + y*y

MOVE.L  D2, D7     // Put return value in the agreed location

RTS
```

- Here is the code that you can run: click here

- Here is my CS170 lecture notes on **Pass-by-reference** if you want to review: click here

---