1 1/2

Subroutines with Parameters

- Parameters are used to pass information between the caller and the callee.

- The way that the caller and callee pass input parameters and output (result) to each other is through a rigid agreement (contract).

- Caller ~~agree~~ agrees to put the input parameters in a specific location, and the callee agrees to get the input parameters from that location.

- Similarly, callee agrees to put return values in a specific location, and the caller agrees to get the return values from that ~~location~~ location.

Romeo ⇔ Juliet
meeting place agreement.

Q: What can you use to pass parameters?

A: 3 places:

(1) registers. — non-recursive subroutines

(2) memory (rarely used) — non-recursive

(3) stack !!! — recursive subroutines.

special place in memory.

- We will first look at non-recursive subroutines and only at how parameters are passed in registers.

Let's look at the different ways to pass parameters. in registers — by value
\ by reference.

```
main()
  { int a,b,c,d;

      add a,b and put sum in 'sum'
      add c,d and put sum in 'sum'

  }

Add(x,y)
  { add x,y and return result in sum}.
```

There are many ways to accomplish this.

Method 1:

sub1.5 by value

```
main()
{ int a, b, c, d, sum;

    sum = Add(a,b);
    sum = Add(c,d);

}

Add(int x, int y)
{ return(x + y);
}
```

Do Add FIRST

```
f()
{ int c, d, sum
    sum = Add(c,d);
}
```

a & b are passed by value

i.e.: Caller passes the value of the parameters to the callee.

Key: make agreement on:

(1) where to put Param 1, 2, 3... etc

(2) How params are Passed.

```
*
* sub1.s: Add subroutine
*         input: d0 = first number
*                d1 = second number
*         output: d0 = sum
*
        xdef start, stop, pause1
        xdef a, b, c, d, sum

start:
        move.l  a, d0           ; put first parameter in agreed location
        move.l  b, d1           ; put second parameter in agreed location
        jsr     Add
        move.l  d0, sum         ; get sum from agreed return location
pause1:
        move.l  c, d0           ; put first parameter in agreed location
        move.l  d, d1           ; put second parameter in agreed location
        jsr     Add
        move.l  d0, sum         ; get sum from agreed return location

stop:   nop

a:      dc.l 23
b:      dc.l 12
c:      dc.l 11
d:      dc.l 67
sum:    ds.l 1


* ----------------------------------- This is the only way to show clearly
* ----------------------------------- where a subroutine starts !!!!!!!!!
* Subroutine Add
* Input: d0 = first integer
*        d1 = second integer
* Output: d0 = sum
* ----------------------------------- If that's not clear enough, I don't
* ----------------------------------- know what else to do.......
Add:
        add.l d1, d0            ; Add inputs and put result in agreed location
        rts

        end

.gdbinit:
break start
break stop
break pause1
disp/dw &sum
disp/dw &d
disp/dw &c
disp/dw &b
disp/dw &a
disp/i $pc
```

Method 2:

sub1a.s
sub1b.s
by reference

```
main ()
{ int a, b, c, d, sum

    sum = Add (&a, &b);
    sum = Add (&c, &d);

}

Add ( int *x, int *y)
{
    return (*x + *y);
}

f()
{
    int c, d, sum
    sum = Add(c, d);
}
```

Variables are passed by reference

ie: you pass the address of the variables to the callee.

```
*
* sub2a.s: Add subroutine
*          input: d0 = ADDRESS of the first integer
*                 d1 = ADDRESS of the second integer
*          output: d0 = sum
*
        xdef start, stop, pause1
        xdef a, b, c, d, sum

start:
        move.l  #a, d0          ; put first parameter in agreed location
        move.l  #b, d1          ; put second parameter in agreed location
        jsr     Add
        move.l  d0, sum         ; get sum from agreed return location
pause1:
        move.l  #c, d0          ; put first parameter in agreed location
        move.l  #d, d1          ; put second parameter in agreed location
        jsr     Add
        move.l  d0, sum         ; get sum from agreed return location

stop:   nop

a:      dc.l 23
b:      dc.l 12
c:      dc.l 11
d:      dc.l 67
sum:    ds.l 1


* ------------------------------------ This is the only way to show clearly
* ------------------------------------ where a subroutine starts !!!!!!!!!
* Subroutine Add
* Input: d0 = ADDRESS of the first integer
*        d1 = ADDRESS of the second integer
* Output: d0 = sum
* ------------------------------------ If that's not clear enough, I don't
* ------------------------------------ know what else to do.......
Add:
        movea.l d0, a0          ;
        move.l (a0), d7         ; get first number
        movea.l d1, a0          ;
        add.l (a0), d7          ; add second number to the first
        move.l d7, d0           ; put sum in agreed return location
        rts

        end

.gdbinit:
break start
break stop
break pause1
disp/dw &sum
disp/dw &d
disp/dw &c
disp/dw &b
disp/dw &a
disp/i $pc
```

```
*
* sub2b.s: Add subroutine
*         input: A0 = ADDRESS of the first integer
*                A1 = ADDRESS of the second integer
*         output: d0 = sum
*
        xdef start, stop, pause1
        xdef a, b, c, d, sum

start:
        move.l  #a, a0          ; put first parameter in agreed location
        move.l  #b, a1          ; put second parameter in agreed location
        jsr     Add
        move.l  d0, sum         ; get sum from agreed return location
pause1:
        move.l  #c, a0          ; put first parameter in agreed location
        move.l  #d, a1          ; put second parameter in agreed location
        jsr     Add
        move.l  d0, sum         ; get sum from agreed return location

stop:   nop

a:      dc.l 23
b:      dc.l 12
c:      dc.l 11
d:      dc.l 67
sum:    ds.l 1


* ---------------------------------- This is the only way to show clearly
* ---------------------------------- where a subroutine starts !!!!!!!!!
* Subroutine Add
* Input: a0 = ADDRESS of the first integer
*        a1 = ADDRESS of the second integer
* Output: d0 = sum
* ---------------------------------- If that's not clear enough, I don't
* ---------------------------------- know what else to do.......
Add:
        move.l (a0), d7         ; get first number
        add.l (a1), d7          ; add second number to the first
        move.l d7, d0           ; put sum in agreed return location
        rts

        end

.gdbinit:
break start
break stop
break pause1
disp/dw &sum
disp/dw &d
disp/dw &c
disp/dw &b
disp/dw &a
disp/i $pc
```

Method 3:

```
main()
{ int a, b, c, d, sum;

  Add(a, b, &sum);
  Add(c, d, &sum);
}

Add(int x, int y, int *sum)
{ *sum = a + b; }
```

a & b are passed by value
sum is passed by reference.

```
*
* sub3.s: Add subroutine
*         input: d0 = the first integer
*                d1 = the second integer
*                a0 = address of the sum variable
*         output: None. Add will update the sum variable
*
        xdef start, stop, pause1
        xdef a, b, c, d, sum

start:
        move.l  a, d0           ; put first parameter in agreed location
        move.l  b, a1           ; put second parameter in agreed location
        move.l  #sum, a0        ; put third parameter in agreed location
        jsr     Add
pause1:
        move.l  c, d0           ; put first parameter in agreed location
        move.l  d, d1           ; put second parameter in agreed location
        move.l  #sum, a0        ; put third parameter in agreed location
        jsr     Add

stop:   nop

a:      dc.l 23
b:      dc.l 12
c:      dc.l 11
d:      dc.l 67
sum:    ds.l 1


* ---------------------------------- This is the only way to show clearly
* ---------------------------------- where a subroutine starts !!!!!!!!!
* Subroutine Add
* Input: d0 = first integer
*        d1 = second integer
*        a0 = address of sum variable
* ---------------------------------- If that's not clear enough, I don't
* ---------------------------------- know what else to do.......
Add:
        add.l d1, d0            ; add second number to the first
        move.l d0, (a0)         ; update the sum
        rts

        end

.gdbinit:
break start
break stop
break pause1
disp/dw &sum
disp/dw &d
disp/dw &c
disp/dw &b
disp/dw &a
disp/i $pc
```

Method 4

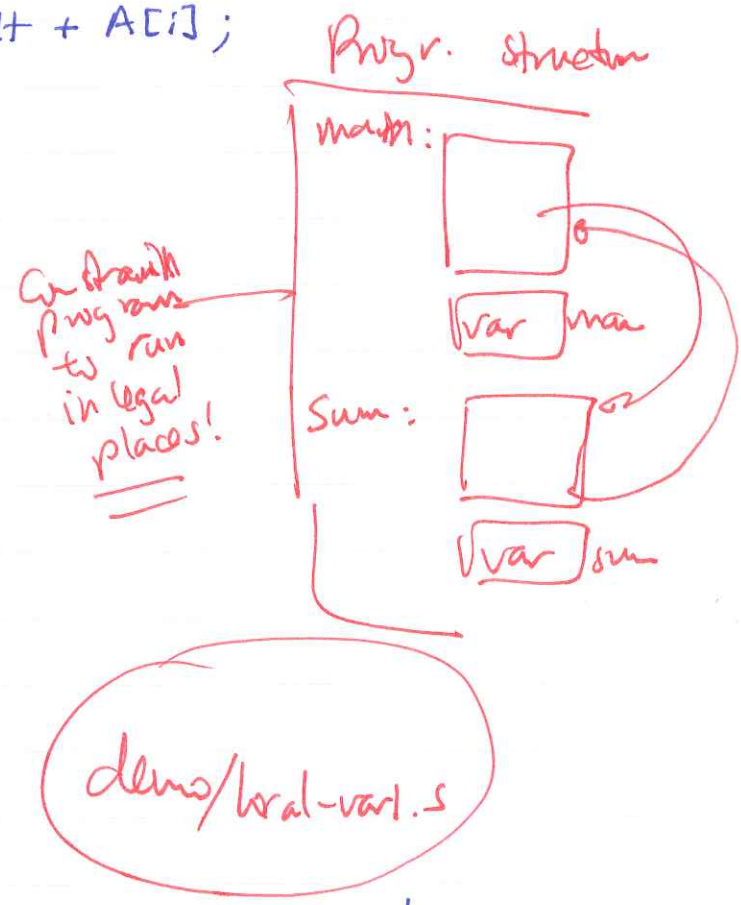
```
main ( )
{ int a, b, c, d, sum;

    Add ( &a, &b, &sum);
    Add ( &c, &d, &sum);

}

Add ( int *x, int *y, int *sum)

{ *sum = *x + *y }
```

```
*
* sub3.s: Add subroutine
*         input: A0 = ADDRESS of the first integer
*                A1 = ADDRESS of the second integer
*                A2 = ADDRESS of the sum variable
*         output: None. Add will update the sum variable
*
        xdef start, stop, pause1
        xdef a, b, c, d, sum

start:
        move.l  #a, a0          ; put first parameter in agreed location
        move.l  #b, a1          ; put second parameter in agreed location
        move.l  #sum, a2        ; put third parameter in agreed location
        jsr     Add
pause1:
        move.l  #c, a0          ; put first parameter in agreed location
        move.l  #d, a1          ; put second parameter in agreed location
        move.l  #sum, a2        ; put third parameter in agreed location
        jsr     Add

stop:   nop

a:      dc.l 23
b:      dc.l 12
c:      dc.l 11
d:      dc.l 67
sum:    ds.l 1


* ---------------------------------- This is the only way to show clearly
* ---------------------------------- where a subroutine starts !!!!!!!!!
* Subroutine Add
* Input: a0 = ADDRESS of the first integer
*        a1 = ADDRESS of the second integer
*        a2 = ADDRESS of the sum variable
* ---------------------------------- If that's not clear enough, I don't
* ---------------------------------- know what else to do.......
Add:
        move.l (a0), d7         ; get first number
        add.l (a1), d7          ; add second number to the first
        move.l d7, (a2)         ; update the sum
        rts

        end

.gdbinit:
break start
break stop
break pause1
disp/dw &sum
disp/dw &d
disp/dw &c
disp/dw &b
disp/dw &a
disp/i $pc
```

Subroutines with local variables (Non-recursive).

Example: write a subroutine that ~~sum~~ returns the sum of an array A of N elements.

```
int sum ( int A[], N)
  { int result, i ;
      result = 0 ;
      for (i = 0 ; i < N ; i++)
          result = result + A[i] ;
      return (result);
  }

main ()
  { int B[10];
    int S ;

    S = sum(B, 10) ;
  }
```



Decision:

(1) A is passed by reference. in d0

(2) N is passed by value in d1

(3) result is returned in d0.

- The way that main calls sum is then as follows:

```
move.l  #B, d0
move.l  #10, d1
bsr     sum
```

} sum(B, 10)

```
move.l  d0, S
```

} put return value in S.

- The subroutine sum has 2 local variables:

result and i.

We can define these variables as:

```
result : ds.l 1
i :      ds.l 1
```

— program using these variables.

Alternately, we can associate with each variable a register and program the subroutine sum to use registers

- Warning: reserving storage for local variables using ds or registers only works with NON-RECURSIVE functions.

```
* sub5.s
*
* Illustrates local variables in non-recursive functions.
* Here, we put local variables in memory.
*
        xdef start, stop, B, s, result, i
        suba.l #80, SP          ; So I don;t crash when displaying the stack
* -------------------------- Main
* main()
*  { int B[10], s;
*    s = sum(B, 3);
*  }
* --------------------------
start:
        move.l #B, d0
        move.l #10, d1
        bsr    sum              ; Call sum with B and 3
        move.l d0, s            ; Put return value in s
stop:   nop

B:      dc.l 2, 1, 3, 4, 1, 2, 1, 1, 2, 2
s:      ds.l 1

* ************************** Subroutine sum with local variables in memory
* int sum(int A[], int N)      ==> A[] in d0, N in d1
*                              ==> result in d0
*  { int result
*    int i
*
*    result = 0;
*    for (i = 0; i < N; i++)
*       result += A[i];
*    return result;
*  }
* --------------------------
* Note: d0 = address of array A[]
*       d1 = value of N
* **************************
sum:    move.l #0, result
        move.l #0, i

forloop:cmp.l i, d1             ; Test if i >= N
        ble    fordone          ; If so, exit for-loop

        move.l d0, a0           ; Get start of array in a0
        move.l i, d2            ; Compute offset
        muls   #4, d2           ;
        move.l 0(a0,d2.w), d2   ; Fetch A[i] in d2
        add.l d2, result        ; result := result + A[i]

        addq.l #1, i            ; increment i
        bra    forloop

fordone:move.l result, d0       ; Return subprogram return value in d0
        rts
* ----------------- Local variables for sum subprogram
result: ds.l   1
i:      ds.l   1
* ------------------------------------------------------  End of sum subroutine
        end
```

The sum subroutine with local variables in memory.

```
sum:    move.l  #0, result          | d0 = A
                                    | d1 = N
        move.l  #0, i

forloop: move.l  i, d7
         cmp.l   d1, d7             (d1 = N)   (done if: i ≥ N).
         bge     fordone

         move.l  d0, a0
         move.l  i, d7
         muls    #4, d7
         move.l  0(a0, d7), d7

         move.l  result, d6
         add.l   d6, d7
         move.l  d7, result

         move.l  i, d7
         add.l   #1, d7
         move.l  d7, i
         bra     forloop

fordone: move.l  result, d0         (return result in d0
         rts
result:  ds.l    1
i    :   ds.l    1
```

The sum subroutine with local variables in registers.

We use d7 for result
and d6 for i.

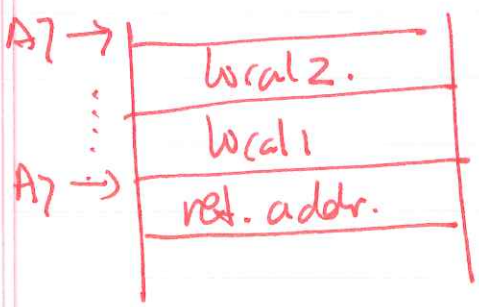
```
sum:    move.l  #0, d7

        move.l  #0, d6

forloop:
        cmp.l   d1, d6
        bge     fordone

        move.l  d0, a0
        move.l  d6, d5
        muls    #4, d5
        move.l  0(a0, d5), d4

        add.l   d4, d7

        add.l   #1, d6          (i = i+1)
        bra     forloop

fordone: move.l d7, d0          (return result in d0)
        rts.
```

Warning: Before you branch to a subroutine, make sure you save all important temporal results in registers to their corresponding variables.

eg. Because if you don't, ~~the subroutine~~ and the subroutine uses some of the registers for calculation, then your temporal results will be overwritten and ~~not~~ no longer available.

- Subroutines with ~~parameter~~ local variable on stack:

demo/local-var2.s



Dest: