CS 2211
Systems Programming

Part Four – A [Part 1]:

Function Memory

POINTERS

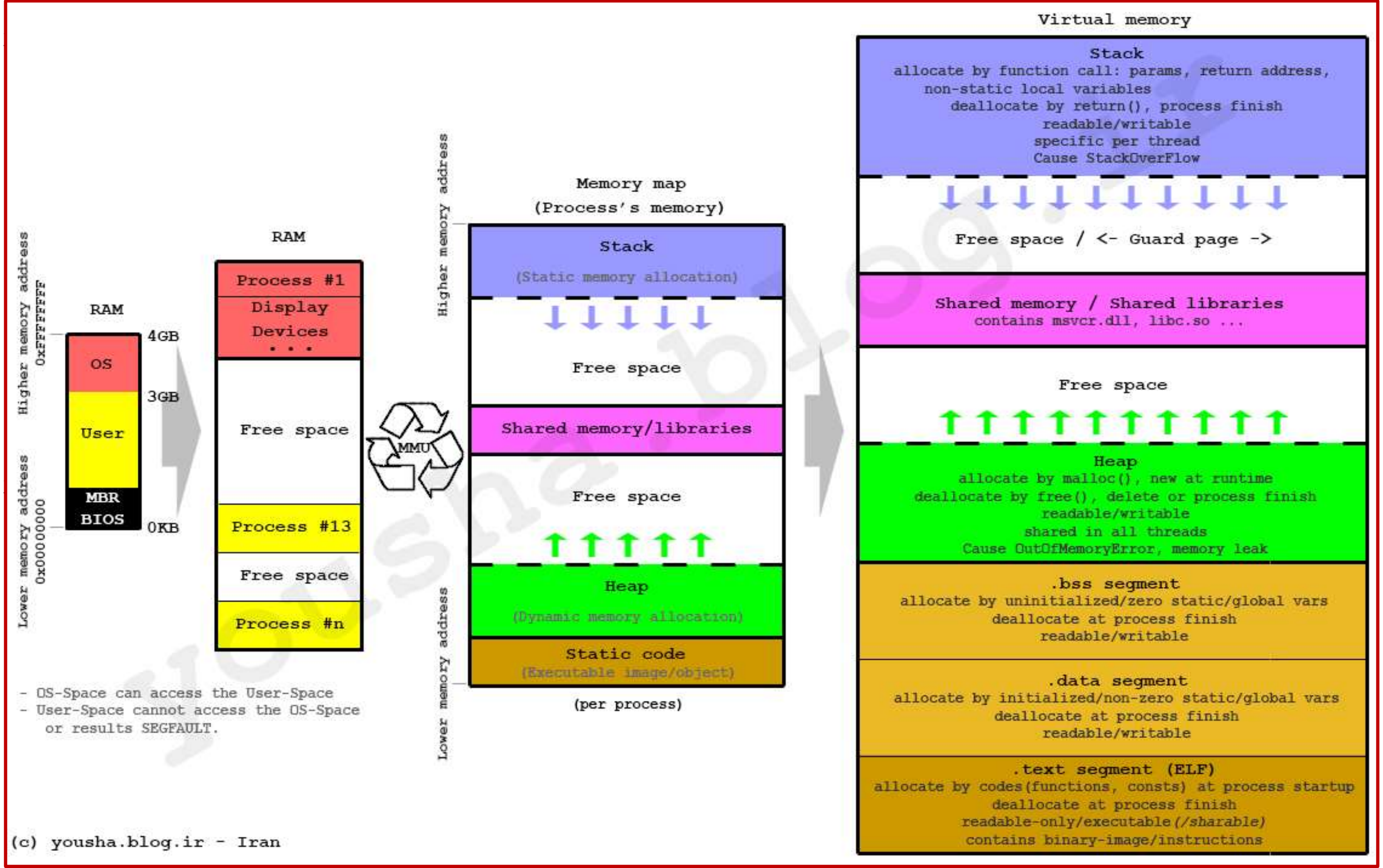
Passing Values TO and FROM a Function

```c
#include <stdio.h>

int division(int numerator, int denominator, int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d remainder\n",x,y,d,r);
     printf("x=%d\n",x);
}
```

POINTERS

Passing Values **TO** and **FROM** a Function

```
#include <stdio.h>

i

{





}


i
{









}
```



allocate by function call: params, return address,
non-static local variables
deallocate by return(), process finish
readable/writable
specific per thread
Cause StackOverFlow

contains msvcr.dll, libc.so ...

allocate by malloc(), new at runtime
deallocate by free(), delete or process finish
readable/writable
shared in all threads
Cause OutOfMemoryError, memory leak

allocate by uninitialized/zero static/global vars
deallocate at process finish
readable/writable

allocate by initialized/non-zero static/global vars
deallocate at process finish
readable/writable

allocate by codes(functions, consts) at process startup
deallocate at process finish
readable-only/executable(/sharable)
contains binary-image/instructions

- OS-Space can access the User-Space
- User-Space cannot access the OS-Space
  or results SEGFAULT.

(c) yousha.blog.ir - Iran

POINTERS

Passing Values TO and FROM a Function

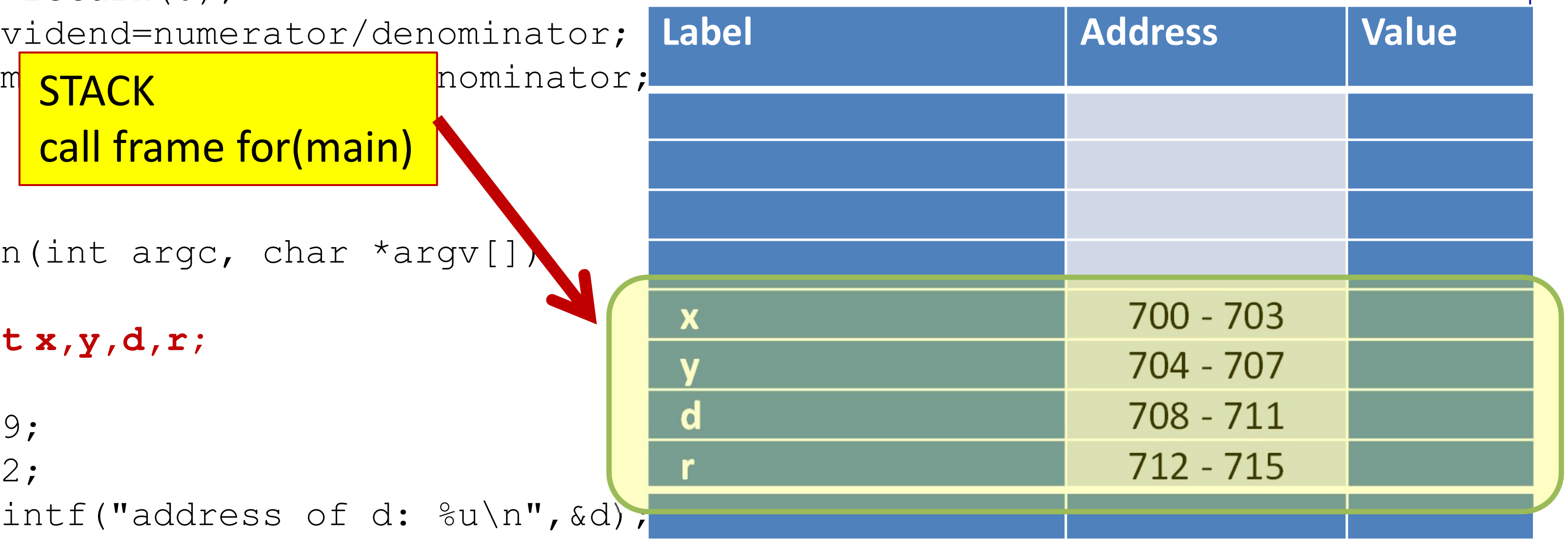
```
#include <stdio.h>

int division(int numerator, int denominator, int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d remainder\n",x,y,d,r);
     printf("x=%d\n",x);
}
```

Label	Address	Value
x	700 - 703	
y	704 - 707	
d	708 - 711	
r	712 - 715	

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int denominator, int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *rem                 nominator;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d remainder\n",x,y,d,r);
     printf("x=%d\n",x);
}
```



STACK
call frame for(main)

Label	Address	Value
x	700 - 703	
y	704 - 707	
d	708 - 711	
r	712 - 715	

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int denominator, int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d remainder\n",x,y,d,r);
     printf("x=%d\n",x);
}
```

Label	Address	Value
x	700 - 703	9
y	704 - 707	2
d	708 - 711	
r	712 - 715	

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int denominator, int *dividend, int *remainder)
{
    print                              %u\n",dividend);
    prin                                %u\n",remainder);
    if (c
        i
    *divi
    *rema
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d remainder\n",x,y,d,r);
     printf("x=%d\n",x);
}
```

output:

```
address of d: 708
address of r: 712
```

el	Address	Value
x	700 - 703	9
y	704 - 707	2
d	708 - 711	
r	712 - 715	

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int denominator, int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
            return(0)

    *
    *
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     prin                      \n",&d);
     prin                      \n",&r);
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d remainder\n",x,y,d,r);
     printf("x=%d\n",x);
}
```

function call (values filled)

```
division ( 9, 2, 708, 712 )
```

	Address	Value
x	700 - 703	9
y	704 - 707	2
d	708 - 711	
r	712 - 715	

```
( 9 , 2, 708, 712 )
```

POINTERS

Passing Values **TO** and **FROM** a Function

(9 , 2, *708,* *712*)

```
#include <stdio.h>

int division(int numerator, int denominator, int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);

","x,y,d,r);
```

Label	Address	Value
numerator	400 - 403	9
denominator	404 - 407	2
dividend	408 - 411	*708*
remainder	412 - 415	*712*
x	700 - 703	9
y	704 - 707	2
d	708 - 711	
r	712 - 715	

function call (values filled)

division (9, 2, *708, 712*)

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int denominator, int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
}


int m                                    ])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);

                                                 ",x,y,d,r);
```

Label	Address	Value
numerator	400 - 403	9
denominator	404 - 407	2
dividend	408 - 411	708
remainder	412 - 415	712
x	700 - 703	9
y	704 - 707	2
d	708 - 711	
r	712 - 715	

STACK
call frame for (division)

function call (values filled)

division (9, 2, *708*, *712*)

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int denominator, int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *                      nominator;
}

int main(int argc, char *argv[])
{
     int x,y,d,r;

     x
     y
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);

                                               ",x,y,d,r);
```

STACK
call frame for (division)

STACK
call frame for (main)

Label	Address	Value
numerator	400 - 403	9
denominator	404 - 407	2
dividend	408 - 411	708
remainder	412 - 415	712
x	700 - 703	9
y	704 - 707	2
d	708 - 711	
r	712 - 715	

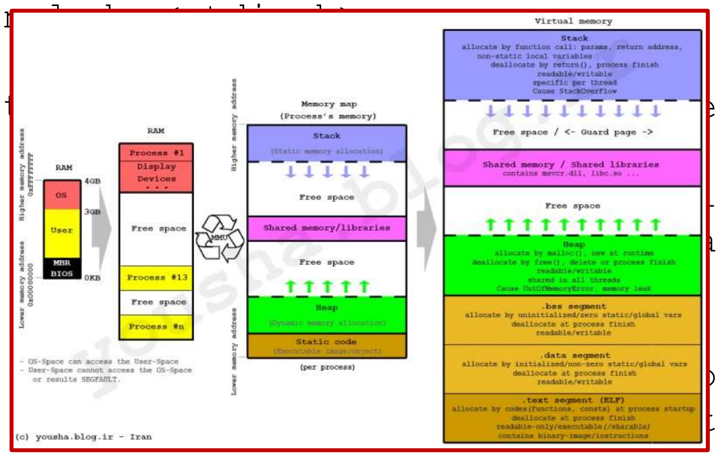
function call (values filled)

division (9, 2, *708*, *712*)

POINTERS

Passing Values TO and FROM a Function

```
#i

int                                    enominator, int *dividend, int *remainder)
{

                                       dend: %u\n",dividend);
                                       inder: %u\n",remainder);



                                       or;
                                       tor,

}
```



RAM
Higher memory address 0xFFFFFFFF
4GB
OS
3GB
User
MBR
BIOS
0KB
Lower memory address 0x00000000
RAM
Process #1
Display Devices
...
Free space
Process #13
Free space
Process #n
- OS-Space can access the User-Space
- User-Space cannot access the OS-Space or results SEGFAULT.
MMU
Higher memory address
Memory map (Process's memory)
Stack
(Static memory allocation)
Free space
Shared memory/libraries
Free space
Heap
(Dynamic memory allocation)
Static code
(Executable image/object)
(per process)
Lower memory address
Virtual memory
Stack
allocate by function call: params, return address, non-static local variables
deallocate by return(), process finish
readable/writable
specific per thread
Cause StackOverFlow
Free space / <- Guard page ->
Shared memory / Shared libraries
contains msvcr.dll, libc.so ...
Free space
Heap
allocate by malloc(), new at runtime
deallocate by free(), delete or process finish
readable/writable
shared in all threads
Cause OutOfMemoryError, memory leak
.bss segment
allocate by uninitialized/zero static/global vars
deallocate at process finish
readable/writable
.data segment
allocate by initialized/non-zero static/global vars
deallocate at process finish
readable/writable
.text segment (ELF)
allocate by codes(functions, consts) at process startup
deallocate at process finish
readable-only/executable(/sharable)
contains binary-image/instructions
(c) yousha.blog.ir - Iran

call frame (division)

```
int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);

                                          ",x,y,d,r);
```

call frame (main)

Label	Address	Value
numerator	400 - 403	9
denominator	404 - 407	2
dividend	408 - 411	708
remainder	412 - 415	712
x	700 - 703	9
y	704 - 707	2
d	708 - 711	
r	712 - 715	

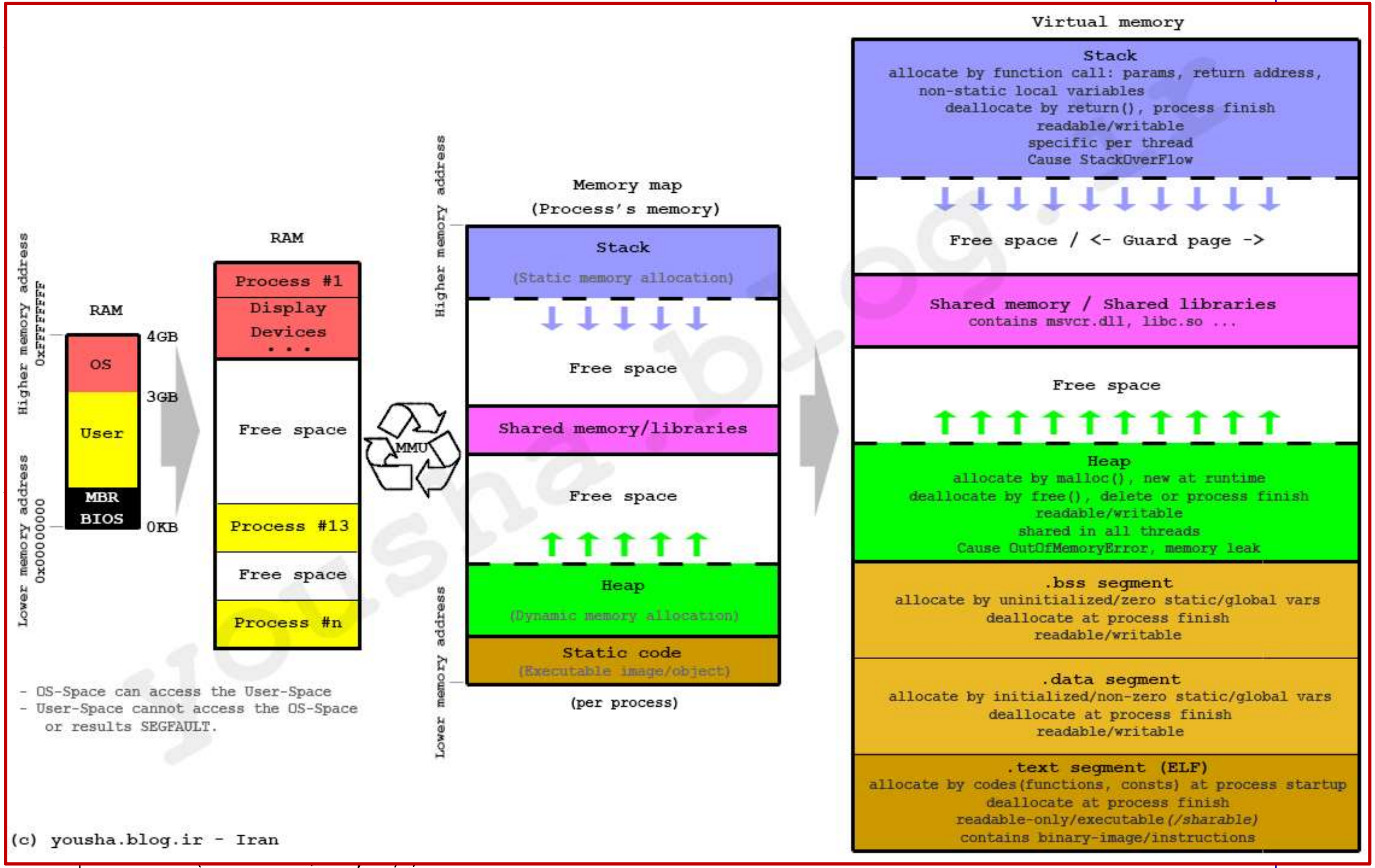
function call (values filled)

```
division ( 9, 2, 708, 712 )
```

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

i

{



}


i
{



}
```



Higher memory address
0xFFFFFFFF
Lower memory address
0x00000000
RAM
4GB
OS
3GB
User
MBR
BIOS
0KB
RAM
Process #1
Display
Devices
. . .
Free space
Process #13
Free space
Process #n
MMU
- OS-Space can access the User-Space
- User-Space cannot access the OS-Space
or results SEGFAULT.
Higher memory address
Lower memory address
Memory map
(Process's memory)
Stack
(Static memory allocation)
Free space
Shared memory/libraries
Free space
Heap
(Dynamic memory allocation)
Static code
(Executable image/object)
(per process)
Virtual memory
Stack
allocate by function call: params, return address,
non-static local variables
deallocate by return(), process finish
readable/writable
specific per thread
Cause StackOverFlow
Free space / <- Guard page ->
Shared memory / Shared libraries
contains msvcr.dll, libc.so ...
Free space
Heap
allocate by malloc(), new at runtime
deallocate by free(), delete or process finish
readable/writable
shared in all threads
Cause OutOfMemoryError, memory leak
.bss segment
allocate by uninitialized/zero static/global vars
deallocate at process finish
readable/writable
.data segment
allocate by initialized/non-zero static/global vars
deallocate at process finish
readable/writable
.text segment (ELF)
allocate by codes(functions, consts) at process startup
deallocate at process finish
readable-only/executable(/sharable)
contains binary-image/instructions
(c) yousha.blog.ir - Iran

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int denominator, int *dividend, int *remainder)
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
```

output:

```
address store in dividend: 708
address stored in remainder: 712
```

abel	Address	Value
numerator	400 - 403	9
denominator	404 - 407	2
dividend	408 - 411	708
remainder	412 - 415	712
x	700 - 703	9
y	704 - 707	2
d	708 - 711	
r	712 - 715	

```
     x=9;
     y=2;
     printf("address of d: %u\n",&d
     printf("address of r: %u\n",&r
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d rem
     printf("x=%d\n",x);
}
```

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int denominator, int *dividend, int *remainder)
{
    printf("address stored in divide
    printf("address stored in remair
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d
     printf("address of r: %u\n",&r
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d rem
     printf("x=%d\n",x);
}
```

```
if (2 < 1)
    return (0);
```

Label	Address	Value
numerator	400 - 403	9
denominator	404 - 407	2
dividend	408 - 411	708
remainder	412 - 415	712
x	700 - 703	9
y	704 - 707	2
d	708 - 711	
r	712 - 715	

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int
{
    printf("address stored in d
    printf("address stored in re
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d
     printf("address of r: %u\n",&r
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d rem
     printf("x=%d\n",x);
}
```

put into the memory location *708* the result of 9 divided by 2

Label	Address	Value
numerator	400 - 403	9
denominator	404 - 407	2
dividend	408 - 411	708
remainder	412 - 415	712
x	700 - 703	9
y	704 - 707	2
d	708 - 711	4
r	712 - 715	

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int
{
    printf("address stored in d
    printf("address stored in re
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d
     printf("address of r: %u\n",&r
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d rem
     printf("x=%d\n",x);
}
```

put into the memory location *712*
the result of 9 modulo 2

Label	Address	Value
numerator	400 - 403	9
denominator	404 - 407	2
dividend	408 - 411	708
remainder	412 - 415	712
x	700 - 703	9
y	704 - 707	2
d	708 - 711	4
r	712 - 715	1

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);

                                                 tor;
                                                 ator;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d remainder\n",x,y,d,r);
     printf("x=%d\n",x);
}
```

STACK (division) memory is freed

output:

9/2 = 4 with 1 remainder

call frame (main)

Label	Address	Value
x	700 - 703	9
y	704 - 707	2
d	708 - 711	4
r	712 - 715	1

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int
{
    printf("address stored in dividend: %u\n",dividend);
    printf("address stored in remainder: %u\n",remainder);

                                              tor;
                                              ator;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d remainder\n",x,y,d,r);
     printf("x=%d\n",x);
}
```

STACK (division) memory is freed

output:

```
x = 9
```

call frame (main)

Label	Address	Value
x	700 - 703	9
y	704 - 707	2
d	708 - 711	4
r	712 - 715	1

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, i
{
    printf("address stored in
    printf("address stored in
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    numerator = 7;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d
     printf("address of r: %u\n",&r
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d rem
     printf("x=%d\n",x);
}
```

… **lets go back but with one line of code added:**

```
numerator = 7;
```

Label	Address	Value
numerator	400 - 403	9
denominator	404 - 407	2
dividend	408 - 411	708
remainder	412 - 415	712
x	700 - 703	9
y	704 - 707	2
d	708 - 711	4
r	712 - 715	1

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int division(int numerator, int
{
    printf("address stored in d
    printf("address stored in re
    if (denominator < 1)
        return(0);
    *dividend=numerator/denominator;
    *remainder=numerator%denominator;
    numerator = 7;
}


int main(int argc, char *argv[])
{
     int x,y,d,r;

     x=9;
     y=2;
     printf("address of d: %u\n",&d
     printf("address of r: %u\n",&r
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d rem
     printf("x=%d\n",x);
}
```

… **added a new line of code**

```
numerator = 7;
```

Label	Address	Value
numerator	400 - 403	7
denominator	404 - 407	2
dividend	408 - 411	708
remainder	412 - 415	712
x	700 - 703	9
y	704 - 707	2
d	708 - 711	4
r	712 - 715	1

POINTERS

Passing Values **TO** and **FROM** a Function

```
#include <stdio.h>

int division(int numerator, int
{
    printf("address stored in di
    printf("address stored in remainder: %u\n",remainder);

                                             tor;
                                            ator;
    numerator = 7;
}


int main(int argc, char *argv[])
{


     x=9;
     y=2;
     printf("address of d: %u\n",&d);
     printf("address of r: %u\n",&r);
     division(x,y,&d,&r);
     printf("%d/%d = %d with %d remainder\n",x,y,d,r);
     printf("x=%d\n",x);
}
```

NO CHANGE:

pass-by-value

output:

9/2 = 4 with 1 remainder

call frame (main)

Label	Address	Value
x	700 - 703	9
y	704 - 707	2
d	708 - 711	4
r	712 - 715	1

Pointers in Functions

END OF PART 1

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

pass-by-value
verus
pass-by-reference

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	1
b	404 - 407	2

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	1
b	404 - 407	2

OUTPUT:

```
before swaps a = 1
before swaps b = 2
```

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("b         aps: a = %d\n", a);
    printf("b         aps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

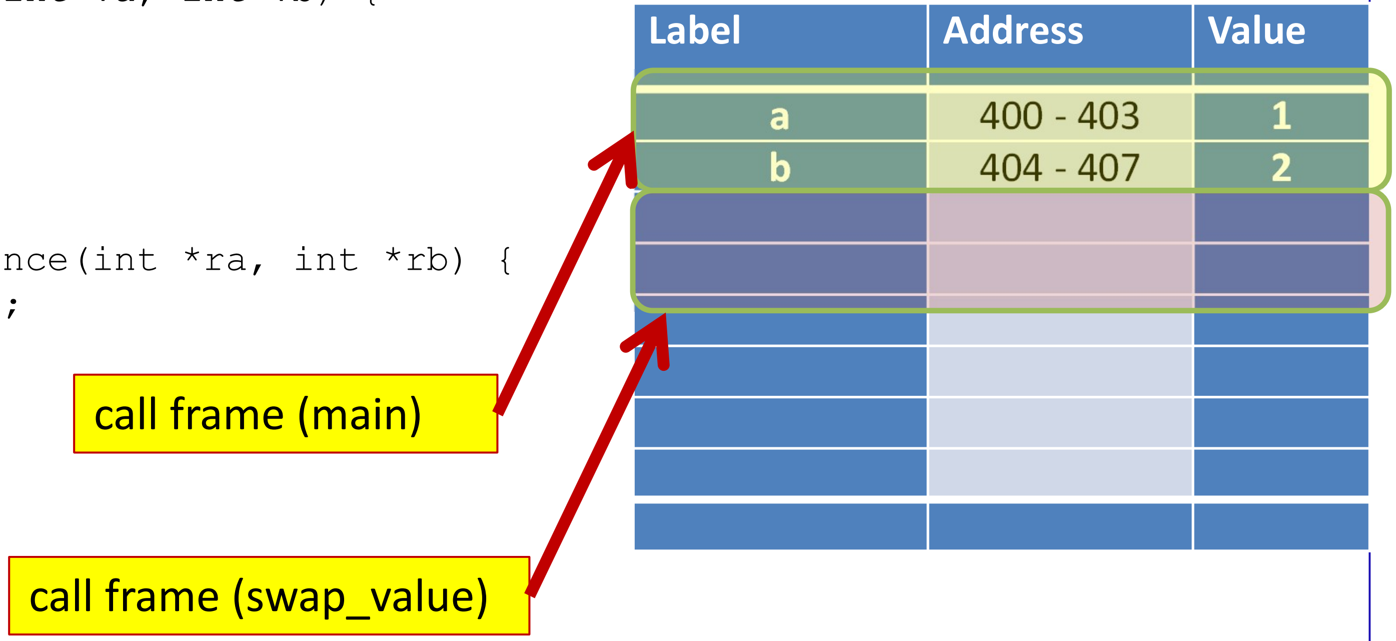
(1, 2)

Label	Address	Value
a	400 - 403	1
b	404 - 407	2

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```



Label	Address	Value
a	400 - 403	1
b	404 - 407	2

call frame (main)

call frame (swap_value)

POINTERS

(1, 2)

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb){
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

call frame (main)

Label	Address	Value
a	400 - 403	1
b	404 - 407	2
va	512 - 515	1
vb	516 -519	2

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	1
b	404 - 407	2
va	512 - 515	1
vb	516 -519	2
vTmp	520 - 523	1

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	1
b	404 - 407	2
va	512 - 515	2
vb	516 -519	2
vTmp	520 - 523	1

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	1
b	404 - 407	2
va	512 - 515	2
vb	516 -519	1
vTmp	520 - 523	1

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	1
b	404 - 407	2
va	512 - 515	2
vb	516 -519	1
vTmp	520 - 523	1

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	1
b	404 - 407	2

OUTPUT:

```
before swaps a = 1
before swaps b = 2
after swap_value: a = 1
after swap_value: b = 2
```

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("afte                b = %d\n", b);
```

(400, 404)

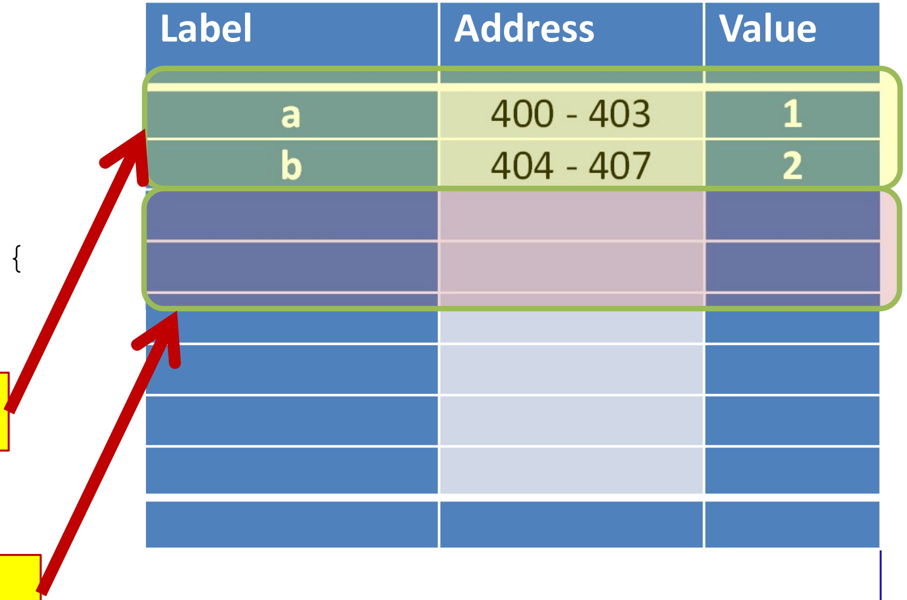
```
    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	1
b	404 - 407	2

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```



Label	Address	Value
a	400 - 403	1
b	404 - 407	2

call frame (main)

call frame (swap_reference)

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}
```

(400, 404)

```
void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

call frame (main)

Label	Address	Value
a	400 - 403	1
b	404 - 407	2
ra	512 - 515	400
rb	516 -519	404

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	1
b	404 - 407	2
ra	512 - 515	400
rb	516 -519	404
rTmp	520 - 523	1

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	2
b	404 - 407	2
ra	512 - 515	400
rb	516 -519	404
rTmp	520 - 523	1

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	2
b	404 - 407	1
ra	512 - 515	400
rb	516 -519	404
rTmp	520 - 523	1

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	2
b	404 - 407	1
ra	512 - 515	400
rb	516 -519	404
rTmp	520 - 523	1

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	2
b	404 - 407	1

OUTPUT:

```
before swaps a = 1
before swaps b = 2
after swap_value: a = 1
after swap_value: b = 2
after swap_reference:  a = 2
after swap_reference : b = 1
```

POINTERS

Passing Values TO and FROM a Function

```
void swap_value(int va, int vb) {
  int vTmp = va;
  va = vb;
  vb = vTmp;
}

void swap_reference(int *ra, int *rb) {
  int rTmp = *ra;
  *ra = *rb;
  *rb = rTmp;
}

int main()
{
    int a = 1;
    int b = 2;
    printf("before swaps: a = %d\n", a);
    printf("before swaps: b = %d\n", b);
    swap_value(a, b);
    printf("after swap_value: a = %d\n", a);
    printf("after swap_value: b = %d\n", b);

    swap_reference(&a, &b);
    printf("after swap_reference: a = %d\n", a);
    printf("after swap_reference: b = %d\n", b);
}
```

Label	Address	Value
a	400 - 403	2
b	404 - 407	1

OUTPUT:

```
before swaps a = 1
before swaps b = 1
after swap_value: a = 1
after swap_value: b = 2
after swap_reference:  a = 2
after swap_reference : b = 1
```

Pointers in Functions

END OF PART 2

CS 2211
Systems Programming

Part Four – A [part 2]:

Function Memory

Pointers and Arrays

beginning OF PART 1

POINTERS

Label:

what we have been calling label
- is really the **variable** label

Address Label:

there is also a label for the address

-> SO: we can pass the label (name for) the address we are using

- arrays are actually pointers

```
double dbray[5];   /*  5 x 8 bytes   */
```

Label	Address	Value
dbray[0]	400 - 407	
dbray[1]	408 – 415	
dbray[2]	416 - 423	
dbray[3]	424 - 431	
dbray[4]	432 - 439	

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5 x 8 bytes  */
```

Label:

- is really the **variable** label

Address Label:

- there is also a label for the address

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	
	dbray[1]	408 – 415	
	dbray[2]	416 - 423	
	dbray[3]	424 - 431	
	dbray[4]	432 - 439	

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5 x 8 bytes  */
double *d_ptr;      /* 4 bytes */
double value;       /* 8 bytes */
```

Label:

- is really the **variable** label

Address Label:

- there is also a label for the address

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	
	dbray[1]	408 – 415	
	dbray[2]	416 - 423	
	dbray[3]	424 - 431	
	dbray[4]	432 - 439	
	d_ptr	440 - 443	
	value	444 - 451	

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5 x 8 bytes  */
double *d_ptr;      /* 4 bytes */
double value;       /* 8 bytes */
int i;            /* 4 bytes */
int offset;       /* 4 bytes */
```

Label:

- is really the **variable** label

Address Label:

- there is also a label for the address

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	
	dbray[1]	408 – 415	
	dbray[2]	416 - 423	
	dbray[3]	424 - 431	
	dbray[4]	432 - 439	
	d_ptr	440 - 443	
	value	444 - 451	
	i	452 - 455	
	offset	456 - 459	

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5 x 8 bytes  */
double *d_ptr;      /* 4 bytes */
double value;       /* 8 bytes */
int i;            /* 4 bytes */
int offset;       /* 4 bytes */

 for (i=0; i < 5; i++)
     dbray[i] = (double) i + 10.0;
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	12.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	
	value	444 - 451	
	i	452 - 455	5
	offset	456 - 459	

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5 x 8 bytes  */
double *d_ptr;      /* 4 bytes */
double value;       /* 8 bytes */
int i;             /* 4 bytes */
int offset;        /* 4 bytes */

 for (i=0; i < 5; i++)
     dbray[i] = (double) i + 10.0;

 d_ptr = &(dbray[0]);
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	12.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	
	i	452 - 455	5
	offset	456 - 459	

POINTERS

- arrays are actually pointe

```
double dbray[5];    /*
double *d_ptr;      /*
double value;       /*
int i;             /* 4
int offset;        /* 4

 for (i=0; i < 5; i++)
     dbray[i] = (doubl

 d_ptr = &(dbray[0]);
```

INSTEAD OF

d_ptr = &(dbray[0]);

COULD HAVE BEEN

d_ptr = dbray;

THESE ARE THE SAME (both are address labels)
- computer sends the value (address)
referenced by the address label

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	12.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	
	i	452 - 455	5
	offset	456 - 459	

POINTERS

- arrays are actually pointer

```
double dbray[5];     /*
```

INSTEAD OF

d_ptr = &(dbray[0]);

WARNING:

difference between:

dbray and **&(dbray[0])**

- BOTH are FIXED i.e. 400 (you can not change this value)

d_ptr - you can change the value (address of what it references)

dress labels)

ue (address)

referenced by the address label

```
d_ptr = &(dbray[0]);
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	12.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	
	i	452 - 455	5
	offset	456 - 459	

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  
double *d_ptr;      /* 4
double value;       /* 8
int i;            /* 4 b
int offset;       /* 4 b

 for (i=0; i < 5; i++)
     dbray[i] = (double

 d_ptr = &(dbray[0]);
 value = 37;
 offset = 2;
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	12.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	37
	i	452 - 455	5
	offset	456 - 459	2

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*
double *d_ptr;      /* 4
double value;       /* 8
int i;            /* 4 b
int offset;       /* 4 b

 for (i=0; i < 5; i++)
     dbray[i] = (double

 d_ptr = &(dbray[0]);
 value = 37;
 offset = 2;

*(&(dbray[0])+offset) = value;   /*??*/
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	12.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	37
	i	452 - 455	5
	offset	456 - 459	2

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5
double *d_ptr;      /* 4
double value;       /* 8
int i;            /* 4 by
int offset;       /* 4 by

 for (i=0; i < 5; i++)
     dbray[i] = (double

 d_ptr = &(dbray[0]);
 value = 37;
 offset = 2;

*(&(dbray[0])+offset) = value;   /*??*/
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	37.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	37
	i	452 - 455	5
	offset	456 - 459	2

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5
double *d_ptr;      /* 4
double value;       /* 8
int i;            /* 4 by
int offset;       /* 4 by

 for (i=0; i < 5; i++)
     dbray[i] = (double

 d_ptr = &(dbray[0]);
 value = 37;
 offset = 2;

*(&(dbray[0])+offset) = value;
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	12.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	37
	i	452 - 455	5
	offset	456 - 459	2

```
decomposed:
    *(&(dbray[0])+2) = 37;
/*  &(dbray[0] is the address 400
     add two times the size of the variable type
     (double is 4 bytes so 4 x 2 = 16 bytes)
     add 16 to 400 to get the new address: 416)  */
```

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5
double *d_ptr;      /* 4
double value;       /* 8
int i;            /* 4 by
int offset;       /* 4 by

 for (i=0; i < 5; i++)
     dbray[i] = (double

 d_ptr = &(dbray[0]);
 value = 37;
 offset = 2;

*(&(dbray[0])+offset) = value;
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	37.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	37
	i	452 - 455	5
	offset	456 - 459	2

```
decomposed:
    *(&(dbray[0])+2) = 37;
/* assign the data (37) found at the variable named
   'value' to the memory location at the computed
    address (416)
   i.e. put the value 37 in the memory at 416 */
```

POINTERS

- arrays are actually pointe

```
double dbray[5];    /*
double *d_ptr;      /*
double value;       /*
int i;            /* 4
int offset;       /* 4

 for (i=0; i < 5; i++
     dbray[i] = (doub

 d_ptr = &(dbray[0]);
 value = 37;
 offset = 2;

*(&(dbray[0])+offset) = value;
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	37.0
dbray+3 &(dbray[3])	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	37
	i	452 - 455	5
	offset	456 - 459	2

```
remember:
   *blah simply reads as:
   - find the address stored in the variable (blah)
   - do something at that exact memory location
```

POINTERS

- arrays are actually pointe

```
double dbray[5];    /*
double *d_ptr;      /*
double value;       /*
int i;            /* 4
int offset;       /* 4

 for (i=0; i < 5; i++
     dbray[i] = (doub

 d_ptr = &(dbray[0]);
 value = 37;
 offset = 2;

*(&(dbray[0])+offset) = value;
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	37.0
dbray+3 &(dbray[3])	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	37
	i	452 - 455	5
	offset	456 - 459	2

```
remember:
   *d_ptr = 42.0;
      what is the result of this line of code?
```

POINTERS

- arrays are actually pointe

```
double dbray[5];    /*
double *d_ptr;      /*
double value;       /*
int i;            /* 4
int offset;       /* 4

 for (i=0; i < 5; i++
     dbray[i] = (doub

 d_ptr = &(dbray[0]);
 value = 37;
 offset = 2;

*(&(dbray[0])+offset) = value;
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	42.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	37.0
dbray+3 &(dbray[3])	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	37
	i	452 - 455	5
	offset	456 - 459	2

remember:

```
   *d_ptr = 42.0;
      what is the result of this line of code?
```

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5
double *d_ptr;      /* 4
double value;       /* 8
int i;            /* 4 by
int offset;       /* 4 by

 for (i=0; i < 5; i++)
     dbray[i] = (double

 d_ptr = &(dbray[0]);
 value = 37;
 offset = 2;

*(dbray + offset) = value;
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	37.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	37
	i	452 - 455	5
	offset	456 - 459	2

NOTE:

```
   *(dbray + offset) = 37;
/* this also works:
   dbray is the same as &(dbray[0])
   BUT - parentheses are required on LH of
   an expression */
```

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5
double *d_ptr;      /* 4
double value;       /* 8
int i;            /* 4 by
int offset;       /* 4 by

 for (i=0; i < 5; i++)
     dbray[i] = (double

 d_ptr = &(dbray[0]);
 value = 37;
 offset = 2;

*(dbray + offset) = value;

/* question: how would you print out
   these addresses ?
      *(dbray + offset)
      *(&(dbray[0]) + offset)
   to show they are the same address */
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	37.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	400
	value	444 - 451	37
	i	452 - 455	5
	offset	456 - 459	2

POINTERS

[side bar: scope of variables]

- arrays are actually pointers

```
double dbray[5];    /*   5 x 8 bytes   */
double *d_ptr;      /* 4 bytes */
double value;       /* 8 bytes */
/* int i;             /* 4 bytes */
int offset;       /* 4 bytes */

 for (int i=0; i < 5; i++)
     dbray[i] = (double) i + 10.0;

 printf("i=%d\n",i);
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	
	dbray[1]	408 – 415	
	dbray[2]	416 - 423	
	dbray[3]	424 - 431	
	dbray[4]	432 - 439	
	d_ptr	440 - 443	
	value	444 - 451	
	offset	452 - 455	

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5 x 8 bytes  */
double *d_ptr;      /* 4 bytes */
double value;       /* 8 bytes */
/* int i;            /* 4 bytes */
int offset;       /* 4 bytes */

 for (int i=0; i < 5; i++)
     dbray[i] = (double) i + 10.0;

 printf("i=%d\n",i);
```

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	12.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	
	value	444 - 451	
	offset	452 - 455	
	i	456 - 459	5

POINTERS

- arrays are actually pointers

```
double dbray[5];    /*  5 x 8 bytes  */
double *d_ptr;      /* 4 bytes */
double value;       /* 8 bytes */
/* int i;            /* 4 bytes */
int offset;       /* 4 bytes */

 for (int i=0; i < 5; i++)
     dbray[i] = (double) i + 10.0;

 printf("i=%d\n",i);
```

WARNING:

variable **i** is undefined:

error:

variable scope only within the FOR loop

Address Label	Label	Address	Value
dbray &(dbray[0])	dbray[0]	400 - 407	10.0
	dbray[1]	408 – 415	11.0
	dbray[2]	416 - 423	12.0
	dbray[3]	424 - 431	13.0
	dbray[4]	432 - 439	14.0
	d_ptr	440 - 443	
	value	444 - 451	
	offset	452 - 455	

Pointers and Arrays

END OF PART 1

Pointers and Dynamic Memory

Beginning OF PART 1

POINTERS

DYNAMIC MEMORY ALLOCATION

- **static memory allocation** (non-changing)
  - the size (in bytes) is known BEFORE a program starts to execute
  - when the program is loaded into memory, allocation of declared variables is performed

- sometimes a program does not know exactly how much memory it may need

i.e. reading a line of text – could be a character array of any size
- always declaring a humongous array very wasteful

- so: use **dynamic memory allocation**

  ask the O/S to set aside **x** amount of memory during execution

POINTERS

```
double *a;      /* a pointer variable */
```

Label	Address	Value
a	400 - 403	

POINTERS

```
double *a;      /* a pointer variable */
a = ( double *) malloc (40);
```

requesting O/S to set aside 40 bytes configured to handle values of type **double** and assign the address of this memory block in the pointer variable **a**

NOTE: **{DM}** – is not a label

- just something to put in temporary symbol for **allocated dynamic memory** (reserved memory for use later….)

Label	Address	Value
a	400 - 403	**10000**
{DM}	10000 - 10039	

POINTERS

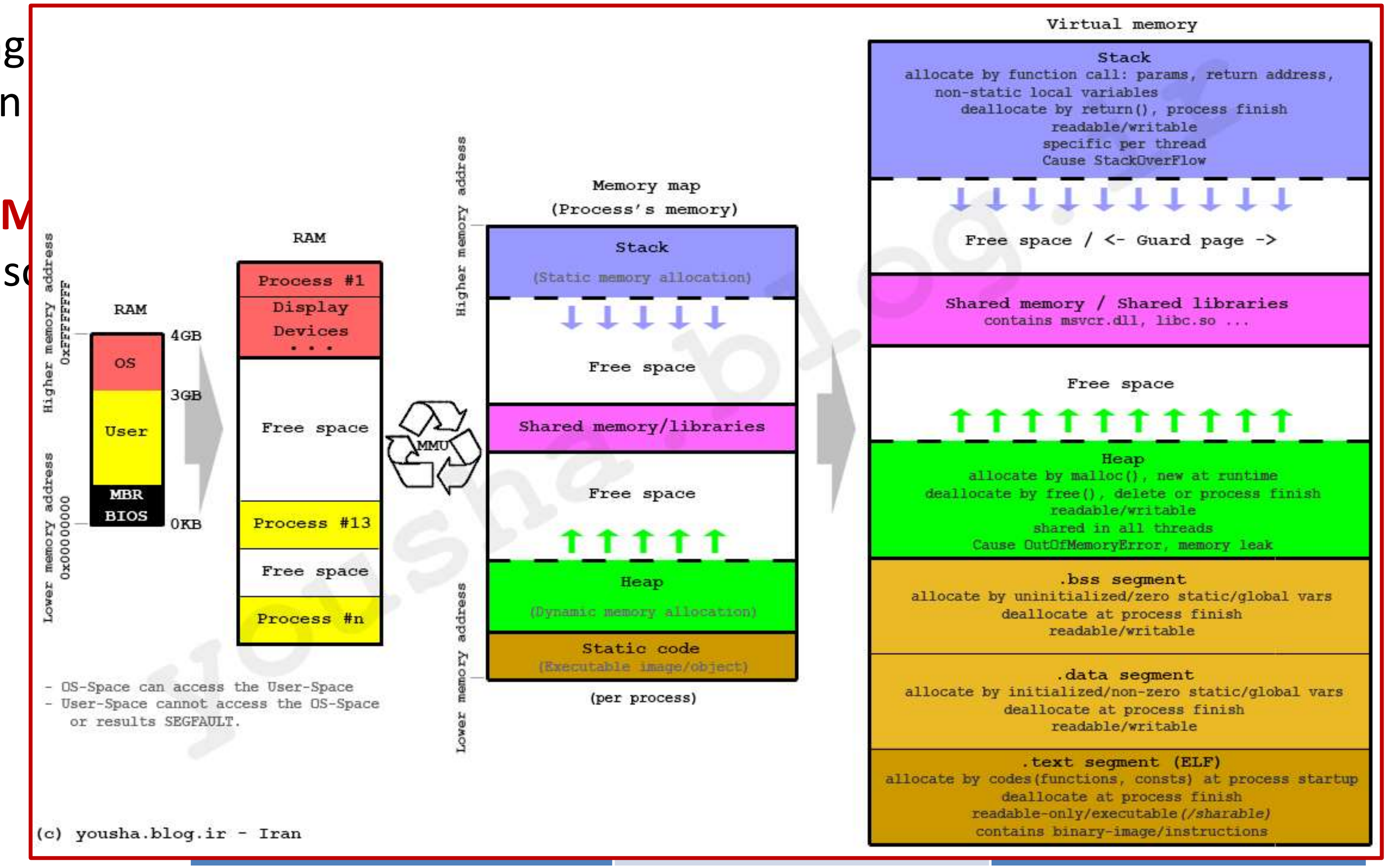
```
double *a;      /* a pointer variable */
a = ( double *) malloc (40);
```

requesting

and assign

NOTE: {DM

- just s



Higher memory address
0xFFFFFFFF
Lower memory address
0x00000000
RAM
OS
User
MBR
BIOS
4GB
3GB
0KB
RAM
Process #1
Display
Devices
. . .
Free space
Process #13
Free space
Process #n
MMU
Higher memory address
Lower memory address
Memory map
(Process's memory)
Stack
(Static memory allocation)
Free space
Shared memory/libraries
Free space
Heap
(Dynamic memory allocation)
Static code
(Executable image/object)
(per process)
Virtual memory
Stack
allocate by function call: params, return address,
non-static local variables
deallocate by return(), process finish
readable/writable
specific per thread
Cause StackOverFlow
Free space / <- Guard page ->
Shared memory / Shared libraries
contains msvcr.dll, libc.so ...
Free space
Heap
allocate by malloc(), new at runtime
deallocate by free(), delete or process finish
readable/writable
shared in all threads
Cause OutOfMemoryError, memory leak
.bss segment
allocate by uninitialized/zero static/global vars
deallocate at process finish
readable/writable
.data segment
allocate by initialized/non-zero static/global vars
deallocate at process finish
readable/writable
.text segment (ELF)
allocate by codes(functions, consts) at process startup
deallocate at process finish
readable-only/executable(/sharable)
contains binary-image/instructions
- OS-Space can access the User-Space
- User-Space cannot access the OS-Space
or results SEGFAULT.
yousha.blog

(c) yousha.blog.ir - Iran

POINTERS

```
double *a;      /* a pointer variable */
a = ( double *) malloc (40);
```

```
a[0] = 8;
```

Label	Address	Value
a	400 - 403	10000
*(a+0) a[0]	10000 - 10007	8
{DM}	10008 - 10039	

POINTERS

```
double *a;      /* a pointer variable */
a = ( double *) malloc (40);

a[0] = 8;
```

note:
it is common to mix and to use both **array** and/or **pointer designation** for dynamically allocated memory

Label	Address	Value
a	400 - 403	10000
*(a+0) a[0]	10000 - 10007	8
{DM}	10008 - 10039	

POINTERS

```
double *a;       /* a pointer variable */
a = ( double *) malloc (40);

a[0] = 8;
*(a+2) = 3;
```

Label	Address	Value
a	400 - 403	10000
*(a+0) a[0]	10000 - 10007	8
{DM}	10008 - 10015	
*(a+2) a[2]	10016 - 10023	3
{DM}	10024 - 10039	

POINTERS

```
double *a;       /* a pointer variable */
a = ( double *) malloc (40);

a[0] = 8;
*(a+2) = 3;
a[3] = 9;
```

Label	Address	Value
a	400 - 403	10000
*(a+0) a[0]	10000 - 10007	8
{DM}	10008 - 10015	
*(a+2) a[2]	10016 - 10023	3
*(a+3) a[3]	10024 - 10031	9
{DM}	10032 - 10039	

POINTERS

```
double *a;      /* a pointer variable */
a = ( double *) malloc (40);

/* can just assume the malloc(40) call set aside
   an area in the heap to accommodate 5 double variables
   (40 bytes is:
      8 bytes (size of a double variable) x 5 )
   so, the virtual labels immediately available for use )
```

Label	Address	Value
a	400 - 403	10000
*(a+0) a[0]	10000 - 10007	
*(a+1) a[1]	10008 - 10015	
*(a+2) a[2]	10016 - 10023	
*(a+3) a[3]	10024 - 10031	
*(a+4) a[4]	10032 - 10039	

POINTERS

```
double *a;      /* a pointer variable */
a = ( double *) malloc (40);
```

- **static memory** is deallocated at the end of the function (including MAIN – which is just another function)

- **dynamic memory** -> not so much…

Label	Address	Value
a	400 - 403	10000
{DM}	10000 - 10039	

POINTERS

```
double *a;      /* a pointer variable */
a = ( double *) malloc (40);
```

if **dynamic memory** is declared in a function and that function ends,
{DM} is still allocated and set aside - O\S can **NOT** use it again.

if not released, it can lead to the dreaded "**MEMORY LEAK**"
in C not uncommon to run out of memory because of
memory leaks

Label	Address	Value
a	400 - 403	10000
{DM}	10000 - 10039	

POINTERS

```
double *a;       /* a pointer variable */
a = ( double *) malloc (40);
```

dynamic memory MUST release memory when done by calling free()

Label	Address	Value
a	400 - 403	10000
{DM}	10000 - 10039	

POINTERS

```
double *a;      /* a pointer variable */
a = ( double *) malloc (40);
...
...
free(a);
```

NOTICE: ONLY dynamic memory is released
  - the pointer variable **a** is still in memory until end of function.

Label	Address	Value
a	400 - 403	

POINTERS

```
double *a;      /* a pointer variable */
a = ( double *) malloc (40);
...
...
return (0);
```

- when function ends [even main()] and **a** is released
  the block of memory has set aside by malloc(40) remains.
  BUT !!! pointer to it (**a**) is GONE

- no way to access this block {DM} of memory - **BAD - BAD**

Label	Address	Value
{DM}	10000 - 10039	

POINTERS

```
int *x;      /* a pointer variable */
x = (int *) malloc (70);
```

…. a second cause for memory leaks

Label	Address	Value
x	600 - 603	11000
{DM}	11000 - 11069	

POINTERS

```
int *x;      /* a pointer variable */
x = (int *) malloc (70);
...
...
x = (int *) malloc (300);
```

.... a second call to malloc() assigned to x

Label	Address	Value
x	600 - 603	20300
{DM}	11000 - 11069	
{DM}	20300 - 20599	

POINTERS

```
int *x;      /* a pointer variable */
x = (int *) malloc (70);
...
...
x = (int *) malloc (300);
free (x);
```

…. free(x) only frees where x is pointing to currently

Label	Address	Value
x	600 - 603	
{DM}	11000 - 11069	

POINTERS

```
int *x;      /* a pointer variable */
x = (int *) malloc (70);
...
...
x = (int *) malloc (300);
free (x);
```

the block **11000** to **11069** can **not** be access

result:
can not be used (blocked off)
  - and can **not** be access

.... free(x) only frees where x is pointing to currently

Label	Address	Value
x	600 - 603	
{DM}	11000 - 11069	

POINTERS

DYNAMIC MEMORY ALLOCATION

variations on the malloc() function

notice **malloc()** requests exact number of bytes
- programmer must know how many based on how to be used

calloc() - sets aside cells (memory) and initializes all to zero

```
double *a;
```

```
a = (double *) calloc (70, 8);
```

70 x 8 bytes
70 double variables
560 byes

```
// could have used
a = (double *) calloc ( 70, sizeof(double) );
```

- same thing -> sizeof(?) returns size of variable type
  based on O/S.

POINTERS

```
double *a;      /* a pointer variable */
a = (double *) calloc (70, sizeof(double) );

/* size of double: 8 bytes
   70 x 8 bytes (70 double variables)
   results in a request for 560 bytes */
```

Label	Address	Value
a	400 - 403	10000
{DM}	10000 - 10559	

Pointers and Dynamic Memory

END OF PART 1

Double Pointers

Beginning OF PART 1

POINTERS (double pointers)

```
int i;          /* an integer variable */
int *ptr1_i;    /* a pointer variable that points to i */
int **ptr2_i;   /* a pointer variable that points to i */
int x;          /* another integer variable */
```

double pointers:

indirection (pointer) to a variable containing an address (pointer)to an actual value

(instead of a pointer to a variable address that contains the value)

(i.e. double pointer is simply a pointer to a pointer that points to a value.

Label	Address	Value
i	400 - 403	
ptr1_i	404 - 407	
ptr2_i	408 - 411	
x	412 - 415	

POINTERS (double pointers)

```
int i;          /* an integer variable */
int *ptr1_i;    /* a pointer variable that points to i */
int **ptr2_i;   /* a pointer variable that points to i */
int x;          /* another integer variable */

i = 37;
```

Label	Address	Value
i	400 - 403	37
ptr1_i	404 - 407	
ptr2_i	408 - 411	
x	412 - 415	

POINTERS (double pointers)

```
int i;          /* an integer variable */
int *ptr1_i;    /* a pointer variable that points to i */
int **ptr2_i;   /* a pointer variable that points to i */
int x;          /* another integer variable */

i = 37;
ptr1_i = &i;
```

Label	Address	Value
i	400 - 403	37
ptr1_i	404 - 407	400
ptr2_i	408 - 411	
x	412 - 415	

POINTERS (double pointers)

```
int i;         /* an integer variable */
int *ptr1_i;   /* a pointer variable that points to i */
int **ptr2_i;  /* a pointer variable that points to i */
int x;         /* another integer variable */

i = 37;
ptr1_i = &i;
ptr2_i = &ptr1_i;
```

Label	Address	Value
i	400 - 403	37
ptr1_i	404 - 407	400
ptr2_i	408 - 411	404
x	412 - 415	

POINTERS (double pointers)

```
int i;         /* an integer variable */
int *ptr1_i;   /* a pointer variable that points to i */
int **ptr2_i;  /* a pointer variable that points to i */
int x;         /* another integer variable */

i = 37;
ptr1_i = &i;
ptr2_i = &ptr1_i;
x = **ptr2_i;
```

Label	Address	Value
i	400 - 403	37
ptr1_i	404 - 407	400
ptr2_i	408 - 411	404
x	412 - 415	37

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int twoDimArray(int **passedArray)
{
    printf("address of passedArray[0]: %u\n", & passedArray[0]);
    printf("address of passedArray[1]: %u\n", & passedArray[1]);
    passedArray [0][1] = 56;
    *(*(passedArray+1)+0) = -31;    /* same as passedArray [1][0] */
}


int main(int argc, char *argv[])
{
    int **m;                   /* 4 bytes (just an address) */
    m = (int **) calloc (2, sizeof(int *) ); /* 2 x 4 bytes */
    m[0] = (int *) calloc ( 3, sizeof(int)); /* 3 x 4 bytes */
    m[1] = (int *) calloc ( 2, sizeof(int)); /* 2 x 4 bytes */
    printf("address of m[0]: %u\n", &m[0]);
    printf("address of m[1]: %u\n", &m[1]);

    twoDimArray(m);

    printf("value of m[0][1]: %d \n",m[0][1]);
    printf("value of m[1][0]: %d \n",m[1][0]);

}
```

POINTERS (double pointers)

```
int **m;                          /* 4 bytes (just an address) */
                /* create space in the stack to hold an address
                   value and label that address variable as m */
```

Label	Address	Value
m	400 - 404	

POINTERS (double pointers)

```
int **m;                          /* 4 bytes (just an address) */
m = (int **) calloc (2, sizeof(int *) ); /* 2 x 4 bytes */
       /* create a space in the heap (dynamic memory) that
          can hold two addresses and assign the memory
          location to the variable labelled m */
```

Label	Address	Value
m	400 - 404	10100
{DM}	10100 - 10107	

POINTERS (double pointers)

```
int **m;                          /* 4 bytes (just an address) */
m = (int **) calloc (2, sizeof(int *) ); /* 2 x 4 bytes */
```

just enough space in memory has been allocated (set aside) to hold two address values that will have the labels we can us of m[0] and m[1] that will eventually point to two other address variables. [two different ways to visualize the same thing]

Label	Address	Value
m	400 - 404	10100
{DM}	10100 - 10107	

Label	Address	Value
m	400 - 404	10100
*(m+0) m[0]	10100 - 10103	
*(m+1) m[1]	10104 - 10107	

POINTERS (double pointers)

m has the address of **10100** (points to that location in memory.
-the place that **m** points to has enough space allocated to hold two address values
4 bytes x 2 = 8 bytes allocated).

so, by using indirection:
if **m** points to 10100 then **m + 0** (m + (0 x size of each unit), which in this case is 4 bytes)
10100 + (0 bytes) points to the location of 10100
and
m + 1 (m + (1 x size of each unit), which in this case is 4 bytes)
10100 + (4 bytes) points to the location of 10104

Label	Address	Value
m	400 - 404	10100
{DM}	10100 - 10107	

m	400 - 404	10100
*(m+0) m[0]	10100 - 10103	
*(m+1) m[1]	10104 - 10107	

POINTERS (double pointers)

```
int **m;                                /* 4 bytes (just an address) */
m = (int **) calloc (2, sizeof(int *) ); /* 2 x 4 bytes */
m[0] = (int *) calloc ( 3, sizeof(int)); /* 3 x 4 bytes */
 /* create a new space in the heap (dynamic memory) that
           can hold three interger values and assign the
           memory location to the variable labelled m[0] */
```

Label	Address	Value
m	400 - 404	10100
{DM}	10100 - 10107	
{DM}	10108 - 10119	

Label	Address	Value
m	400 - 404	10100
*(m+0) m[0]	10100 - 10103	10108
*(m+1) m[1]	10104 - 10107	
((m+0)+0) m[0][0]	10108 - 10111	
((m+0)+1) m[0][1]	10112 - 10115	
((m+0)+2) m[0][2]	10116 - 10119	

POINTERS (double pointers)

```
int **m;                              /* 4 bytes (just an address) */
m = (int **) calloc (2, sizeof(int *) ); /* 2 x 4 bytes */
m[0] = (int *) calloc ( 3, sizeof(int)); /* 3 x 4 bytes */
m[1] = (int *) calloc ( 2, sizeof(int)); /* 2 x 4 bytes */
 /* create a new space in the heap (dynamic memory) that
          can hold two interger values and assign the
          memory location to the variable labelled m[1] */
```

Label	Address	Value
m	400 - 404	10100
{DM}	10100 - 10107	
{DM}	10108 - 10119	
{DM}	10120 - 10127	

Label	Address	Value
m	400 - 404	10100
*(m+0) m[0]	10100 - 10103	10108
*(m+1) m[1]	10104 - 10107	10120
((m+0)+0) m[0][0]	10108 - 10111	
((m+0)+1) m[0][1]	10112 - 10115	
((m+0)+2) m[0][2]	10116 - 10119	
((m+1)+0) m[1][0]	10120 - 10123	
((m+1)+1) m[1][1]	10124 - 10127	

POINTERS (double pointers)

```
int **m;                              /* 4 bytes (just an address) */
m = (int **) calloc (2, sizeof(int *) ); /* 2 x 4 bytes */
m[0] = (int *) calloc ( 3, sizeof(int)); /* 3 x 4 bytes */
m[1] = (int *) calloc ( 2, sizeof(int)); /* 2 x 4 bytes */
m[0][1] = 56;
```

Label	Address	Value
m	400 - 404	10100
{DM}	10100 - 10107	
{DM}	10108 - 10119	
{DM}	10120 - 10127	

Label	Address	Value
m	400 - 404	10100
*(m+0) m[0]	10100 - 10103	10108
*(m+1) m[1]	10104 - 10107	10120
((m+0)+0) m[0][0]	10108 - 10111	
((m+0)+1) m[0][1]	10112 - 10115	56
((m+0)+2) m[0][2]	10116 - 10119	
((m+1)+0) m[1][0]	10120 - 10123	
((m+1)+1) m[1][1]	10124 - 10127	

POINTERS (double pointers)

```
int **m;                     /* 4 bytes (just an address) */
m = (int **) calloc (2, sizeof(int *) ); /* 2 x 4 bytes */
m[0] = (int *) calloc ( 3, sizeof(int)); /* 3 x 4 bytes */
m[1] = (int *) calloc ( 2, sizeof(int)); /* 2 x 4 bytes */
m[0][1] = 56;
*(*(m+1)+0) = -31;      /* same as m[1][0] – either okay */
```

Label	Address	Value
m	400 - 404	10100
{DM}	10100 - 10107	
{DM}	10108 - 10119	
{DM}	10120 - 10127	

Label	Address	Value
m	400 - 404	10100
*(m+0) m[0]	10100 - 10103	10108
*(m+1) m[1]	10104 - 10107	10120
((m+0)+0) m[0][0]	10108 - 10111	
((m+0)+1) m[0][1]	10112 - 10115	56
((m+0)+2) m[0][2]	10116 - 10119	
((m+1)+0) m[1][0]	10120 - 10123	-31
((m+1)+1) m[1][1]	10124 - 10127	

POINTERS

Passing Values TO and FROM a Function

```
#include <stdio.h>

int twoDimArray(int **passedArray)
{
    printf("address of passedArray[0]: %u\n", & passedArray[0]);
    printf("address of passedArray[1]: %u\n", & passedArray[1]);
    passedArray [0][1] = 56;
    *(*(passedArray+1)+0) = -31;    /* same as passedArray [1][0] */
}


int main(int argc, char *argv[])
{
    int **m;                   /* 4 bytes (just an address) */
    m = (int **) calloc (2, sizeof(int *) ); /* 2 x 4 bytes */
    m[0] = (int *) calloc ( 3, sizeof(int)); /* 3 x 4 bytes */
    m[1] = (int *) calloc ( 2, sizeof(int)); /* 2 x 4 bytes */
    printf("address of m[0]: %u\n", &m[0]);
    printf("address of m[1]: %u\n", &m[1]);

    twoDimArray(m);

    printf("value of m[0][1]: %d \n",m[0][1]);
    printf("value of m[1][0]: %d \n",m[1][0]);

}
```

POINTERS

Passing Values TO and FROM a Function

```c
#include <stdio.h>

int twoDimArray(int **passedArray)
{
    printf("address of passedArray[0]: %u\n", & passedArray[0]);
    printf("address of passedArray[1]: %u\n", & passedArray[1]);
    passedArray [0][1] = 56;
    *(*(passedArray+1)+0) = -31;    /* same as passedArray [1][0] */
}

int main(int argc, char *argv[])
{
    int **m;                   /* 4 bytes (just an address) */
    m = (int **) calloc (2, sizeof(int *) ); /* 2 x 4 bytes */
    m[0] = (int *) calloc ( 3, sizeof(int)); /* 3 x 4 bytes */
    m[1] = (int *) calloc ( 2, sizeof(int)); /* 2 x 4 bytes */
    printf("address of m[0]: %u\n", &m[0]);
    printf("address of m[1]: %u\n", &m[1]);

    twoDimArray(m);

    printf("value of m[0][1]: %d \n",m[0][1]);
    printf("value of m[1][0]: %d \n",m[1][0]);
    free(m[0]); /* every call to malloc or calloc MUST be free'd */
    free(m[1]);
    free(m);
}
```

Double Pointers

END OF PART 1