

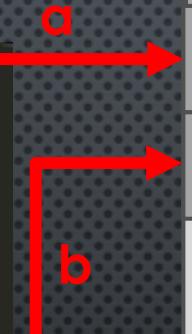
# INTRODUCTION TO POINTERS

Memory representations are not entirely accurate, they are simplified  
for understanding purposes about working, not architecture

# HOW MEMORY WORKS ?

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main(){
5     int a = 0;
6     int b = 3;
7
8     printf("%x", &b);
9 }
```



Process Memory

0xff00fa03	Start of stack frame
0xff00fa02	0
0xff00fa00	3
0xff000000	End of stack frame

Output :

```
$sh ./a.out
ff00fa00
```

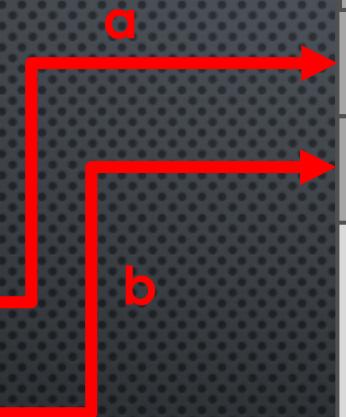
# WHAT IS A POINTER ?

- A POINTER IS A VARIABLE : HAS AN ADDRESS
- A POINTER CONTAINS THE ADDRESS OF ANOTHER VARIABLE
- THE ADDRESS CONTAINED COMES FROM THE CONTEXT WHERE THE POINTER IS INITIALIZED

# HOW POINTERS WORKS ?

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main(){
5
6     // variable
7     int a = 0; ----->
8
9     // pointer to the variable
10    int* b = &a; ----->
11
12    printf("%d", *b);
13 }
```



Process Memory

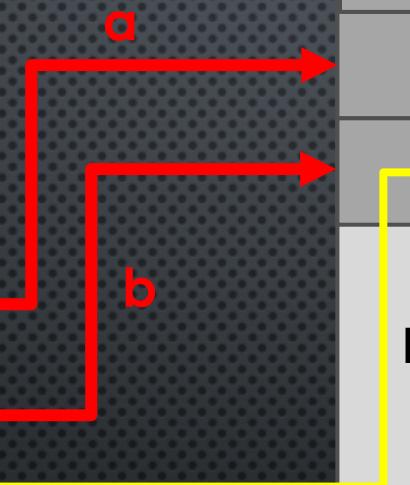
0xff00fa03	Start of stack frame
0xff00fa02	0
0xff00fa00	ff00fa02
0xff000000	End of stack frame

# HOW POINTERS WORKS ?

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main(){
5
6     // variable
7     int a = 0; -----|
8
9     // pointer to the variable
10    int* b = &a; -----|
11
12    printf("%d", *b); -----|
13 }
```

Process Memory	
0xff00fa03	Start of stack frame
0xff00fa02	0
0xff00fa00	ff00fa02
b	
0xff000000	End of stack frame



# HOW POINTERS WORKS ?

Code :

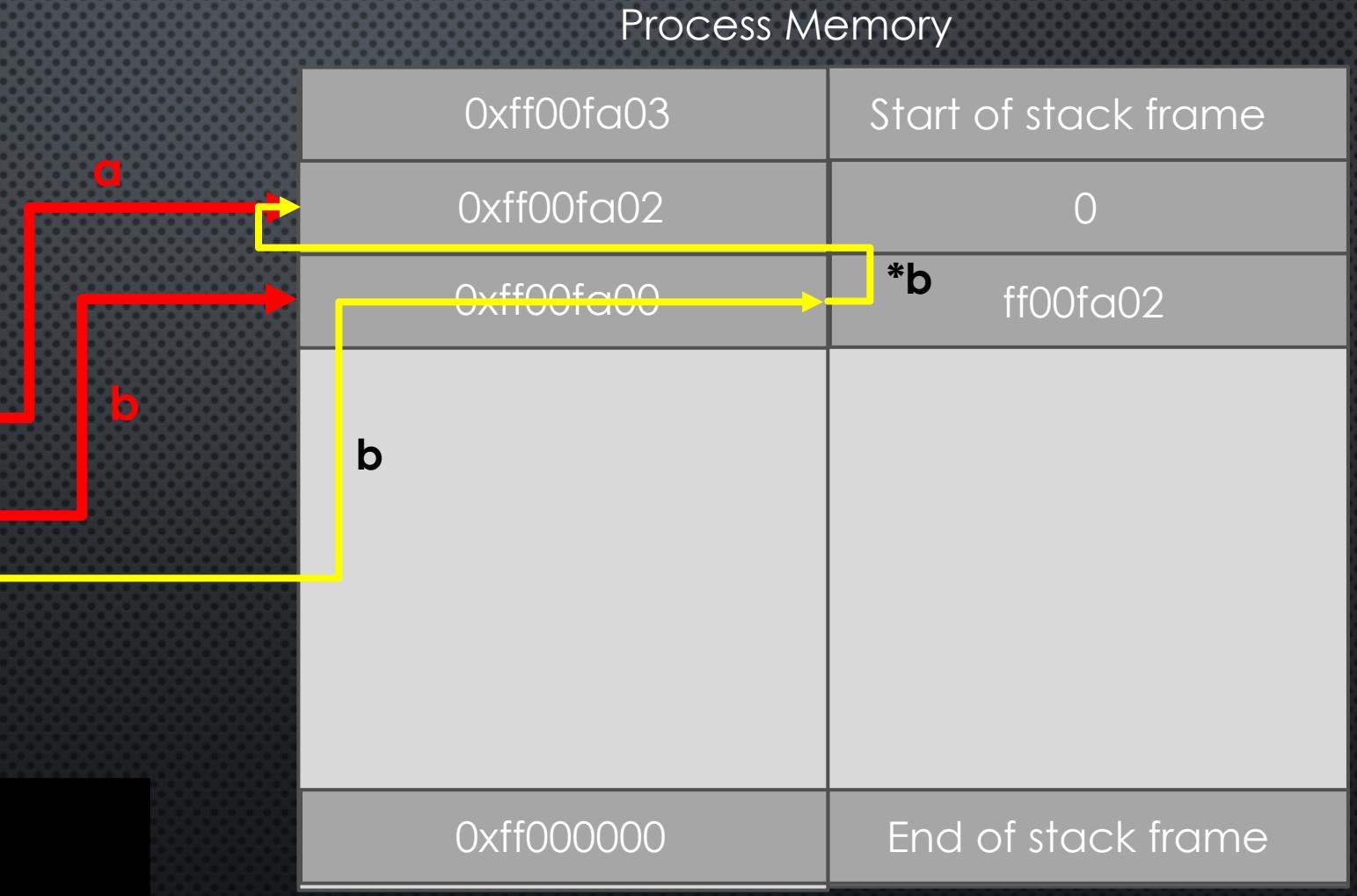
```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main(){
5
6     // variable
7     int a = 0; -----
8
9     // pointer to the variable
10    int* b = &a; -----
11
12    printf("%d", *b); -----
13 }
```



# HOW POINTERS WORKS ?

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main(){
5
6     // variable
7     int a = 0; -----
8
9     // pointer to the variable
10    int* b = &a; -----
11
12    printf("%d", *b); -----
13 }
```



Output :

```
$sh ./a.out
0
```

# WHAT IS A POINTER ?

- A POINTER IS A VARIABLE : HAS AN ADDRESS
- A POINTER CONTAINS THE ADDRESS OF ANOTHER VARIABLE
- THE ADDRESS CONTAINED COMES FROM THE CONTEXT WHERE THE POINTER IS INITIALIZED

# WHAT FOR ?

- WHEN YOU CALL A FUNCTION : NEW STACKFRAME, CANNOT ACCESS VARIABLES FROM THE CALLER CONTEXT
- YOU PASS VARIABLES AS ARGUMENTS OF THE FUNCTION : SAME PROBLEM, BECAUSE YOU CREATE A COPY OF THE ORIGINAL VARIABLE IN THE NEW STACKFRAME
- SOLUTION : POINTER, CONTAINS ADDRESS OF THE VARIABLE IN THE CALLER'S STACKFRAME

# A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int pow(int a, int b);
5
6 int main()
7 {
8     int a = 4;
9     int b = 5;
10    int c = pow(a,b);
11    printf("%d", c);
12 }
13
14 int pow(int a, int b)
15 {
16     int i;
17     int c = 1;
18     for(i=0; i<b; i++)
19     {
20         c*=b;
21     }
22     return c;
23 }
24 }
```

Main  
Context

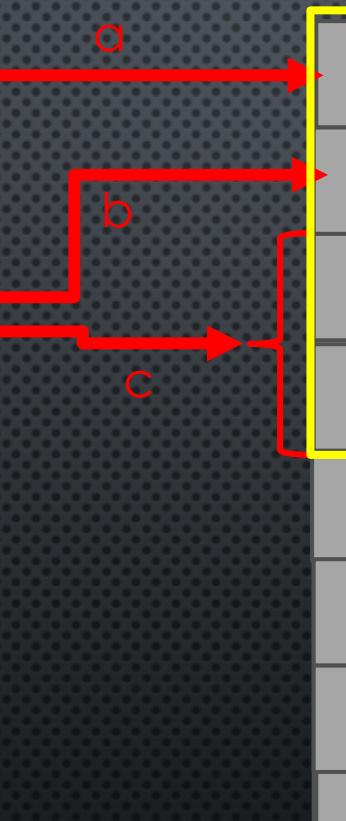
Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	NULL
0xff000016	Adress of pow function
0xff00000e	
0xff000006	
0xff000004	
0xff000002	
0xff000000	

# A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int pow(int a, int b);
5
6 int main()
7 {
8     int a = 4;
9     int b = 5;
10    int c = pow(a,b);
11    printf("%d", c);
12 }
13
14 int pow(int a, int b)
15 {
16     int i;
17     int c = 1;
18     for(i=0; i<b; i++)
19     {
20         c*=b;
21     }
22     return c;
23 }
24 }
```



Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	NULL
0xff000016	Adress of pow function
0xff00000e	
0xff000006	
0xff000004	
0xff000002	
0xff000000	

# A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int pow(int a, int b);
5
6 int main()
7 {
8     int a = 4;
9     int b = 5;
10    int c = pow(a,b);
11    printf("%d", c);
12 }
13
14 int pow(int a, int b)
15 {
16     int i;
17     int c = 1;
18     for(i=0; i<b; i++)
19     {
20         c*=b;
21     }
22     return c;
23 }
```

« Caller »  
Context

Pow  
Context

Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	NULL
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call Adress to return when pow end
0xff000006	
0xff000004	
0xff000002	
0xff000000	

# A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int pow(int a, int b);
5
6 int main()
7 {
8     int a = 4;
9     int b = 5;
10    int c = pow(a,b);
11    printf("%d", c);
12 }
13 Copy of a
14
15 int pow(int a, int b)
16 {
17     int i;
18     int c = 1;
19     for(i=0; i<b; i++)
20     {
21         c*=b;
22     }
23     return c;
24 }
```

Process Memory

0xff00001c	Main stack frame	4
0xff00001a		5
0xff000018		NULL
0xff000016	Adress of pow function	
0xff00000e	Pow stack frame	Adress of pow call
0xff000006		4
0xff000004		5
0xff000002		0
0xff000000	End of stack	1024

# A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int pow(int a, int b);
5
6 int main()
7 {
8     int a = 4;
9     int b = 5;
10    int c = pow(a,b);
11    printf("%d", c);
12 }
13
14 int pow(int a, int b)
15 {
16     int i;
17     int c = 1;
18     for(i=0; i<b; i++)
19     {
20         c*=b;
21     }
22     return c;
23 }
24 }
```

Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	1024
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call
0xff000006	4
0xff000004	5
0xff000002	0
0xff000000	End of stack 1024

# A LITTLE ABOUT STACKFRAME AND CONTEXT

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int pow(int a, int b);
5
6 int main()
7 {
8     int a = 4;
9     int b = 5;
10    int c = pow(a,b);
11    printf("%d", c);
12 }
13
14 int pow(int a, int b)
15 {
16     int i;
17     int c = 1;
18     for(i=0; i<b; i++)
19     {
20         c*=b;
21     }
22 }
23 return c;
24 }
```

Output :

```
$ sh ./a.out
1024
```

Process Memory

0xff00001c	Main stack frame	4
0xff00001a		5
0xff000018		1024
0xff000016	Adress of pow function	
0xff00000e	Pow stack frame	Adress of pow call
0xff000006		4
0xff000004		5
0xff000002		0
0xff000000	End of stack	1024

BUT WHAT ABOUT WITH POINTERS ?

# WITH POINTERS :

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int pow(int a, int b);
5
6 int main()
7 {
8     int a = 4;
9     int b = 5;
10    int* c = &a;
11    pow(d,b);
12    printf("%d", a);
13 }
14
15 void pow(int* a, int b)
16 {
17     int i;
18     int c = *a;
19     for(i=0; i<b; i++)
20     {
21         (*a)*=c
22     }
23 }
```

Main Context

Process Memory

0xff00001c	Main stack frame 4
0xff00001a	5
0xff000018	0xff00001c
0xff000016	Adress of pow function
0xff00000e	
0xff000006	
0xff000004	
0xff000002	
0xff000000	

# BEFORE THE FOR LOOP

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int pow(int a, int b);
5
6 int main()
7 {
8     int a = 4;
9     int b = 5;
10    int* c = &a;
11    pow(d,b);
12    printf("%d", a);
13 }
14
15 void pow(int* a, int b)
16 {
17     int i;
18     int c = *a;
19     for(i=0; i<b; i++)
20     {
21         (*a)*=c
22     }
23 }
```

« Caller »  
Context

Pow  
Context

Process Memory

0xff00001c	Main stack frame
0xff00001a	4
0xff000018	0xff00001c
0xff000016	Adress of pow function
0xff00000e	Pow stack frame
0xff000006	Adress of pow call
0xff000004	5
0xff000002	0
0xff000000	End of stack

# BEFORE THE FOR LOOP

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int pow(int a, int b);
5
6 int main()
7 {
8     int a = 4;
9     int b = 5;
10    int* c = &a;
11    pow(d,b);
12    printf("%d", a);
13 }
14
15 void pow(int* a, int b)
16 {
17     int i;
18     int c = *a;
19     for(i=0; i<b; i++)
20     {
21         (*a)*=c
22     }
23 }
```

« Caller »  
Context

Pow  
Context

Process Memory

0xff00001c	Main stack frame
0xff00001a	4
0xff000018	0xff00001c
0xff000016	Adress of pow function
0xff00000e	Pow stack frame
0xff00000e	Adress of pow call
0xff000006	0xff00001c
0xff000004	5
0xff000002	0
0xff000000	End of stack
0xff000000	4

# AFTER THE FOR LOOP

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int pow(int a, int b);
5
6 int main()
7 {
8     int a = 4;
9     int b = 5;
10    int* c = &a;
11    pow(d,b);
12    printf("%d", a);
13 }
14
15 void pow(int* a, int b)
16 {
17     int i;
18     int c = *a;
19     for(i=0; i<b; i++)
20     {
21         (*a)*=c
22     }
23 }
```

« Caller »  
Context

Pow  
Context

Process Memory

0xff00001c	Main stack frame 1024
0xff00001a	5
0xff000018	0xff00001c
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call
0xff000006	0xff00001c
0xff000004	5
0xff000002	0
0xff000000	End of stack 4

# AFTER THE FOR LOOP

Code :

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int pow(int a, int b);
5
6 int main()
7 {
8     int a = 4;
9     int b = 5;
10    int* c = &a;
11    pow(c,b);
12    printf("%d", a);
13 }
14
15 void pow(int* a, int b)
16 {
17     int i;
18     int c = *a;
19     for(i=0; i<b; i++)
20     {
21         (*a)*=c
22     }
23 }
```

Output :

```
$ sh ./a.out
1024
```

Process Memory

0xff00001c	Main stack frame 1024
0xff00001a	5
0xff000018	0xff00001c
0xff000016	Adress of pow function
0xff00000e	Pow stack frame Adress of pow call
0xff000006	0xff00001c
0xff000004	5
0xff000002	0
0xff000000	End of stack 4