CS F111: Computer Programming

(Second Semester 2021-22)

Lect 24: Pointers.

Nikumani Choudhury

Asst. Professor, Dept. of Computer Sc. & Information System

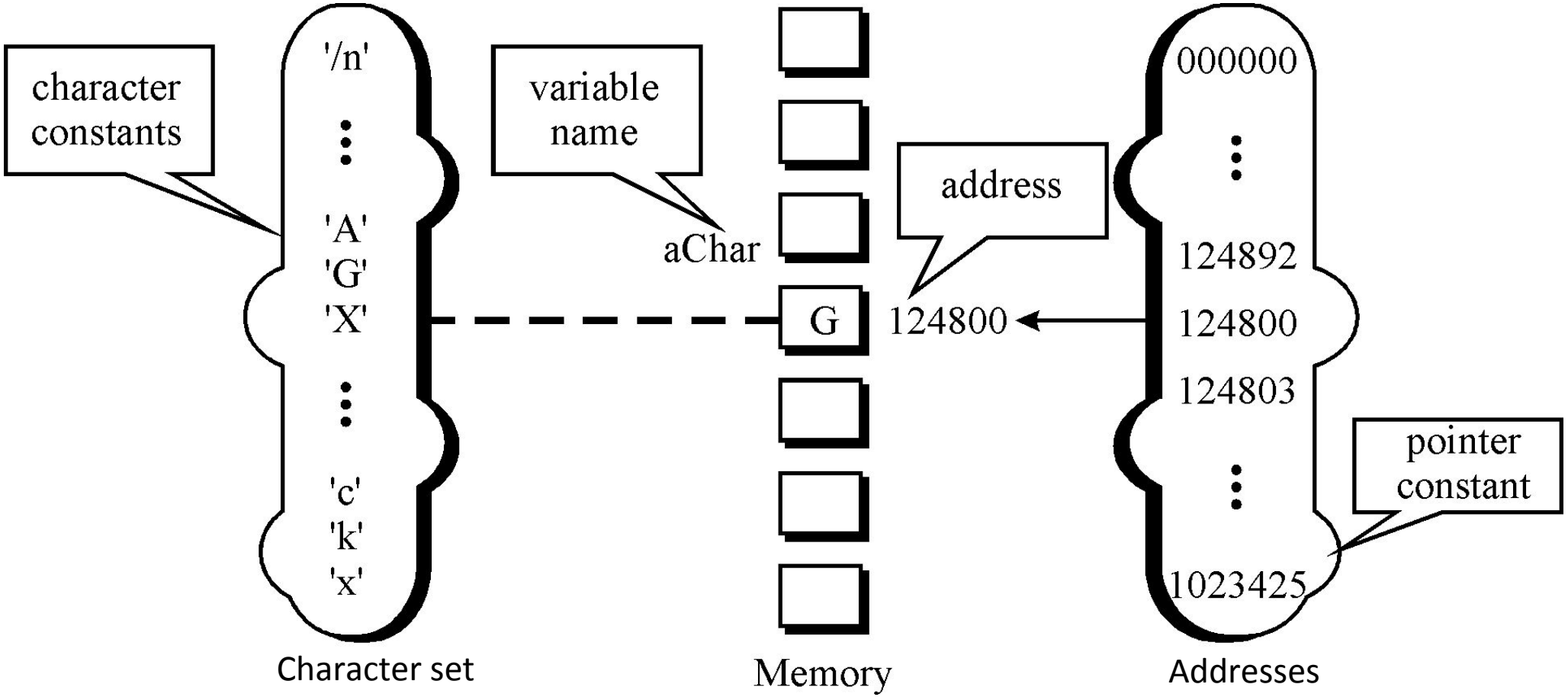
BITS Pilani

Hyderabad Campus

What are Pointers?

- A Pointer is a constant or a variable that contains an address that can be used to access the data stored at that address.
- A derived data type built from one of the fundamental data types.
- What are the applications of pointers?
  - Make a function return multiple values by passing multiple addresses as function arguments
  - Functions can be passed as arguments to other functions
  - Access or change the elements of arrays or structures without copying them inside a function
  - Increase the execution speed and hence complete the execution of the program in less time
  - Manipulate dynamic data structures like structures, linked lists, queues, stacks, trees etc.

Pointer Constant: An address in memory



Pointer Values: & Extracts the address

```
#include <stdio.h>
int main() {
  int a = 167;
  int b = 45;
  double c = 32.78;
  printf ("\n%d is stored at %p", a, &a);
  printf ("\n%d is stored at %p", b, &b);
  printf ("\n%f is stored at %p", c, &c);
  return 0;
}
```

```
167 is stored at 0x7fff9013bd60
45 is stored at 0x7fff9013bd64
32.780000 is stored at 0x7fff9013bd68
```

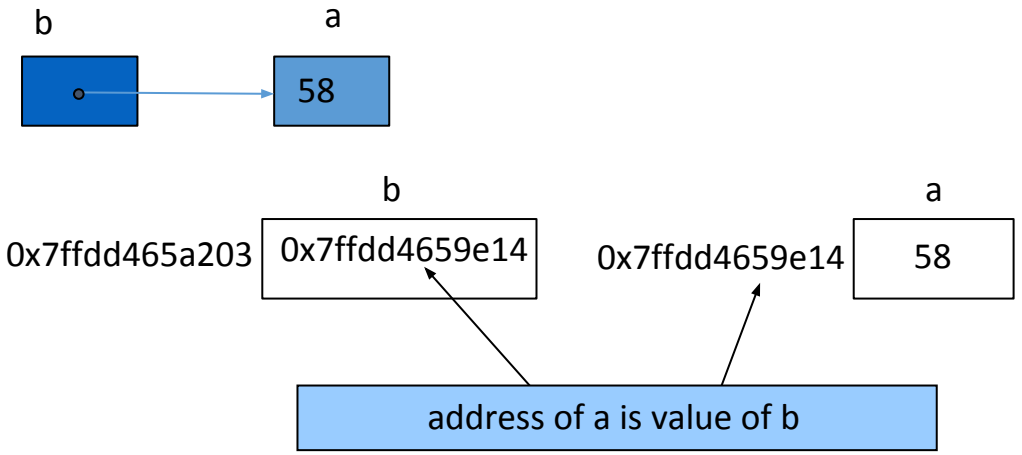
Pointer Operators

`&` (address operator):

- Returns the address of its operand
- Example

```
int a = 58;
int *b;
b = &a;     // b gets address of a
```

- `b` "points to" `a`



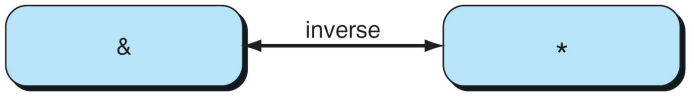
* (indirection/dereferencing operator):

- Returns the value of what it's operand points to
- *b returns a (because b points to a).
- * can be used to assign a value to a location in memory

  *b = 32; // changes a to 32
- Dereferenced pointer (operand of *) must be an lvalue (no constants)

* and & are inverses (cancel each other out)

*&b == b and &*b == b




```c
#include <stdio.h>
int main()
{
    int *p;
    int q = 10;
    p = &q;
    printf ("%p", &*p);
    printf ("\n%p", p);
    return 0;
}
```

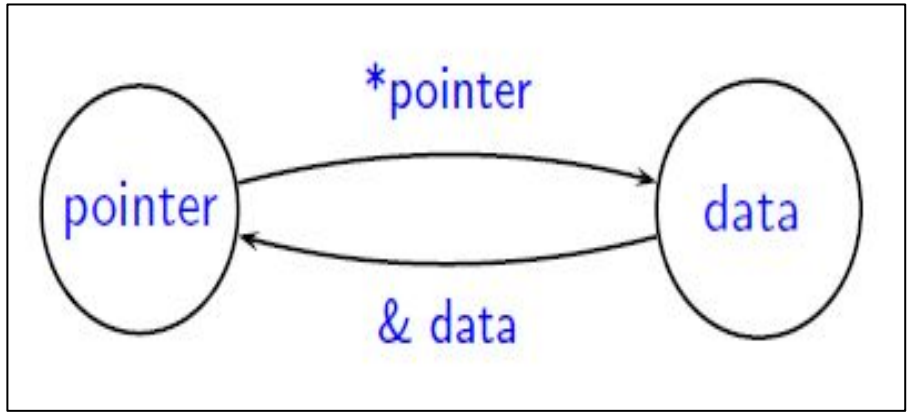
0x7ffd84b3b044
0x7ffd84b3b044

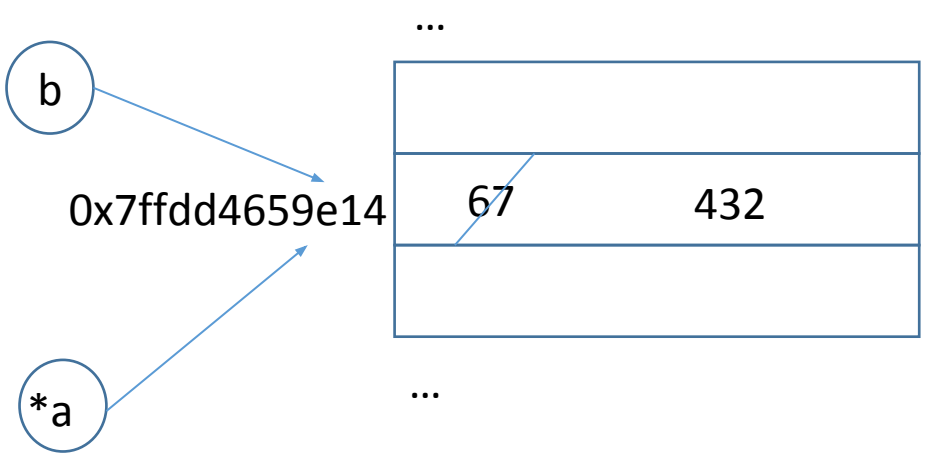
Pointer variables: Declaring, Initializing and Dereferencing

- datatype *pointername;
- int *p; float *q; char *r;
- int *p; int* p; int * p; (1st style is preferred)

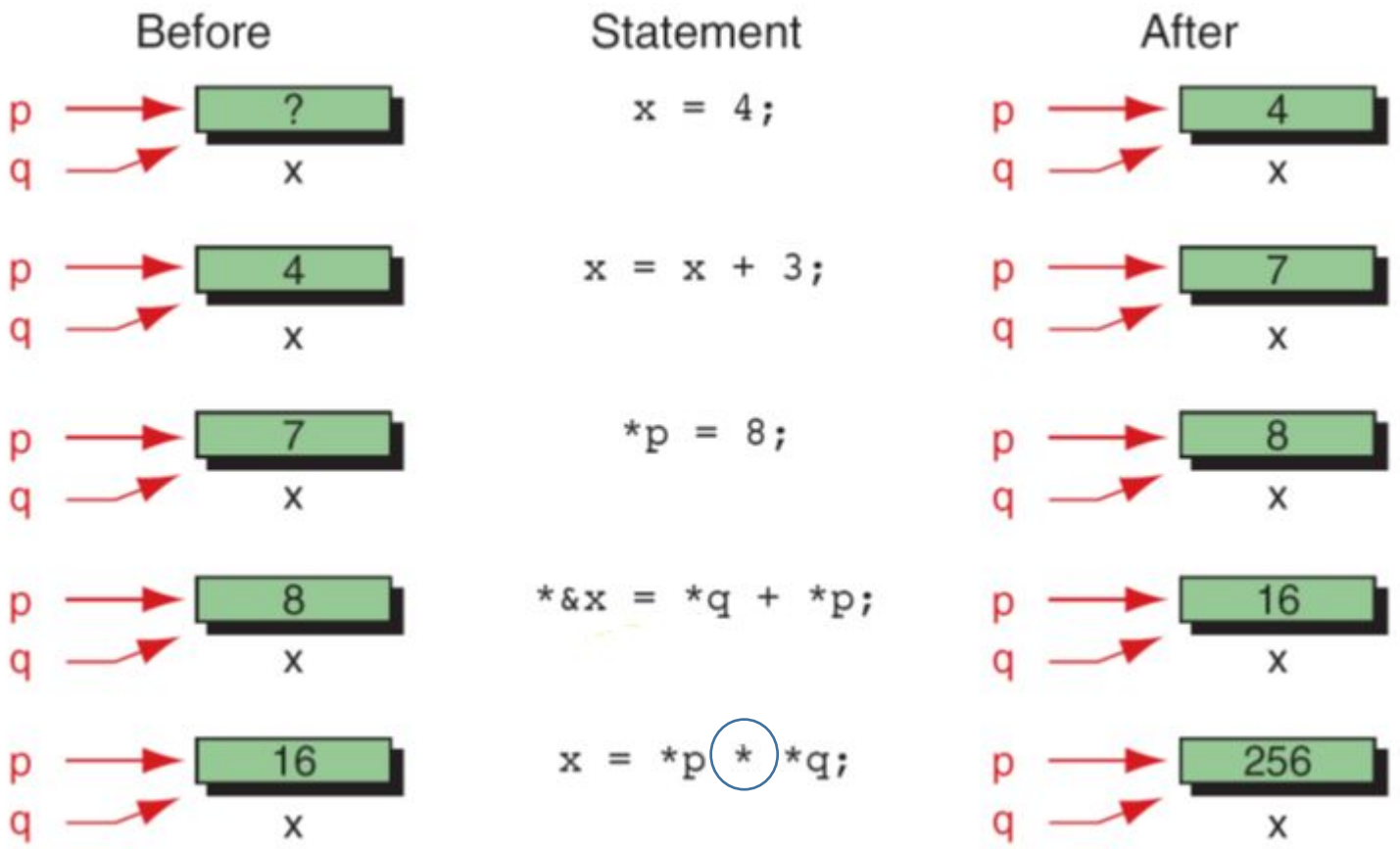
```
#include <stdio.h>
int main(){
   int *a;  int b = 67;
   a = &b;
   printf ("%d", b);
   printf ("\n%d is stored at %p", *a, a);
   *a = 432;
   printf ("\n%d", b);
   printf ("\n%d is stored at %p", *a, a);
   return 0;
}
```

```
67
67 is stored at 0x7ffdd4659e14
432
432 is stored at 0x7ffdd4659e14
```





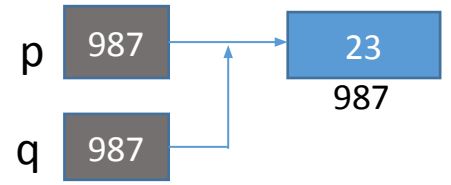
Accessing variables through pointers



More attributes:

1. A pointer variable can be assigned to another pointer of same type:

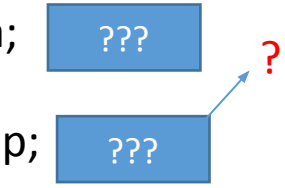
```
int a = 23;
int *p = &a;
int *q = p;
```



2. An uninitialized (or NULL) pointer must not be dereferenced.

Solution

```
int a;
int *p = &a;
*p=58;
int *p=NULL;
printf ("%d" ,*p);
```

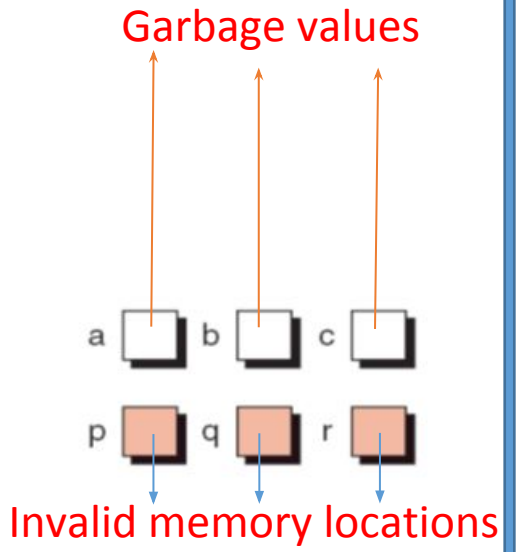


3. Adding 1 to a pointer does not necessarily add 1 to its' value.

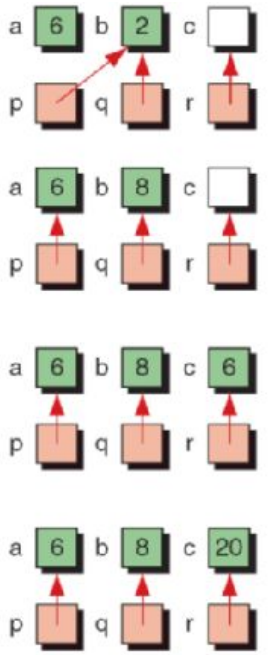
Fun with Pointers

- #include <stdio.h>

```
int main (void)
{
// Local Declarations
   int  a;
   int  b;
   int  c;
   int* p;
   int* q;
   int* r;
```



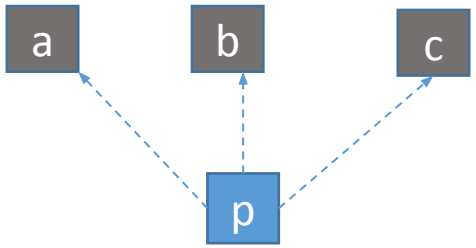

```
   q = p;
   r = &c;


   p  = &a;
   *q = 8;



   *r = *p;



   *r = a + *q + *&c;

   printf("%d %d %d \n",
            a, b, c);

    printf("%d %d %d",
            *p, *q, *r);
    return 0;
}  // main
```



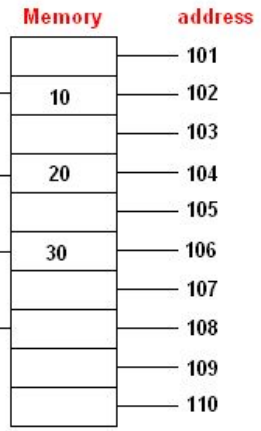

```
Results:
6 8 20
6 8 20
```

Pointer Flexibility

- We can make the same pointer pointing to different variables at different times.




```
#include <stdio.h>

int main()
{
    int a=10,b=20,c=30;
    int *ptr;
    ptr=&a;
    printf("%d\n",*ptr);
    ptr=&b;
    printf("%d\n",*ptr);
    ptr=&c;
    printf("%d",*ptr);
    return 0;
}
```

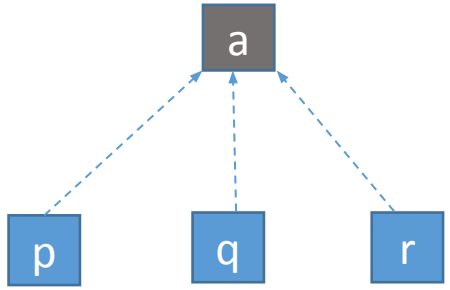


Output:

```
10
20
30
```

- We can make different pointers pointing to the same variable.

```
#include <stdio.h>
int main() {
   int a = 789;
   int *p = &a;
   int *q = &a;
   int *r = &a;

   printf ("%d", *p);
   printf ("\n%d", *q);
   printf ("\n%d", *r);
   return 0;
}
```



Output:

```
789
789
789
```

const int *ptr, int *const ptr, const int *const ptr

- You cannot change the value pointed out by ptr, but you can change the pointer itself.

```
#include <stdio.h>
#include<stdlib.h>
int main() {
int a = 86;
int b = 98;

const int *ptr = &a;
printf( "value pointed by ptr: %d\n", *ptr);

ptr = &b;
printf( "value pointed by ptr: %d\n", *ptr);
return 0;
}
```

int const *ptr = &a;

*ptr = b;

assignment of read-only location '*ptr'

```
value pointed out by ptr: 86
value pointed out by ptr: 98
```

You cannot change the pointer ptr, but can change the value pointed by ptr.

```
#include <stdio.h>
#include<stdlib.h>
int main() {
int a = 86;
int b = 98;
int *const ptr = &a;
printf( "value pointed by ptr:%d\n", *ptr);
printf( "Address ptr is pointing to:%p", ptr);

*ptr = b;
printf( "\nvalue pointed by ptr:%d\n", *ptr);
printf( "Address ptr is pointing to:%p", ptr);
return 0;
}
```

ptr=&b;

assignment of read-only variable 'ptr'

```
value pointed out by ptr:86
Address ptr is pointing to:0x7ffd9a2ee7a0
value pointed out by ptr:98
Address ptr is pointing to:0x7ffd9a2ee7a0
```

You can neither change the value pointed by ptr nor the pointer ptr.

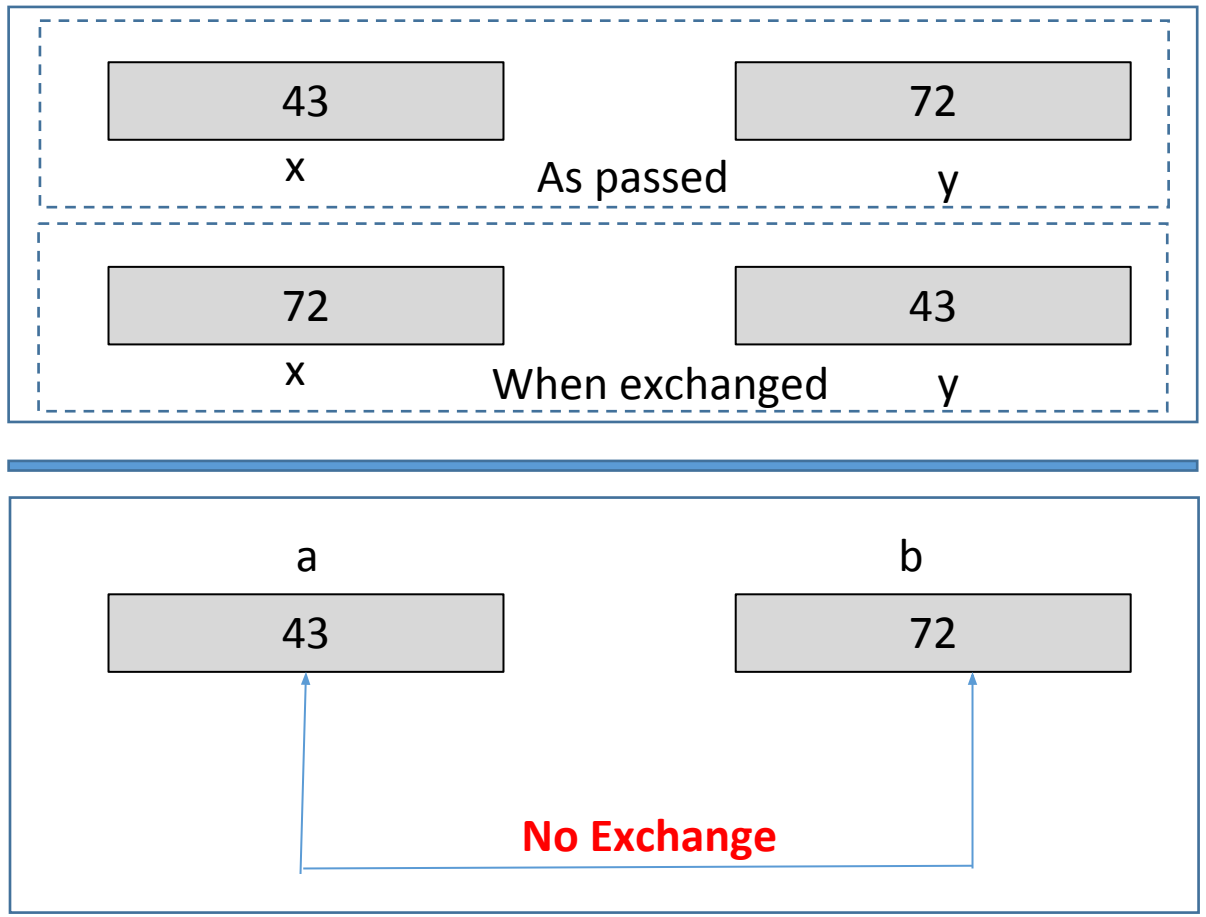
```
#include<stdio.h>
#include<stdlib.h>
int main() {
int a = 86;
int b = 98;

const int *const ptr = &a;
printf( "Value pointed by ptr: %d\n", *ptr);
printf( "Address ptr pointing to: %p", ptr);
ptr = &b;    Or
*ptr = b;
return 0;
}
```

assignment of read-only location '*ptr'

```
Value pointed by ptr: 86
Address ptr pointing to: 0x7fff473276d0
```

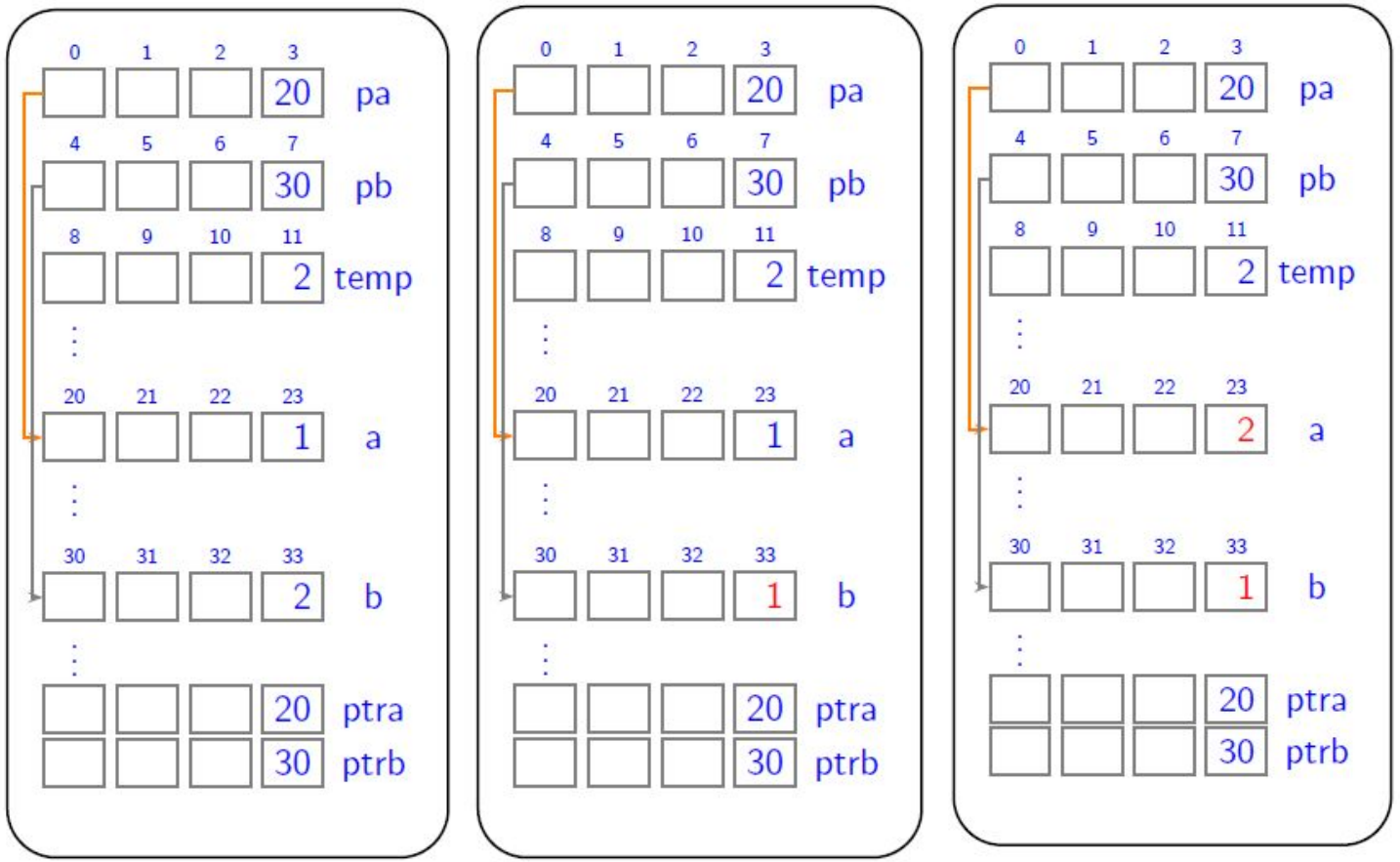
Pointers as Function Arguments: Why?

```
void swap (int x, int y) {
   int temp;
   temp = x;
   x = y;
   y = temp;
return;
}

int main () {
   int a = 43;
   int b = 72;
   swap (a, b);
   printf ("%d %d", a, b);
return 0;
}
```



Swap using Pointers: Right way

```
#include <stdio.h>
void swap(int *pa, int *pb) {
   int temp;
   temp = *pb;
   *pb = *pa;
   *pa = temp;
}
int main(){
int a = 1, b = 2;
int *ptra = &a;
int *ptrb = &b;
swap(ptra, ptrb);
printf ("a = %d  and b = %d", a, b);
return 0;
}
```



Functions returning Pointer variables

```
#include <stdio.h>
int* larger(int*, int*);
int main(){
  int a = 45;
  int b = 108;
  int *p;
  p = larger (&a, &b);
  printf("%d is larger",*p);
  return 0;
}
int* larger (int *px, int *py){
  if(*px > *py)
       return px;
  else
       return py;
}
```

```
108 is larger
```