Swap: Recap

```
void swap(int, int);
main( )
{
    int a, b;
    a = 10, b = 20;
    swap(a, b);
    printf("a = %d, b = %d\n", a, b);
}
void swap( int x, int y )
{
    int temp;
    temp = x;
    x = y;
    y = temp;
    return;
}
```

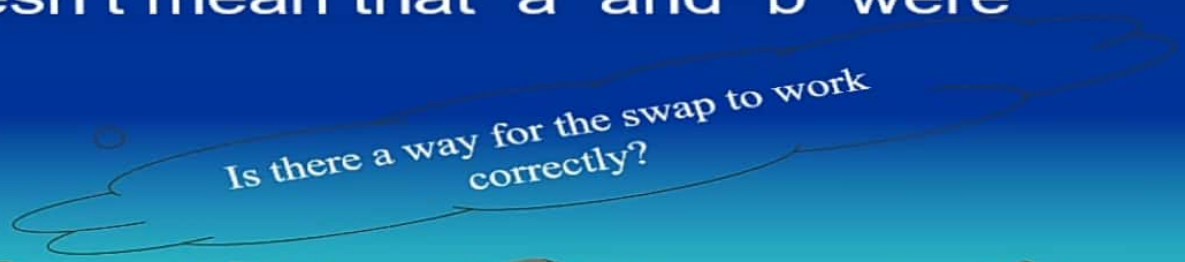
For some reason this does not work !

```
$./a.out
a = 10, b = 20
$
```

Swap: Recap

Why ***swap*** failed to swap!

- When the function is called the values 10 and 20 are passed.
- In swap( ), x = 10 and y = 20.
- swap( ) actually swapped x and y locally
- But this doesn't mean that a and b were swapped.



There was no change in the memory locations/place holders of main()

```
void swap(int, int);
 main( )
{
    int a, b;
    a = 10, b = 20;
    swap(a, b);
    printf("a = %d, b = %d\n", a, b);
}

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

Memory locations/place holders reserved by *main()*

Address		
&a	a	10
&b	b	20

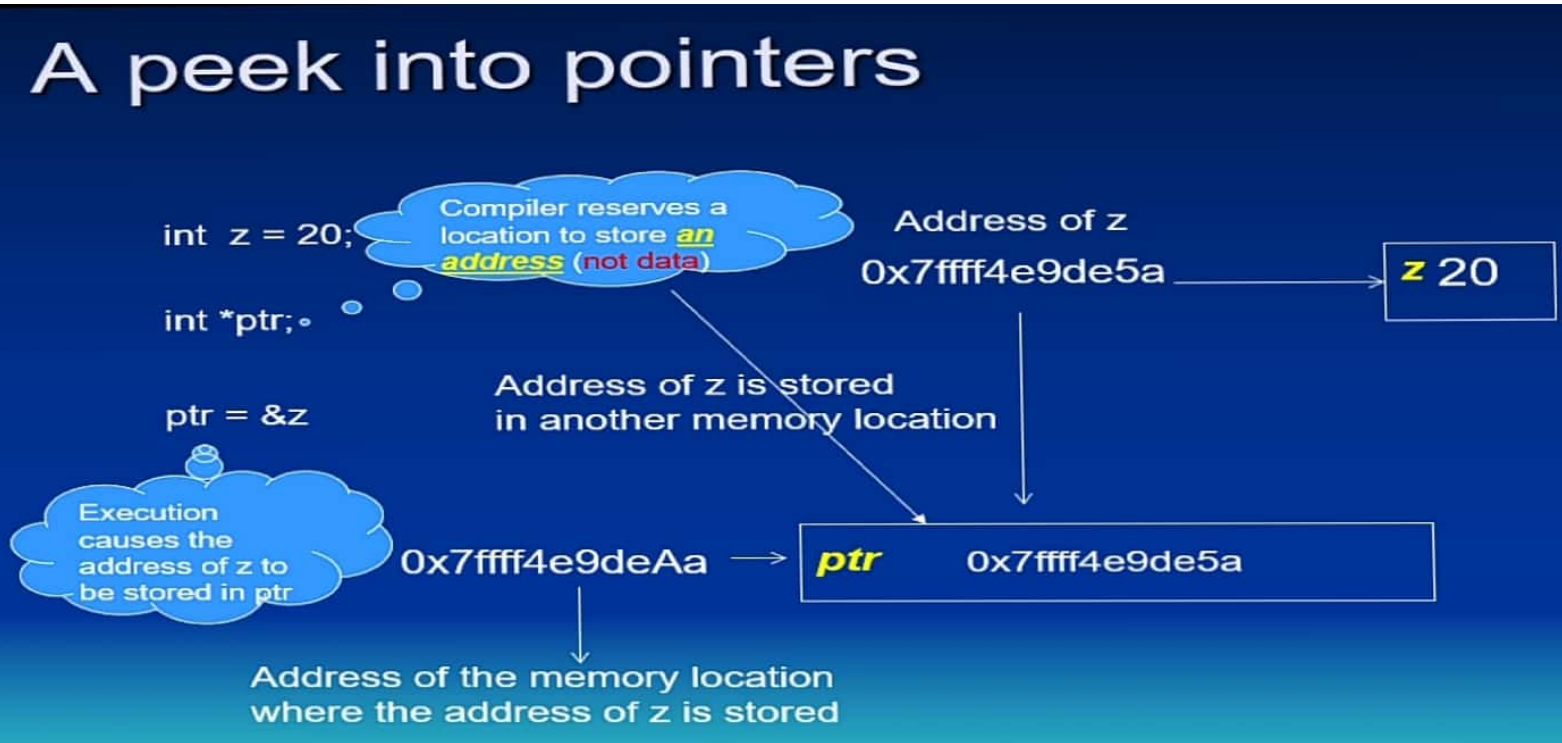
Memory locations/place holders reserved by *swap()*

Address		
&x	x	20
&y	y	10
&t	t	10

How can swap work correctly?

- There are two ways by which information is passed to a function:
  - **Passing values:** This is what our swap is doing
  - **Passing addresses:** In C, we need to program this by passing addresses.
- You now need to pass addresses of a and b viz. &a and &b

A peek into pointers



Passing an address

```
void func( int *a ); //function declaration
```

This declares *a* as a pointer variable which can store the address of some other variable of type int

This memory location can store the address of some other variable of type int

Remember that **a** is a pointer type variable, not a standard variable.

Pointers ...Swapping

```
#include <stdio.h>
void swap(int *a, int *b);
int main()
{
        int  a, b, *ap, *bp;

        a = 10, b = 20;
        ap = &a; bp = &b;

        swap(ap, bp);
        printf("a = %d, b = %d\n", a, b);

        return 0;
}
```

```
void swap(int *x, int *y)
{
        int t;

        t = *x;
        *x = *y;
        *y = t;

        return;
}
```

Passing an array to a function

```
include <stdio.h>
float average(float age[]);
int main()
    {
     float avg, age[]={23.4,55,22.6,3,40.5,18};
     int i; avg = average(age);
     printf("Average of ");
    for(i=0;i<=6;++i) printf("%1.2f +",age[i]);
    printf(" = %.2f\n", avg);
    return 0 ;      }
```

```
float average(float age[])
{
  int i;
  float avg, sum = 0.0;
  for (i = 0; i < 6; ++i)
  {
      sum = sum +age[i];
  }
  avg = (sum / 6);
  return avg;
}
```

```
// (const float *age)   (float *age)   (float age[6])  same
```

Tips and traps

- Omitting the return-type in a function definition causes a syntax error if the function prototype specifies a return type other than int.
- Forgetting to return a value from a function that is supposed to return a value can lead to unexpected errors.
- Returning a value from a function whose return type is void causes a syntax error.
- Even though an omitted return type defaults to int, always state the return type explicitly. The return type for main is however normally omitted.