Fundamentals of Programming

Lecture 11

Chamila Karunatilake

Department of Information and Communication Technology

Faculty of Technology

University of Sri Jayewardenepura

chamilakarunatilake@sjp.ac.lk

C Pointers

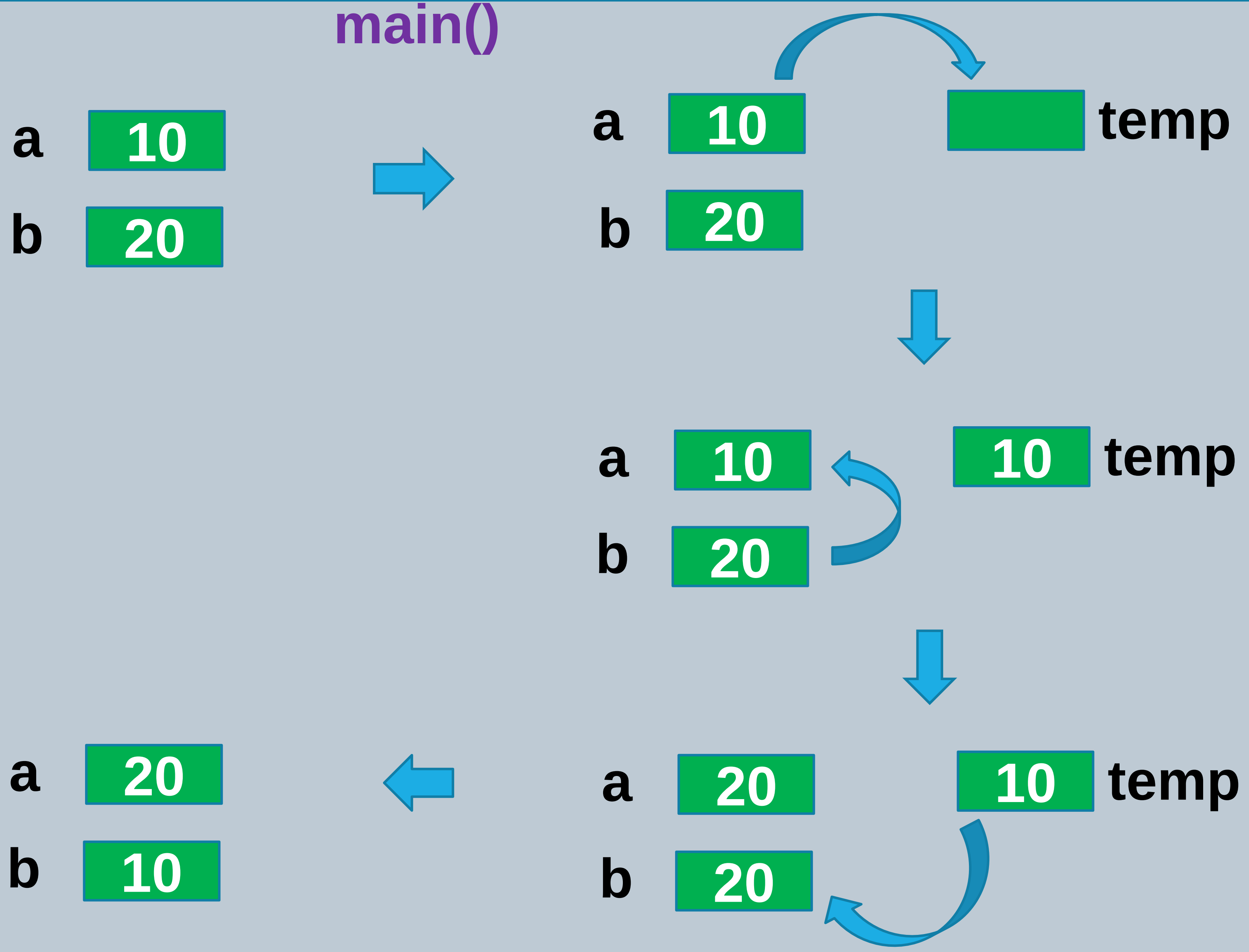
SWAP Program

- Swapping is the process of exchanging two items that replacing each other.

- When it is done to the variables, an extra variable is needed to holds one of the swapping variables while they are exchanging.

```c
#include <stdio.h>
int main()
{
    int a,b,temp;
    a = 10;
    b = 20;
    printf("Before Swap\n");
    printf(" a : %d\n b : %d\n ",a,b);
    temp = a;
    a = b;
    b = temp;
    printf("\nAfter Swap\n\n");
    printf(" a : %d\n b : %d\n ",a,b);
    return 0;
}
```

main()



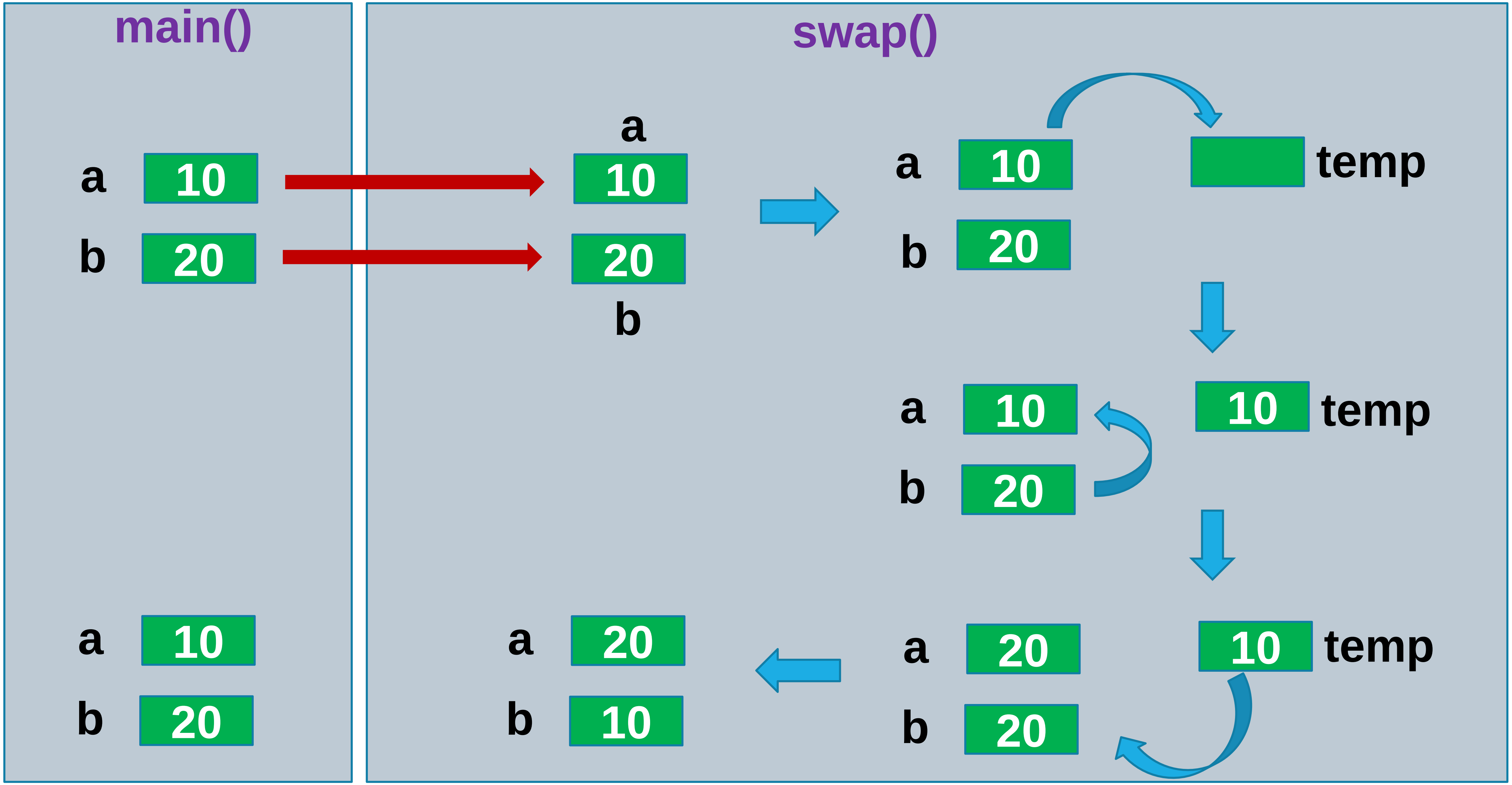
SWAP Program using a function

- What if the same program has to be written using a separate function.
- Then the swapping part is moved to a separate function and that function is called by passing two values to be swapped as parameters .

```c
#include <stdio.h>
int main()
{
    int a,b;
    a = 10;
    b = 20;
    printf("Before Swap\n");
    printf(" a : %d\n b : %d\n ",a,b);
    swap(a,b);
    printf("\nAfter Swap\n\n");
    printf(" a : %d\n b : %d\n ",a,b);
    return 0;
}
```

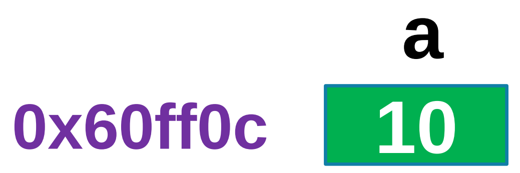
```
void swap(int a,int b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

Can you guess the output?


main()
swap()
a
10
b
20
a
10
20
b
a
10
temp
b
20
a
10
10
temp
b
20
a
20
10
temp
b
20
a
20
b
10
a
10
b
20

Using Memory Address

- In the previous swap example, Instead of copying, If the variables them selves in the main() can be shared with the swap() function, the swapping could be successfully achieved.
- Memory address of the variable can be used as the solution in this matter.

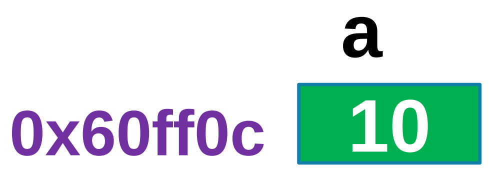


- Memory address can be accessed using ampersand (&) operator.

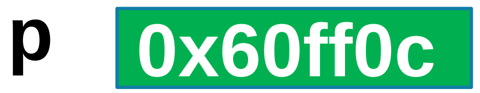
&a → **0x60ff0c**

Introduction to Pointers

- A **pointer** is a memory address of a variable. (Reference)



- There are special kind of variables that can store the memory address (or the reference) of another variable.
- Those variables are called **Pointer variables** (Pointer type variables) and they **holds a pointer** to another variable.



- Here we could say **p is pointing at a** variable.

Pointer Variable Declaration

- Like any variable, a pointer variable should be declared before using it to store a memory address(a pointer or a reference).
- To declare a pointer variable, asterisk(*) character is used followed by the type of the variable that this pointer variable is pointing at.
- There are two ways of declaring pointer type variables.

```
DataType* VariableName
DataType  *VariableName
```

```
int* p;
```

```
int *p;
```

Pointer Variable Declaration

- Non-pointer variables can be declared together if the types of the variables are the same.

  e.g :

```
int a, b, c ;
```

- If this is done with the pointer variables, the first declaration method cannot be used.

```
int* a, b, c ;
```

- The second method can be used where each name of the variable should be marked with the asterisk(*).

```
int *a, *b, *c ;
```

Using Pointers

There are three things that should be needed when working with pointers

- Get the address of a variable.

  This could be done using **&** operator

```
int *p = &x;
```

- Read the contents of an address(**Dereference**).

  This could be done using ***** operator

```
int y = *p;
```

- Change the contents of an address.

  This could also be done using ***** operator

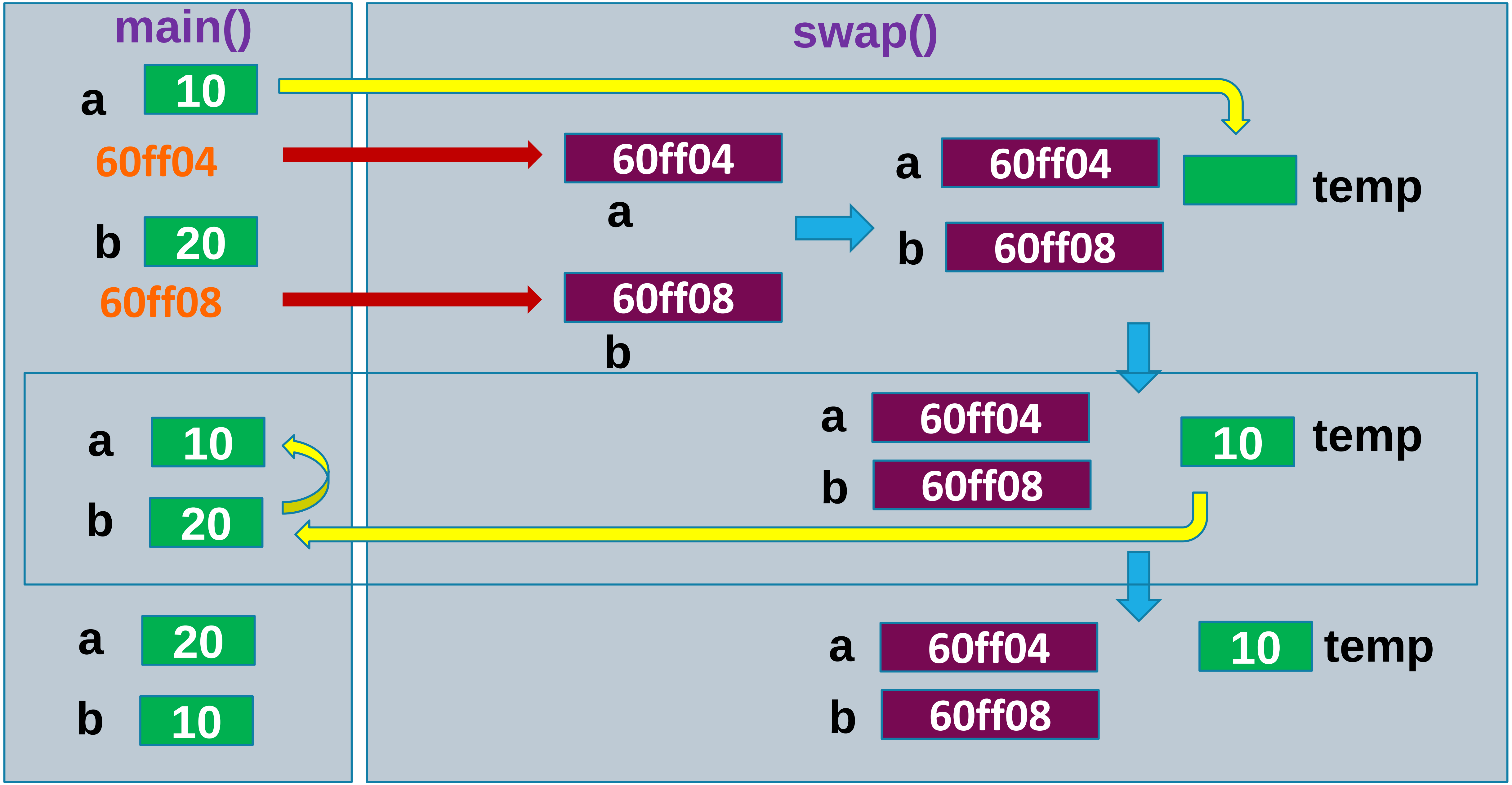
```
 *p = 5;
```

Pointers and Arrays

```
#include <stdio.h>
int main () {
   int a =10;
   int *ptr = &a;
   printf("Address of a variable: %x\n",ptr );
   printf("Value of a variable: %d\n",*ptr );
   printf("Address of ptr variable: %x\n",&ptr );
   return 0;
}
```

```
#include <stdio.h>
int main()
{
    int a,b;
    a = 10;
    b = 20;
    printf("Before Swap\n");
    printf(" a : %d\n b : %d\n ",a,b);
    swap(&a,&b);
    printf("\nAfter Swap\n\n");
    printf(" a : %d\n b : %d\n ",a,b);
    return 0;
}
```

```
void swap(int *a,int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
```


main()
swap()
a
10
60ff04
b
20
60ff08
60ff04
a
60ff08
b
a
60ff04
b
60ff08
temp
a
10
b
20
a
60ff04
b
60ff08
10
temp
a
20
b
10
a
60ff04
b
60ff08
10
temp

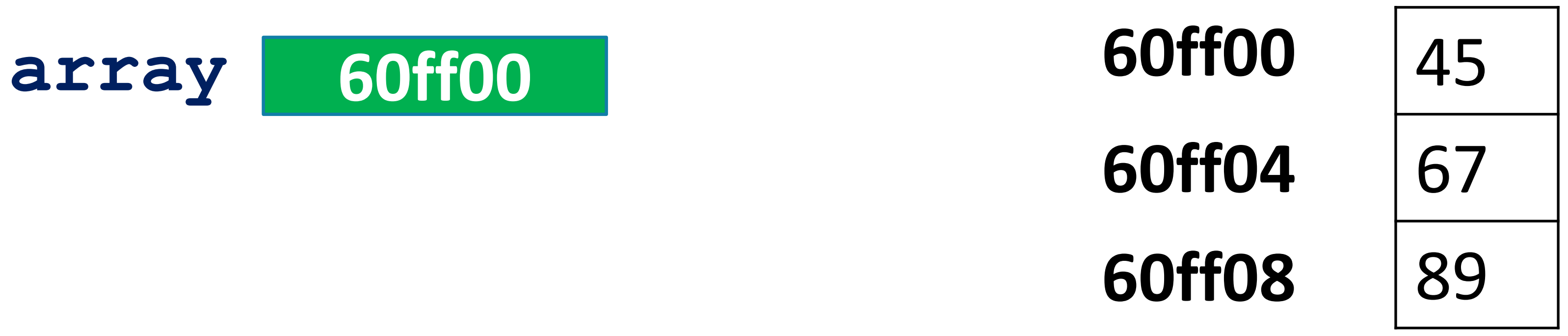
Pointers and Arrays

- There is a very close relation between Arrays and Pointers.
- When array is declared, This variable, **array**, is an extra-big box: three ints' worth of storage.

```
    int array[] = { 45, 67, 89 };
```

- In most places, when the name **array** is used **again**, **a pointer to its first element** is actually being used.
- This is called "**decaying**": the array "decays" to a pointer.

Pointers and Arrays



```
int main () {
    int array[] = {2,4,5};
    printf("Address of array[0] : %x\n", array);
    printf(“Value of array[0]: %d\n", * array );
    return 0;
}
```

Pointers and Arrays

There is several differences between arrays and non-array pointer variables.

- sizeof(an array) is the total size of the array.
- If you use the & operator on an array variable, the result equals the array variable itself.

array == &array == &array[0]

- An array variable can't point anywhere else.

Pointers and Arrays

```
#include <stdio.h>

int main () {
   int var2[] = {2,4,5};
   printf("Address of var1 variable: %x\n",var2 );
   printf("Address of var1 variable: %x\n",&var2 );
   printf("Address of var1 variable: %x\n",&var2[0] );
   return 0;
}
```

Pointers and Strings

```
#include <stdio.h>
int main()
{
    char quote[] = "Cookies make you fat" ;
    printf("quote occupies %i bytes\n",sizeof(quote));
    fortune_cookie(quote);
    return 0;
}
```

Pointers and Strings

```
void fortune_cookie(char *msg)
{
    printf("Message reads: %s\n", msg);
    *msg = 'R';
    printf("Message reads: %s\n", msg);
    msg = "Cookies don't make you fat";
    printf("Message reads: %s\n", msg);
    printf("msg occupies %i bytes\n", sizeof(msg));
}
```

Questions?