C introduction

Pointers

Contents

RGB

Consider a function that calculates the RGB values of a hex color string:

```
int calcRGB(char hexString[]) {
    ...           /* converting hexString into RGB values */
    return ???;
}
```

- It is not possible to return 3 values.

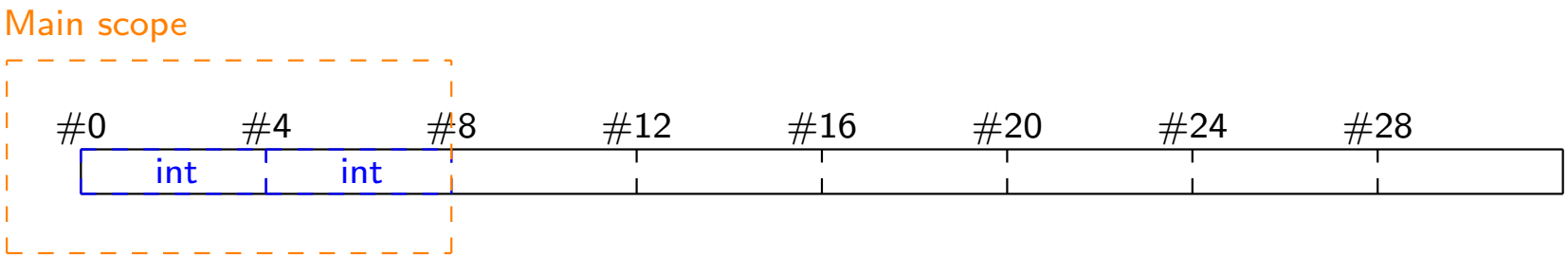
We could write 3 different functions:

```
int calcR(char hexString[]) { ... } /* returns R value */
int calcG(char hexString[]) { ... } /* returns G value */
int calcB(char hexString[]) { ... } /* returns B value */
```

Or we declare the 3 variables before the function call and just tell the function were to put the values.

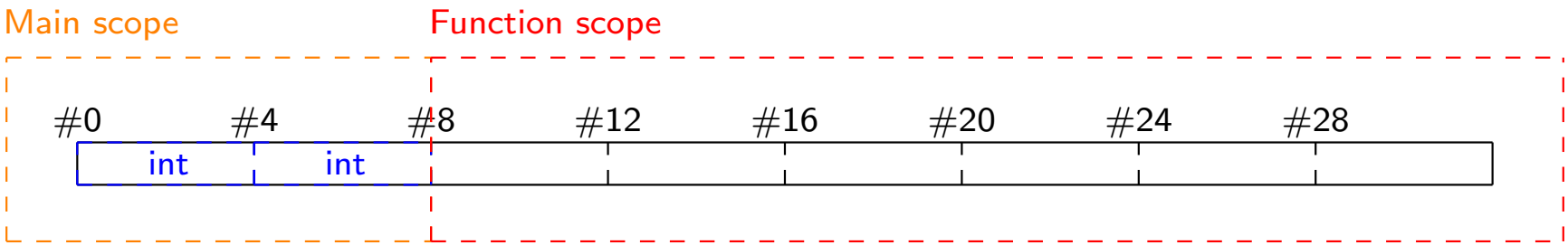
Memory again

- You have two int variables in your main function.



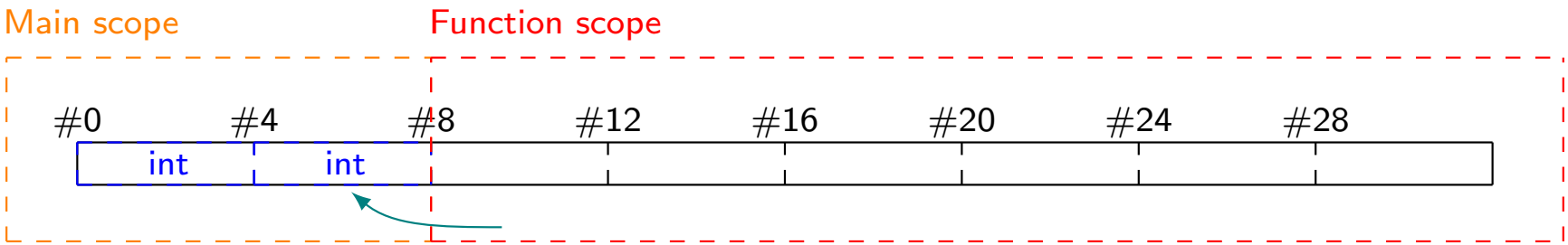
Memory again

- You have two int variables in your main function.
- Now you call a function



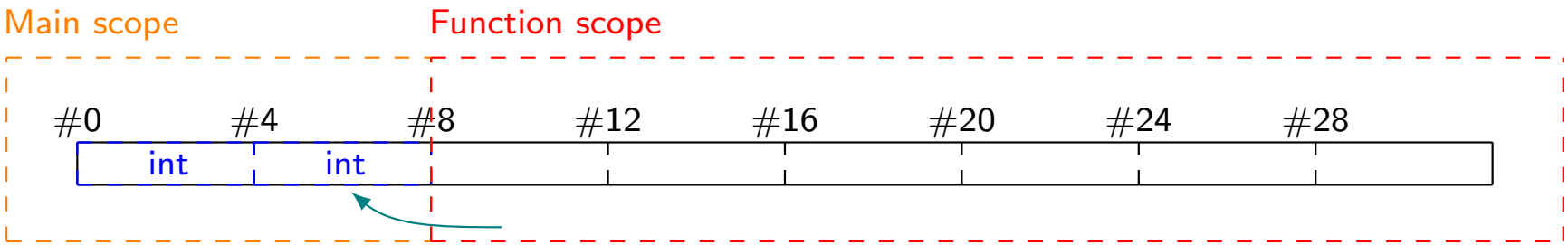
Memory again

- You have two int variables in your main function.
- Now you call a function
- You want to change the value of a variable in the main scope



Memory again

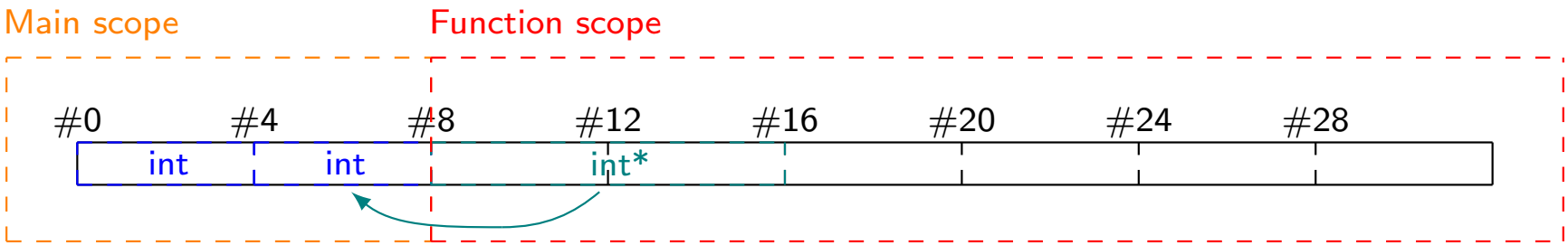
- You have two int variables in your main function.
- Now you call a function
- You want to change the value of a variable in the main scope



- You'll have to pass the address of this variable

Memory again

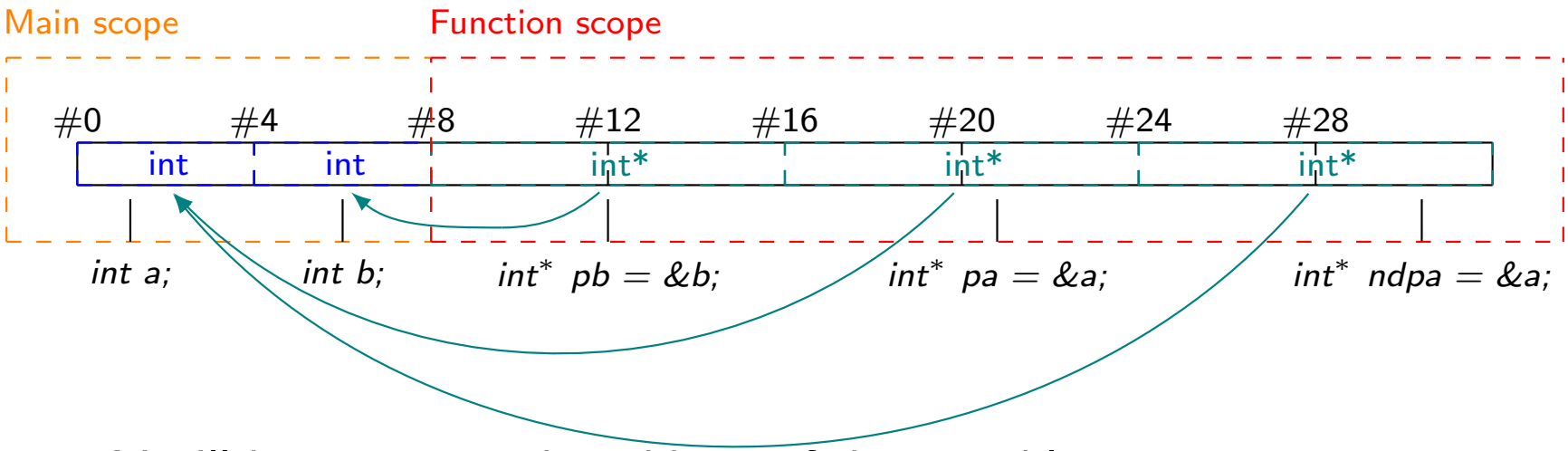
- You have two int variables in your main function.
- Now you call a function
- You want to change the value of a variable in the main scope



- You'll have to pass the address of this variable
- This address is stored in a *pointer* variable

Memory again

- You have two int variables in your main function.
- Now you call a function
- You want to change the value of a variable in the main scope



- You'll have to pass the address of this variable
- This address is stored in a *pointer* variable
- This method is called *call by reference*

Operators

- To declare a Pointer, use the *dereference operator* *****
- To get the address of a variable, C comes with the *adress operator* **&**
- To access the variable a pointer points to, dereference it with the *dereference operator* *****

```
int a = 42;
int* pa;      /* declare an int pointer*/
pa = &a;      /* initialize pa as pointer to a */
*pa = 13;     /* change a */
```

incrementing and decrementing

If you want to increment or decrement the variable a pointer points to, you have to use Parentheses.

```
int a = 42;
int* pa = &a;     /* define pa as pointer to a */
(*pa)++;          /* increment a */
(*pa)--;          /* decrement a */
```

If you had not used the parentheses, you would have in-/decremented the pointer, not the variable it points to. Congratulations, you just invented pointer arithmetic but we will talk later about that.

Back to RGB

Now we can think of the RGB function as one function, taking the hexString and 3 Pointers:

```
void calcRGB(char hexString[], int* r, int* g, int* b) {
    ...
    *r = calculatedRValue;
    *g = calculatedGValue;
    *b = calculatedBValue;
}
```

Call it with

```
int r, g, b;
calcRGB("ffffff", &r, &g, &b);
```

- You now should understand how scanf works.

Returning pointers

Pointers can be return values, too.

But

```
int* someFunction() {
    int a = 42;
    return &a;
}
```

- Dafuq did just happen?

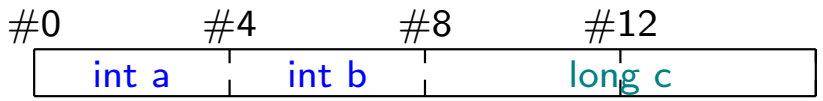
Exercises

- You are now able to solve tasks 22 and 23.

p++

You can in-/decrement a pointer. If you do so, the address it points to will change.
The address changes by the size of the pointer type.

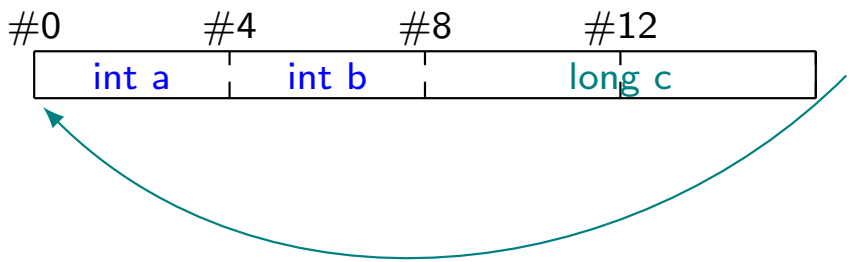
```
int a, b;
long c;
int* p = &a;
p++;
p++;
p++;
```



p++

You can in-/decrement a pointer. If you do so, the address it points to will change.
The address changes by the size of the pointer type.

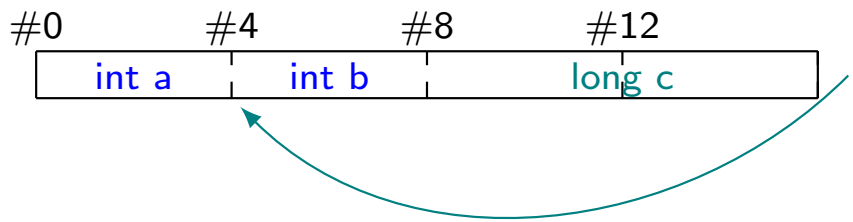
```
int a, b;
long c;
int* p = &a;
p++;
p++;
p++;
```



p++

You can in-/decrement a pointer. If you do so, the address it points to will change.
The address changes by the size of the pointer type.

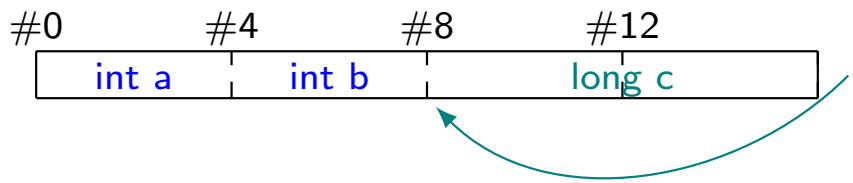
```
int a, b;
long c;
int* p = &a;
p++;
p++;
p++;
```



p++

You can in-/decrement a pointer. If you do so, the address it points to will change.
The address changes by the size of the pointer type.

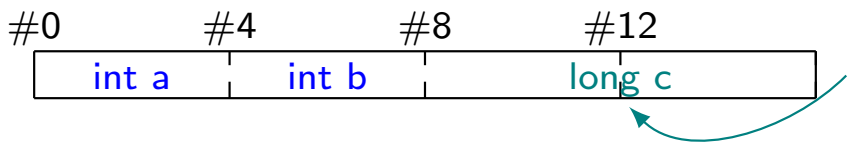
```
int a, b;
long c;
int* p = &a;
p++;
p++;
p++;
```



p++

You can in-/decrement a pointer. If you do so, the address it points to will change.
The address changes by the size of the pointer type.

```
int a, b;
long c;
int* p = &a;
p++;
p++;
p++;
```



- Since the pointer is of type *int**, the target adress moves only the size of *int*

Pointer and Arrays

The identifier of an array can be considered a pointer.
This means we can consider the index as an offset for the pointer and access array elements trough pointer arithmetic:

```
int leet[4] = {1, 3, 3, 7};
int* pleet = leet;
*(pleet++) = 2;
printf("%d %d\n", *pleet, *(pleet + 2));
```

- What is the output?

Pointer and Arrays

The identifier of an array can be considered a pointer.
This means we can consider the index as an offset for the pointer and access array elements trough pointer arithmetic:

```
int leet[4] = {1, 3, 3, 7};
int* pleet = leet;
*(pleet++) = 2;
printf("%d %d\n", *pleet, *(pleet + 2));
```

- What is the output?

```
2 7
```

- Why?

Pointer and Arrays

The identifier of an array can be considered a pointer.
This means we can consider the index as an offset for the pointer and access array elements trough pointer arithmetic:

```
int leet[4] = {1, 3, 3, 7};
int* pleet = leet;
*(pleet++) = 2;
printf("%d %d\n", *pleet, *(pleet + 2));
```

- What is the output?

```
2 7
```

- Why?
  - Hint: Wasn't there a difference between $c++$ and $++c$?

argc and **argv**

You can pass strings to the main function by writing them on the command line.

```
$ ./a.out string1 longer_string2
```

- They are stored in *argv*[1]
- *argv* is an array of pointers to the first character of a string
- **Caution:** *argv[0]* is the name by which you called the program
- *argc*[2] is the number of strings stored in *argv*

[1] Short for *argument value*
[2] Short for *argument count*

Exercises

- You are now able to solve tasks 24 and 25.