Scope

Learn what scope is, the different ways of defining scope, and how a program uses scope.

When we write programs in C, we define many different names for variables, functions, and other identifiers. We may find that our access to these named components is unavailable in certain parts of our program.

Is there a way for us to know when certain names are available? As it turns out, the *scope* of the name tells us exactly this.

What Is Scope?

Scope refers to the part of a program where a name has a meaning.
For example:

```
int someFunction() {
  int myVariableName = 20;
}

int main() {
  char myVariableName[] = "10";
}
```

In the example above:

- `myVariableName` in `someFunction()` refers to an `int` variable whose value is `20`
- `myVariableName` in `main()` refers to an `char[]` variable whose value is `"10"`
- Both these variables have the same name but different meaning depending on their scope

We've actually already done a lot of scoping through out this course!

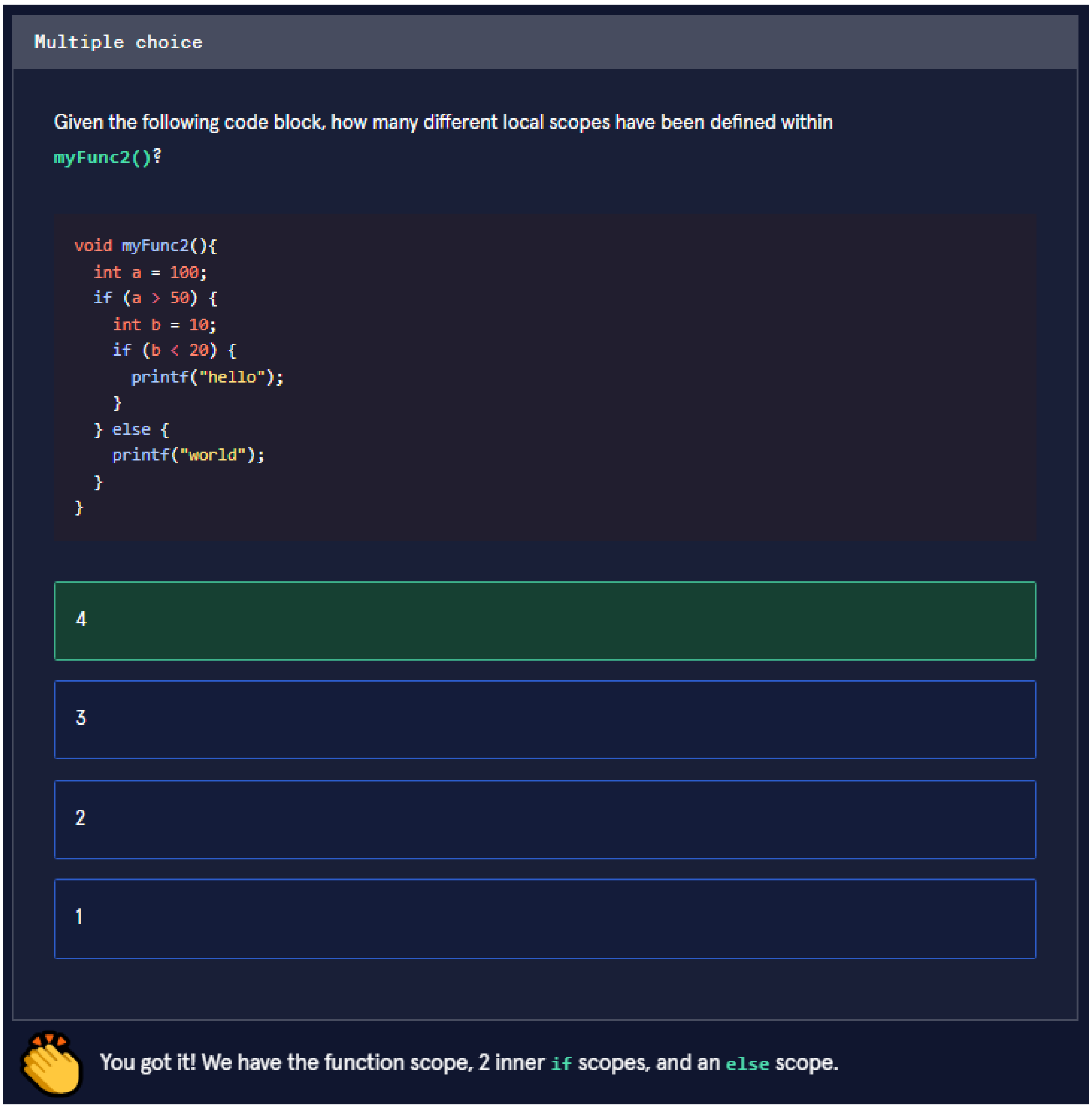
In C, scopes are defined within curly braces, `{}`, and therefore can be created using constructs like `if` statements and functions.
For example:

```
void myFunc() {
  // function scope
  int b = 10;
  if (b > 5) {
    // if scope
    int c = 4;
  }
}
```

There are two scopes in the above example:

- A scope within the `myFunc()` function defined with the outer braces

- A scope within the `if` statement curly braces, also inside the `myFunc()` scope
- The `if` braces “enclose” the meaning of `c` within it and make it unknown outside of `if` block.
- The `myFunc()` braces “enclose” the meaning of `b` within it and make it unknown outside of the function.



Multiple choice

Given the following code block, how many different local scopes have been defined within `myFunc2()`?

```c
void myFunc2(){
  int a = 100;
  if (a > 50) {
    int b = 10;
    if (b < 20) {
      printf("hello");
    }
  } else {
    printf("world");
  }
}
```

[✓ 4]

3

2

1

You got it! We have the function scope, 2 inner `if` scopes, and an `else` scope.

Local Scope

Let’s look at another example:

```c
void myFunc() {
  int b = 200;
}

int main() {
  int a = 10;
  printf("The value of b is %d", b); // `printf()` trying to access `b`
}
```

The above example defines the name `a` known only to `main()` and `b` known only to `myFunc()`.

The local scope means that:

- `b` can only be accessed within the function body of `myFunc()`
- Outside of `myFunc()`, `b` cannot be accessed. In other words `b` is not known to other scopes, and therefore they don't have it declared

If we tried to run this program, we would get a compilation error.

`error: 'b' undeclared (first use in this function)`

This is because the name `b` is not defined in the `main()` function local scope. Similar situations occur when we use other constructs like `if` statements.

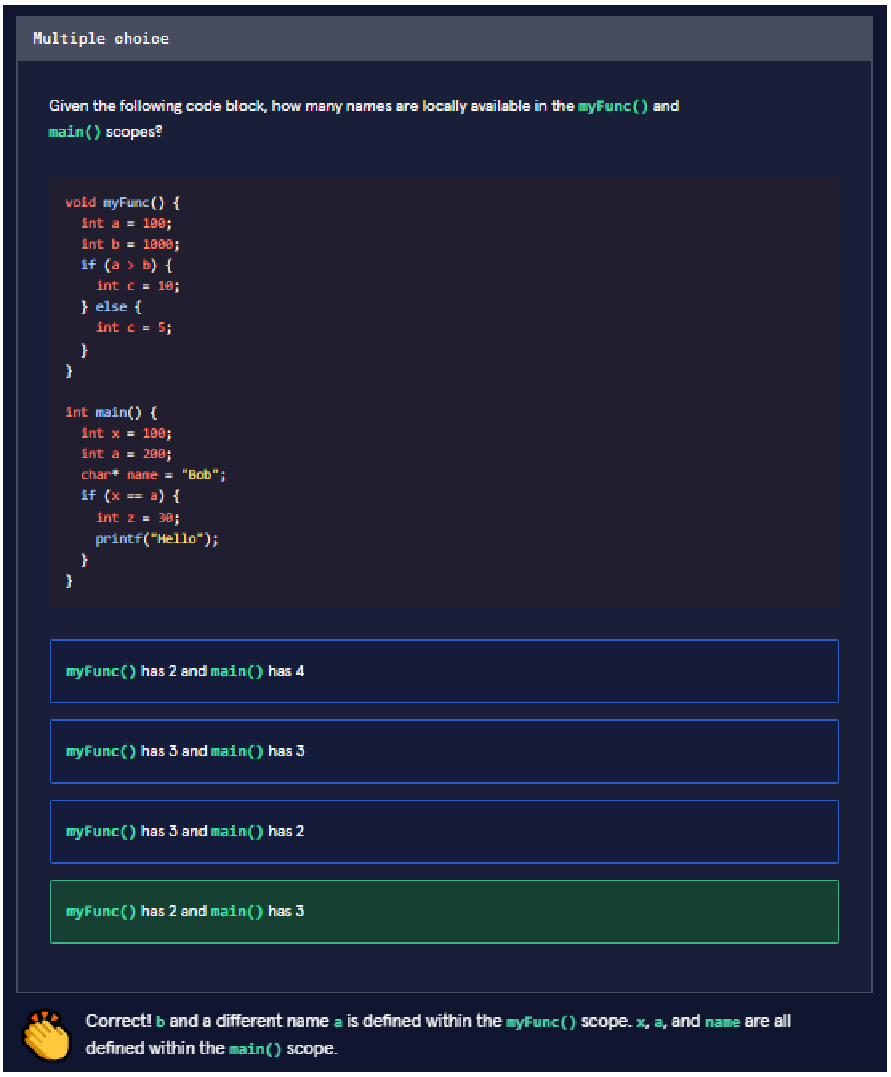
```
void myFunc() {
  printf("The value of c is %d", c);
}

int main() {
  int c = 10;
  if (c > 4) {
    int b = 20;
  }
  printf("The value of b is %d", b);
}
```

In the example above, the following would occur:

- The name `c` is defined locally within the body of `main()` and is unknown outside of the function. `c` is part of the `main()` scope.
- `b` is defined locally within the body of the `if` statement and is unknown outside of the `if` block. `b` is part of the `if` block scope.
- `b` may appear as if it's part of `main()` scope but it's really the `if` scope that is defined inside the `main()` scope.

We would get another compilation error here because we are trying to access the name `b` in the `main()` scope, when it is only defined in the `if` scope.



Multiple choice

Given the following code block, how many names are locally available in the `myFunc()` and `main()` scopes?

```
void myFunc() {
  int a = 100;
  int b = 1000;
  if (a > b) {
    int c = 10;
  } else {
    int c = 5;
  }
}

int main() {
  int x = 100;
  int a = 200;
  char* name = "Bob";
  if (x == a) {
    int z = 30;
    printf("Hello");
  }
}
```

- `myFunc()` has 2 and `main()` has 4
- `myFunc()` has 3 and `main()` has 3
- `myFunc()` has 3 and `main()` has 2
- [✓ `myFunc()` has 2 and `main()` has 3]

Correct! `b` and a different name `a` is defined within the `myFunc()` scope. `x`, `a`, and `name` are all defined within the `main()` scope.

Global Scope

Now that we've covered local scope we can take things global! In C, there exists a top level scope called the global scope. The global scope contains declared names, but access to those names work differently than the local scope.

Let's look at an example:

```
// global scope
int b = 10;

void myFunc() {
  // myFunc() local scope
  printf("Inside myFunc, b value is:%d", b);
}

int main() {
  // main() local scope
  int a = 200;
  printf("The value of b is %d", b);
}
```

In the example:

- The name `b` is defined globally

- A global name is defined outside all functions, including `main()`
- A global name is available to access and modify everywhere in the program including inside functions

Although global names can be useful, they should be avoided because:

- As our program grows, it will be harder to track where a global variable is being read or modified. This makes things harder to test and debug.
- With a large enough program, we might run into naming conflicts between global variables.
- As a general rule, functions shouldn't modify the state of variables outside of its local scope. This keeps function logic isolated from each other and keeps your code reusable and modular.

Parent And Child Scoping

So far we've established that a name defined inside an `if` scope cannot be accessed outside that scope. We've also been able to access a name defined in a function scope, like `main()`, inside an `if` scope. This is possible because of the relationship between scopes.

Let's look at this example:

```
int f = 10;

int main() {
  int a = 20;
  if (a > 10) {
    int b = 30;
    printf("The sum of b, a, and f is: %d", b + a + f);
  }
}
```

We know that `f` is a global name and is available everywhere in the program and that `a` is local to `main()` and `b` is local to the `if` block. How does C know to look at the global scope for `f` and `main()` scope for `a`? This is the parent and child relationship of scopes!

- The global scope is the parent of the `main()` scope. Alternatively, we can say `main()` is the child of the global scope because it is defined within the global scope.
- The `main()` scope is the parent of the `if` scope, making the `if` scope the child of the `main()` scope.

The compiler takes the following steps to find the definition of `f` and `a`:

1. The compiler tries to look for the definition of `f` and `a` locally in the `if` block.
2. If `f` or `a` (or both) can't be found, it checks the parent scope of the `if` block, the `main()` scope in this case.

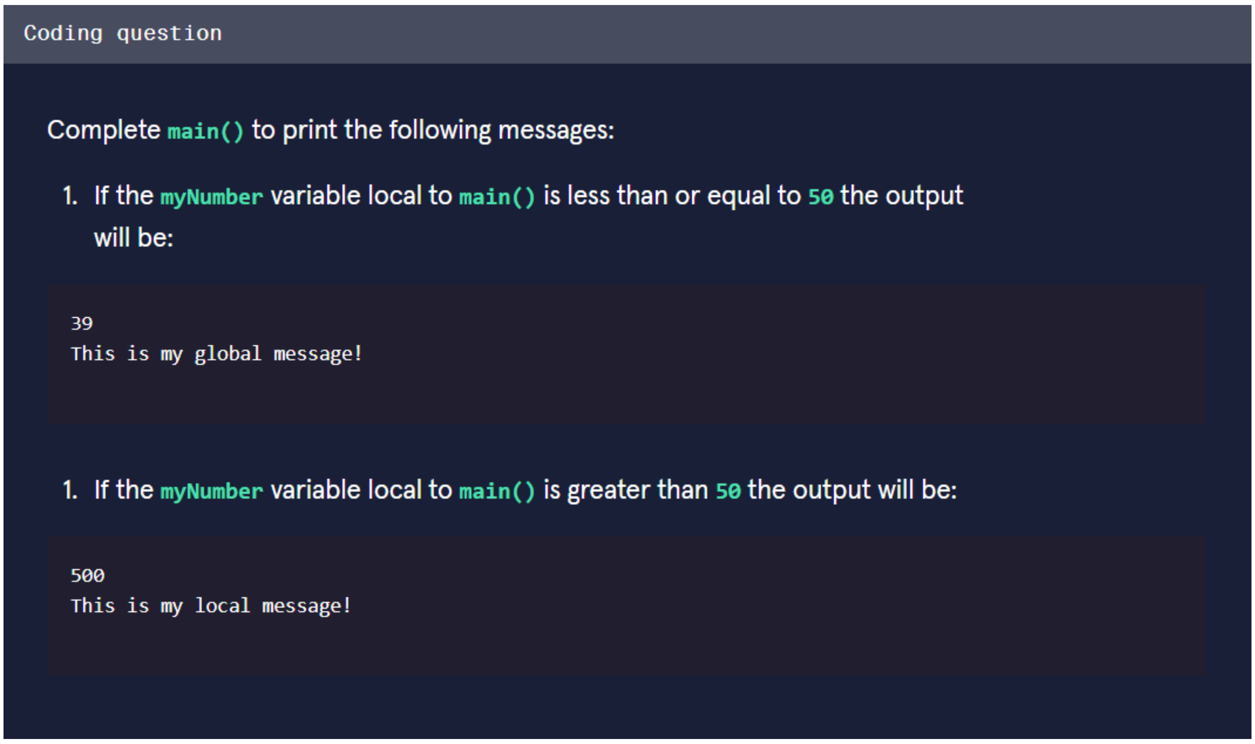
3. If `f` or `a` (or both) can’t be found, it checks the parent scope of `main()`, the global scope in this case.
4. These steps continue repeatedly until all the names are found or there are no parent scopes left to check. At this point a compilation error would occur.

In this example, we stop looking for `a` at the `main()` scope because its defined there and we stop looking for `f` at the global scope because `f` is defined there.

Great job learning about scope! We learned:

- What scope is
- Different types of scopes (local and global)
- Parent and child relationship between scopes

Try and complete the code challenge to test your understanding.



Coding question

Complete `main()` to print the following messages:

1. If the `myNumber` variable local to `main()` is less than or equal to `50` the output will be:

```
39
This is my global message!
```

1. If the `myNumber` variable local to `main()` is greater than `50` the output will be:

```
500
This is my local message!
```

```c
#include <stdio.h>

// Write your code below...

char* myMessage = "This is my global message!";

void myFunc() {
  char* myMessage = "This is my local message!";
  printf("%s\n", myMessage);
}
```

```c
int main() {
  // You can change `myNumber` to be larger than `50`
  int myNumber = 39;

  if (myNumber <= 50) {
    printf("%d\n", myNumber);
    printf("%s\n", myMessage);
  } else {
    int myNumber = 500;
    printf("%d\n", myNumber);
    myFunc();
  }
}
```