CSE 3100: Systems Programming

Lecture 5: Arrays and Pointer Basics

Department of Computer Science and Engineering

University of Connecticut

1. **Arrays in C**

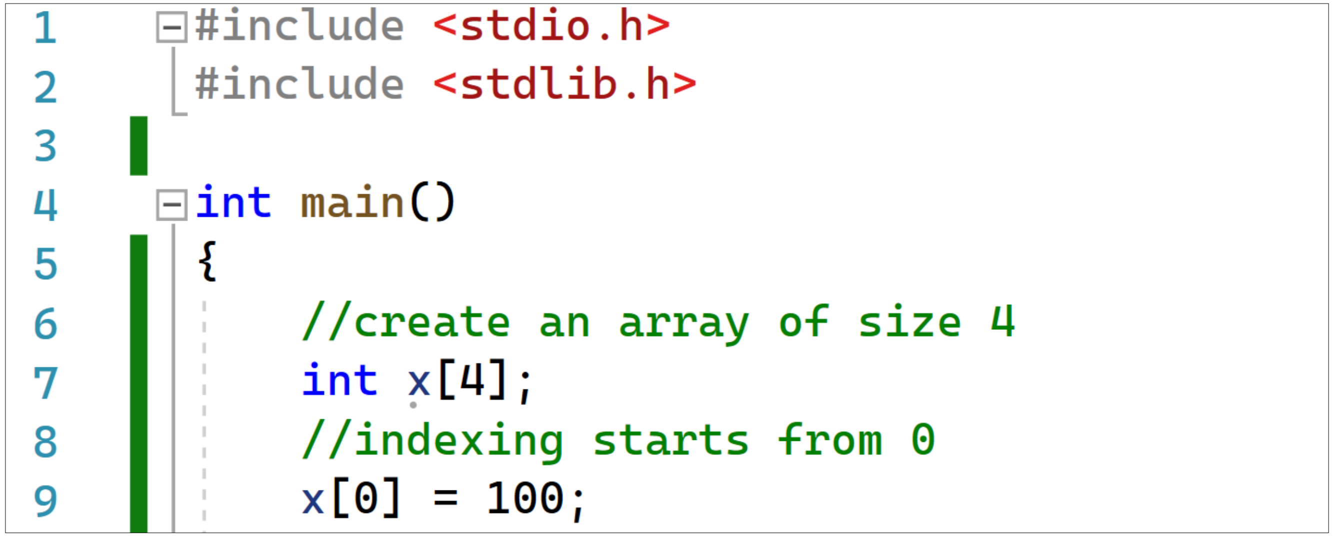
2. **Passing By Value**

3. **Passing By Reference**

4. **Pointers and Memory in C**

What happens if you want to associate multiple values with a variable?

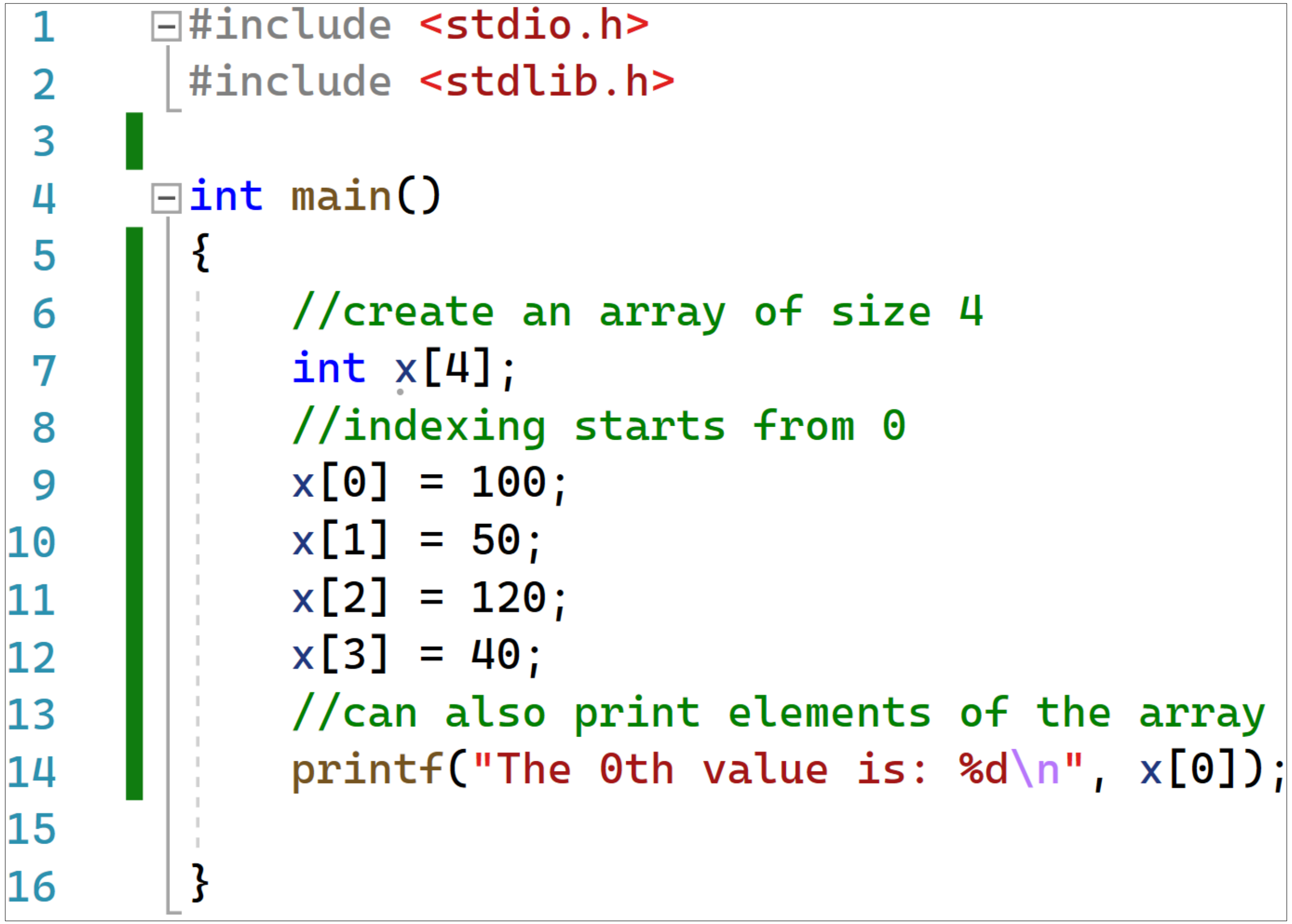
Use an array!



```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    //create an array of size 4
    int x[4];
    //indexing starts from 0
    x[0] = 100;
```

What happens if you want to associate multiple values with a variable?

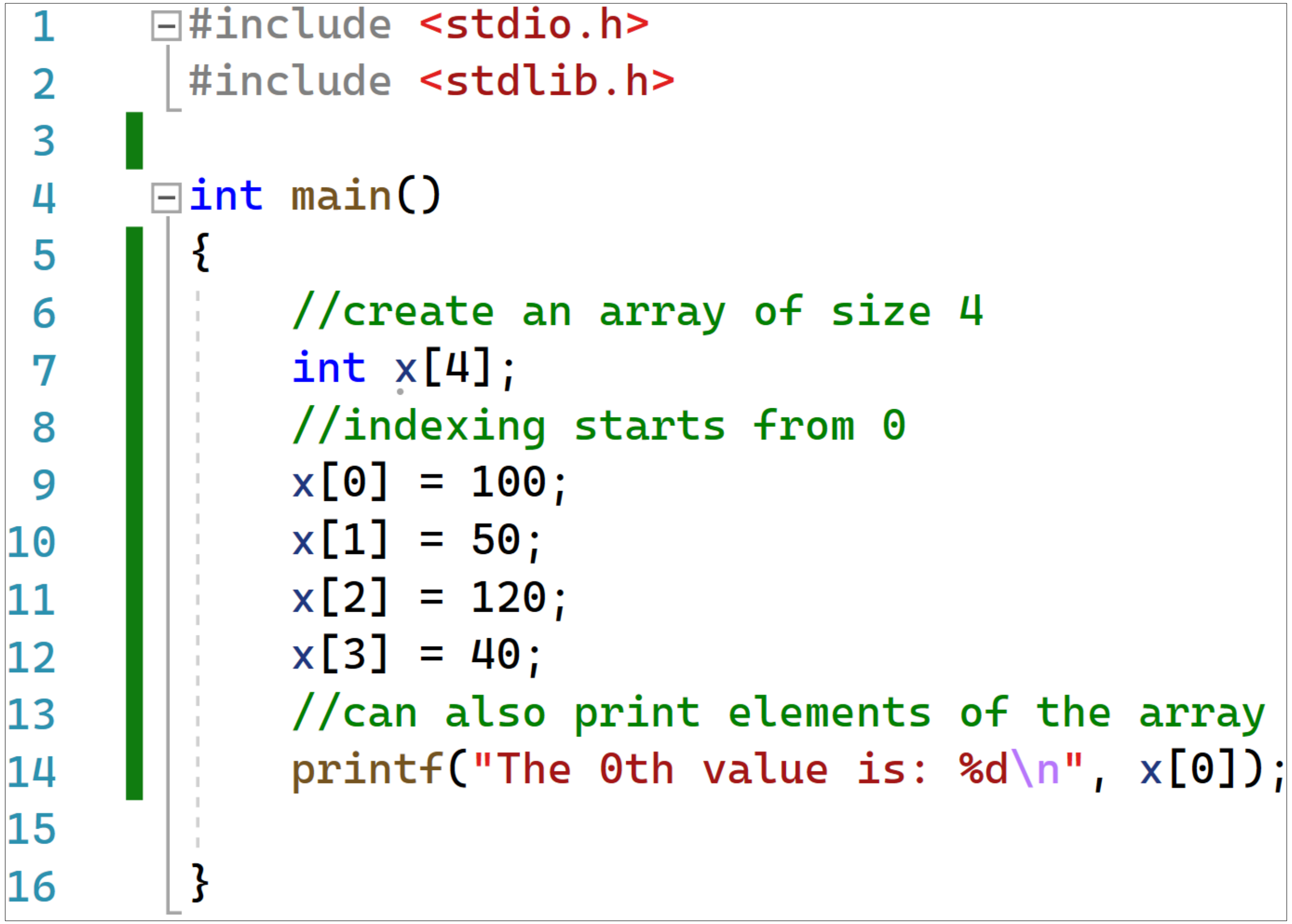
Use an array!



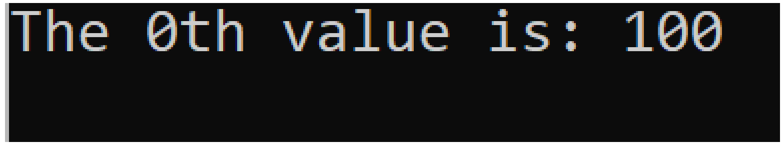
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    //create an array of size 4
    int x[4];
    //indexing starts from 0
    x[0] = 100;
    x[1] = 50;
    x[2] = 120;
    x[3] = 40;
    //can also print elements of the array
    printf("The 0th value is: %d\n", x[0]);

}
```

What happens if you want to associate multiple values with a variable?

Use an array!



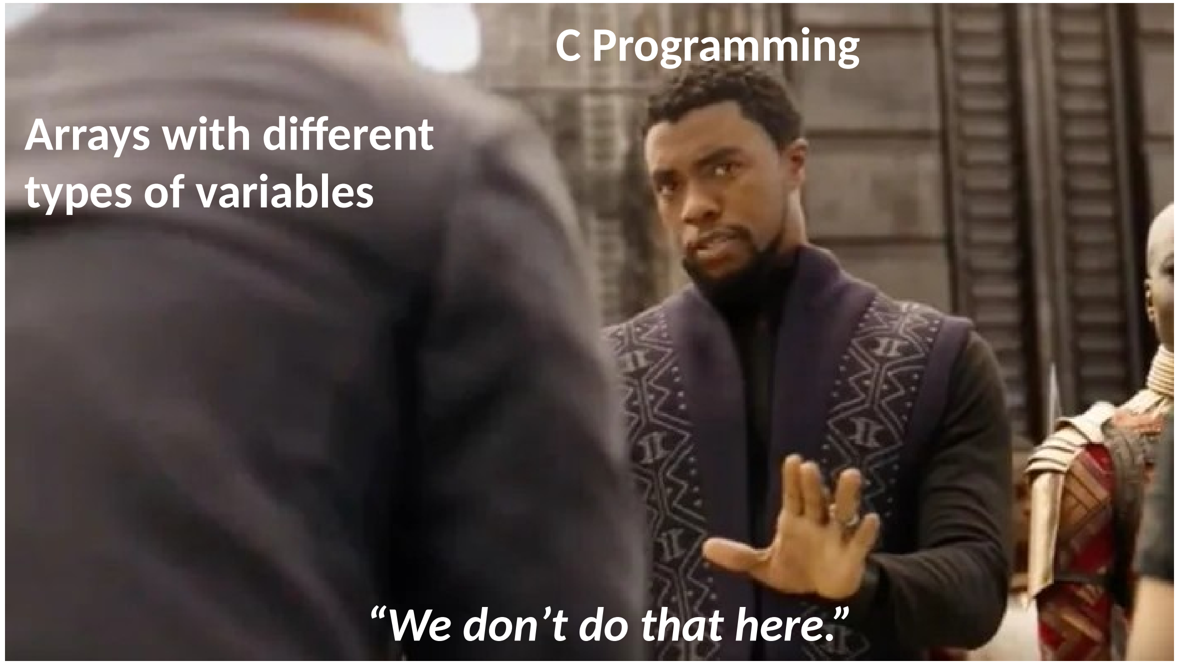
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    //create an array of size 4
    int x[4];
    //indexing starts from 0
    x[0] = 100;
    x[1] = 50;
    x[2] = 120;
    x[3] = 40;
    //can also print elements of the array
    printf("The 0th value is: %d\n", x[0]);

}
```



```
The 0th value is: 100
```

Arrays in C

- Arrays represent a linear, contiguous collection of "things"
- Each "thing" in the array has the same **fixed** type.
- Examples
  - Array of characters
  - Array of integers
  - Array of doubles
  - Arrays of arrays….

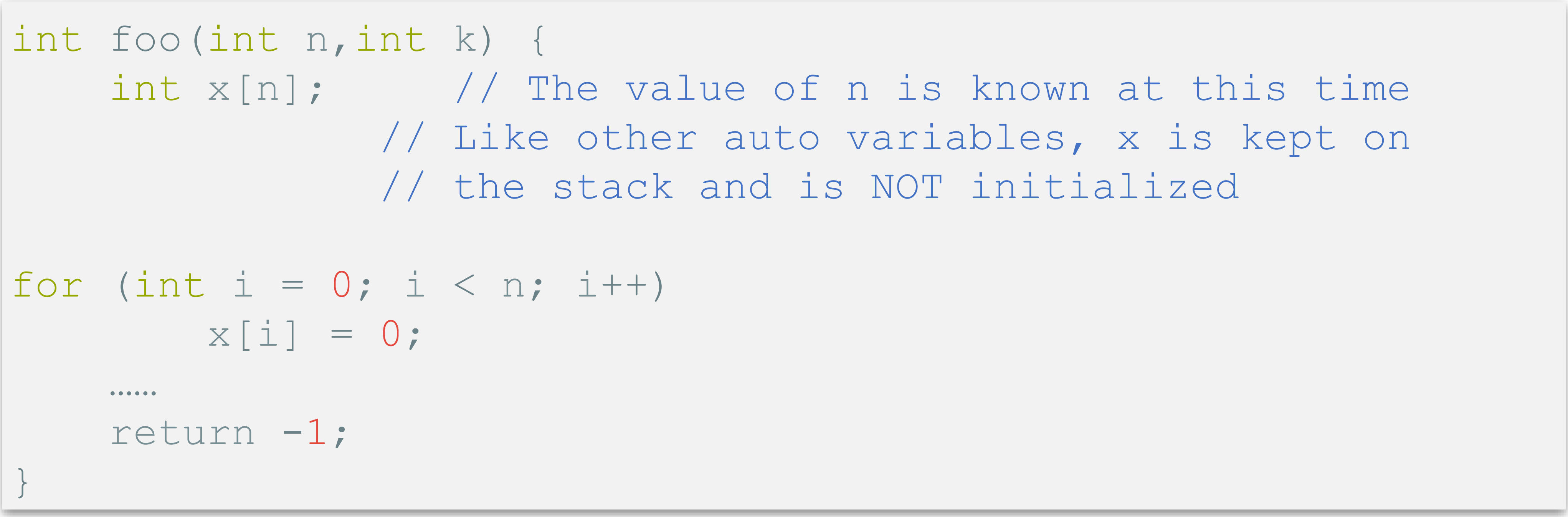

C Programming
Arrays with different types of variables
"We don't do that here."

Syntax for Array Initialization

```c
// initialize array with a list
int y[5] = {1, 2, 3, 4, 5};

// Number of elements is optional if all elements are listed
int z[] = {1, 2, 3, 4, 5};

// Specify the value of first 2 elements. The rest are set to 0
int a[5] = {1, 2};

// C99. b will have 1, 2, 0, 0, 5.
int b[5] = {1, 2, [4] = 5};
```

Arrays as Automatic Variables

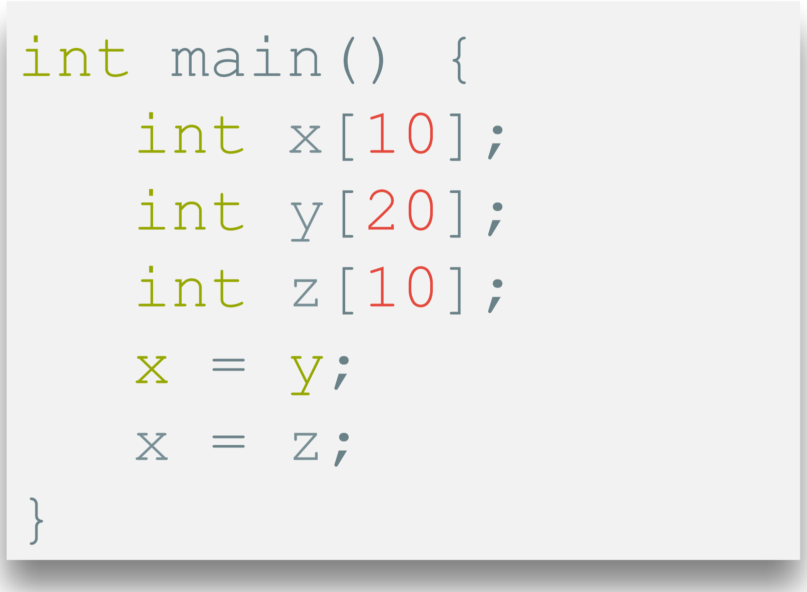
- You can declare arrays inside *any function or block*
  - Destroyed when exiting from the function or block
- Variable length arrays(VLA, C99) The size of your array can depend on *function arguments or other known values*



```
int foo(int n,int k) {
    int x[n];     // The value of n is known at this time
               // Like other auto variables, x is kept on
               // the stack and is NOT initialized

for (int i = 0; i < n; i++)
        x[i] = 0;
    ……
    return -1;
}
```

Array Assignment

- You ***cannot*** assign a whole array at once to another array
  - Even when the types match

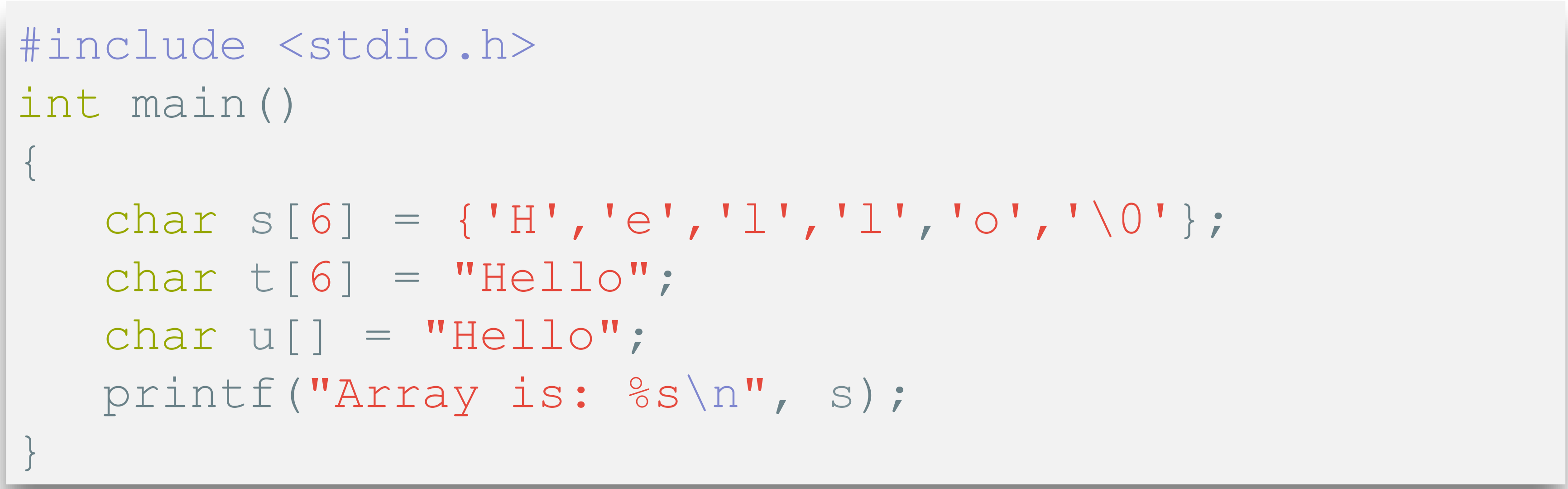


```
int main() {
    int x[10];
    int y[20];
    int z[10];
    x = y;
    x = z;
}
```

```
a.c: In function 'main':
a.c:5:7: error: assignment to
expression with array type
     x = y;
       ^
a.c:6:7: error: assignment to
expression with array type
     x = z;
       ^
```

Strings are Arrays Too!

- A string is a char array that ends with a 0 (null character)
    - Memory that stores 0 is part of the string
- It can be initialized with a list of characters
- or a string (double-quoted literal)



```
#include <stdio.h>
int main()
{
   char s[6] = {'H','e','l','l','o','\0'};
   char t[6] = "Hello";
   char u[] = "Hello";
   printf("Array is: %s\n", s);
}
```

What would this look like in memory?

```
#include <stdio.h>
int main()
{
   char s[6] =
{'H','e','l','l','o','\0'};
   char t[6] = "Hello";
   char u[] = "Hello";
   printf("Array is: %s\n", s);
}
```

	Addr.	Value
s[5]	505	0
s[4]	504	'o'
s[3]	503	'l'
s[2]	502	'l'
s[1]	501	'e'
s[0]	500	'H'

Arrays and Functions

- Arrays can be passed to functions!
  - With one big caveat…
- Calling convention in C
  - BY VALUE for everything….
  - EXCEPT arrays…
- Arrays are always passed BY REFERENCE
  - Passed as “pointers” – we‘ll look at pointers soon
- Functions cannot return arrays
  - No easy assignments

~~1. Arrays in C~~

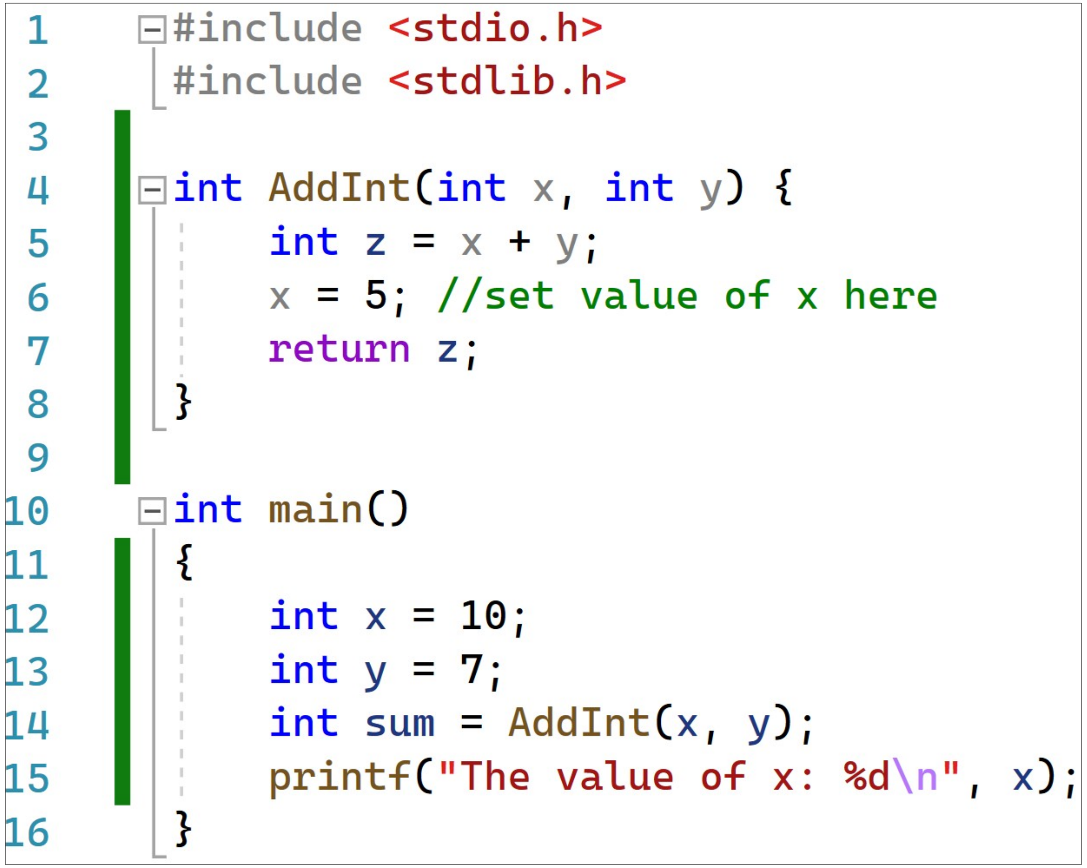
2. Passing By Value

3. Passing By Reference

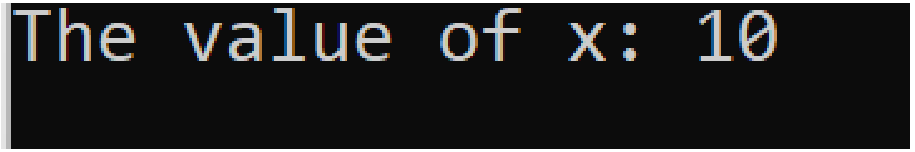
4. Pointers and Memory in C

What does passing by value mean?

In this code what will be the value of x at the end?



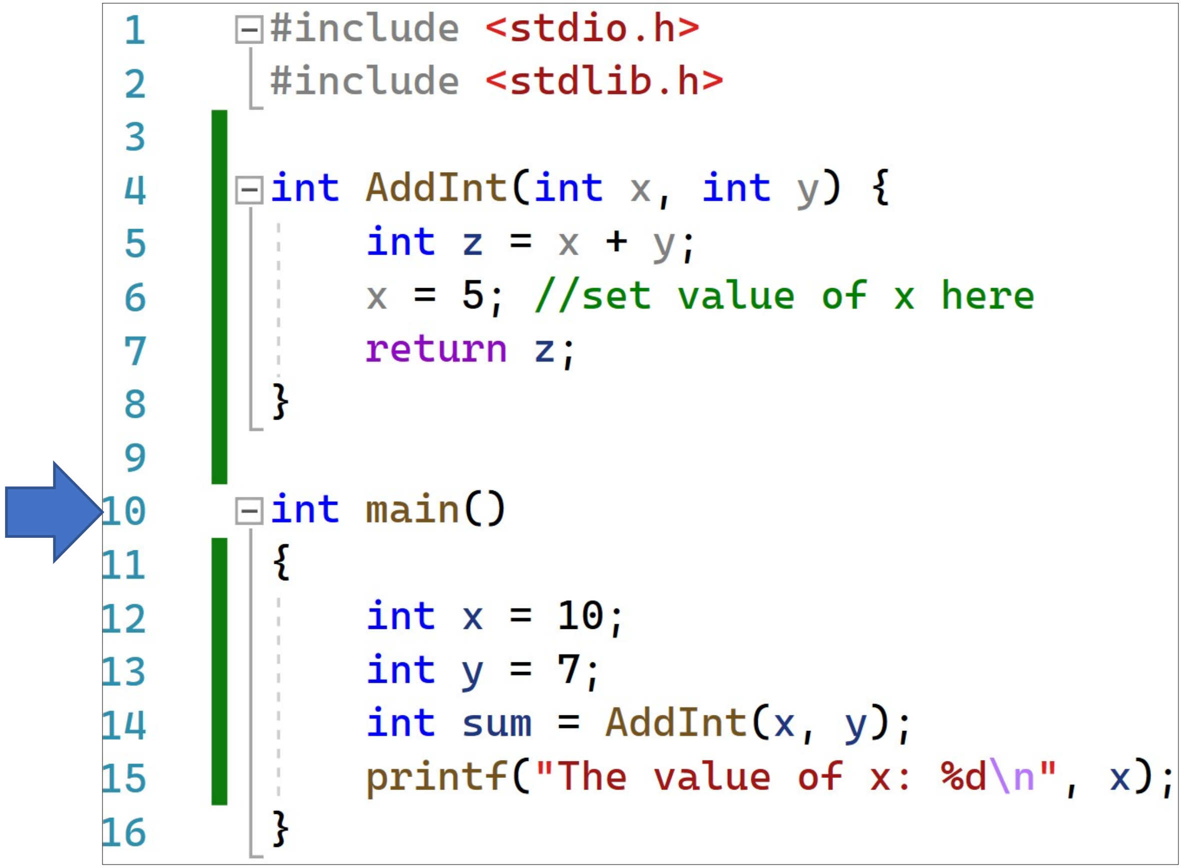
```c
#include <stdio.h>
#include <stdlib.h>

int AddInt(int x, int y) {
    int z = x + y;
    x = 5; //set value of x here
    return z;
}

int main()
{
    int x = 10;
    int y = 7;
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x);
}
```



```
The value of x: 10
```

What is going on here?

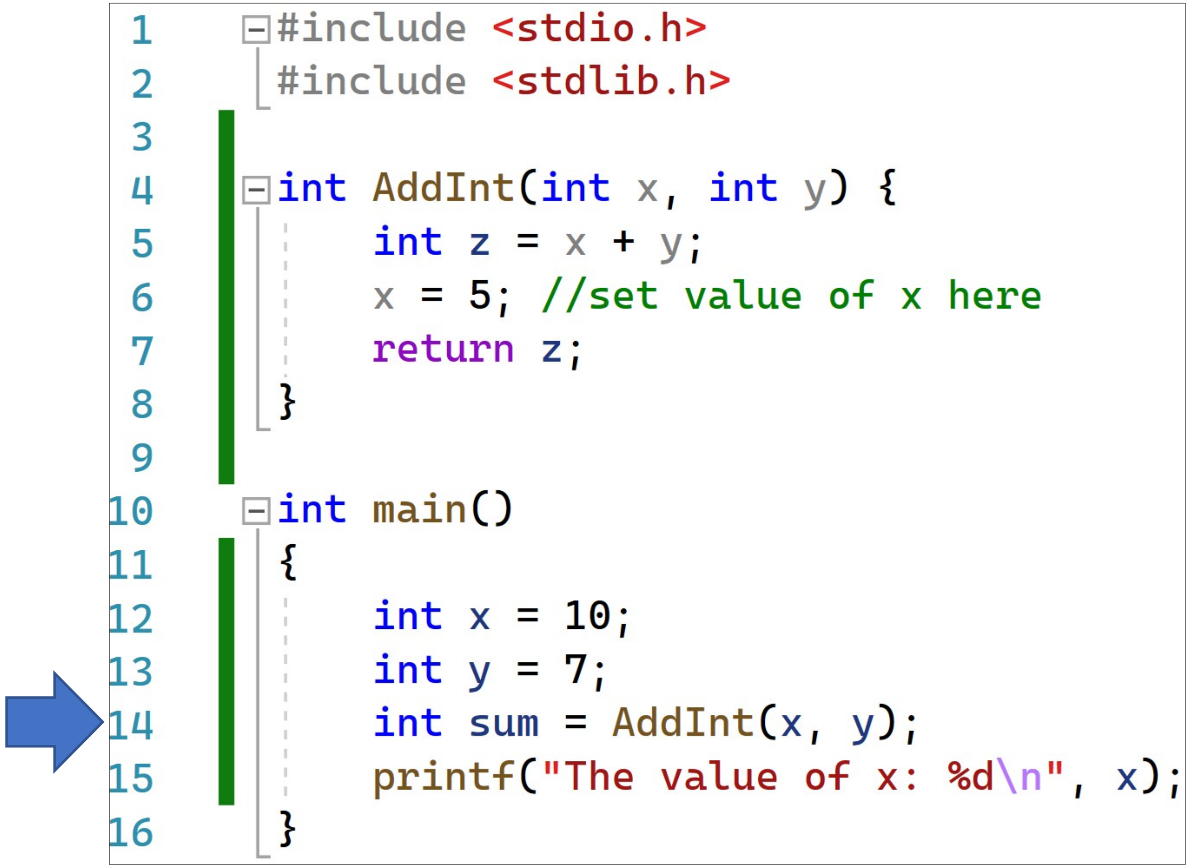
Start in main…



```c
#include <stdio.h>
#include <stdlib.h>

int AddInt(int x, int y) {
    int z = x + y;
    x = 5; //set value of x here
    return z;
}

int main()
{
    int x = 10;
    int y = 7;
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x);
}
```

What is going on here?

Start in main…



```c
#include <stdio.h>
#include <stdlib.h>

int AddInt(int x, int y) {
    int z = x + y;
    x = 5; //set value of x here
    return z;
}

int main()
{
    int x = 10;
    int y = 7;
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x);
}
```

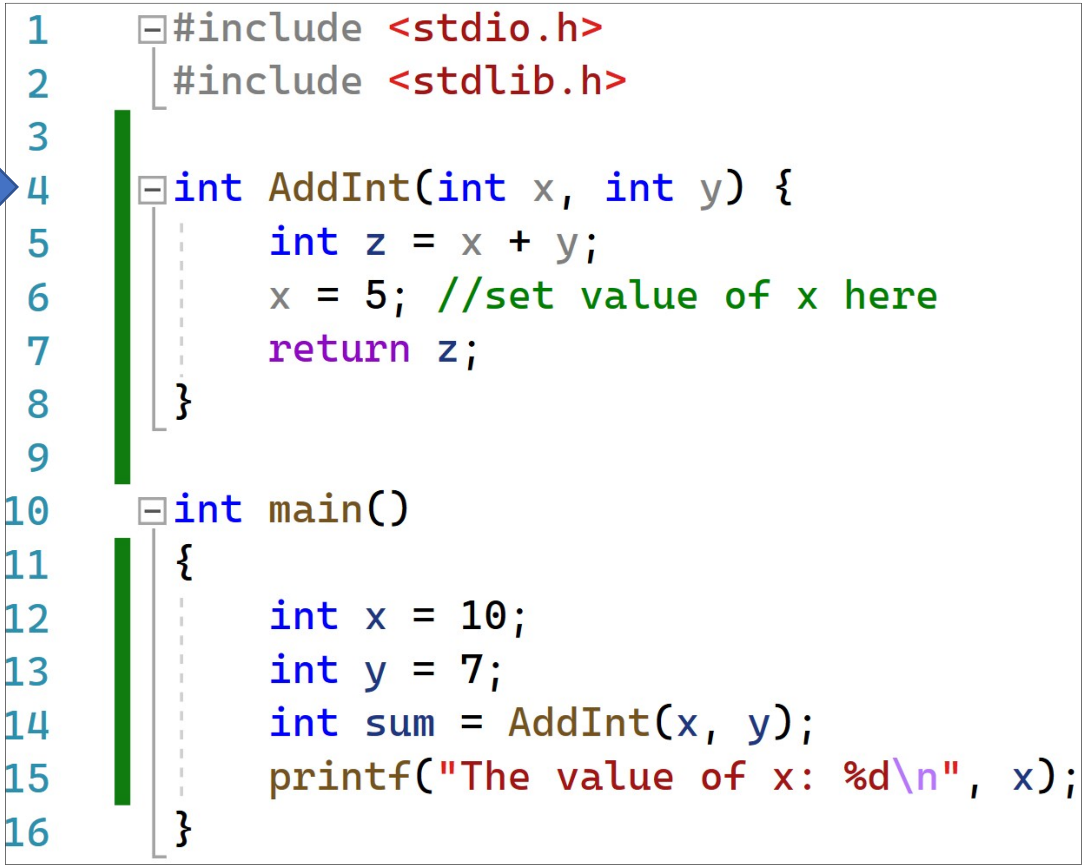
Variables in Main

x =10

y= 7

What is going on here?

Now go to AddInt…



```c
#include <stdio.h>
#include <stdlib.h>

int AddInt(int x, int y) {
    int z = x + y;
    x = 5; //set value of x here
    return z;
}

int main()
{
    int x = 10;
    int y = 7;
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x);
}
```

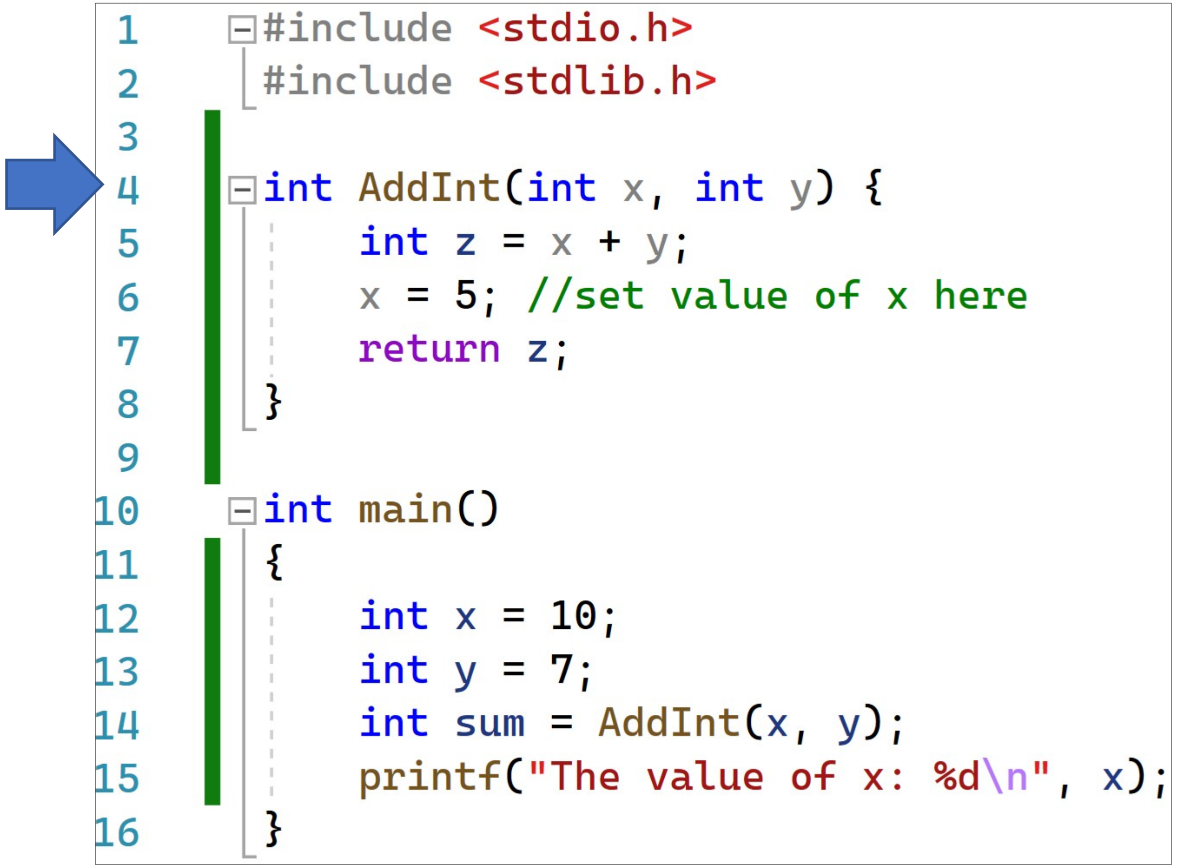
Variables in Main

x =10

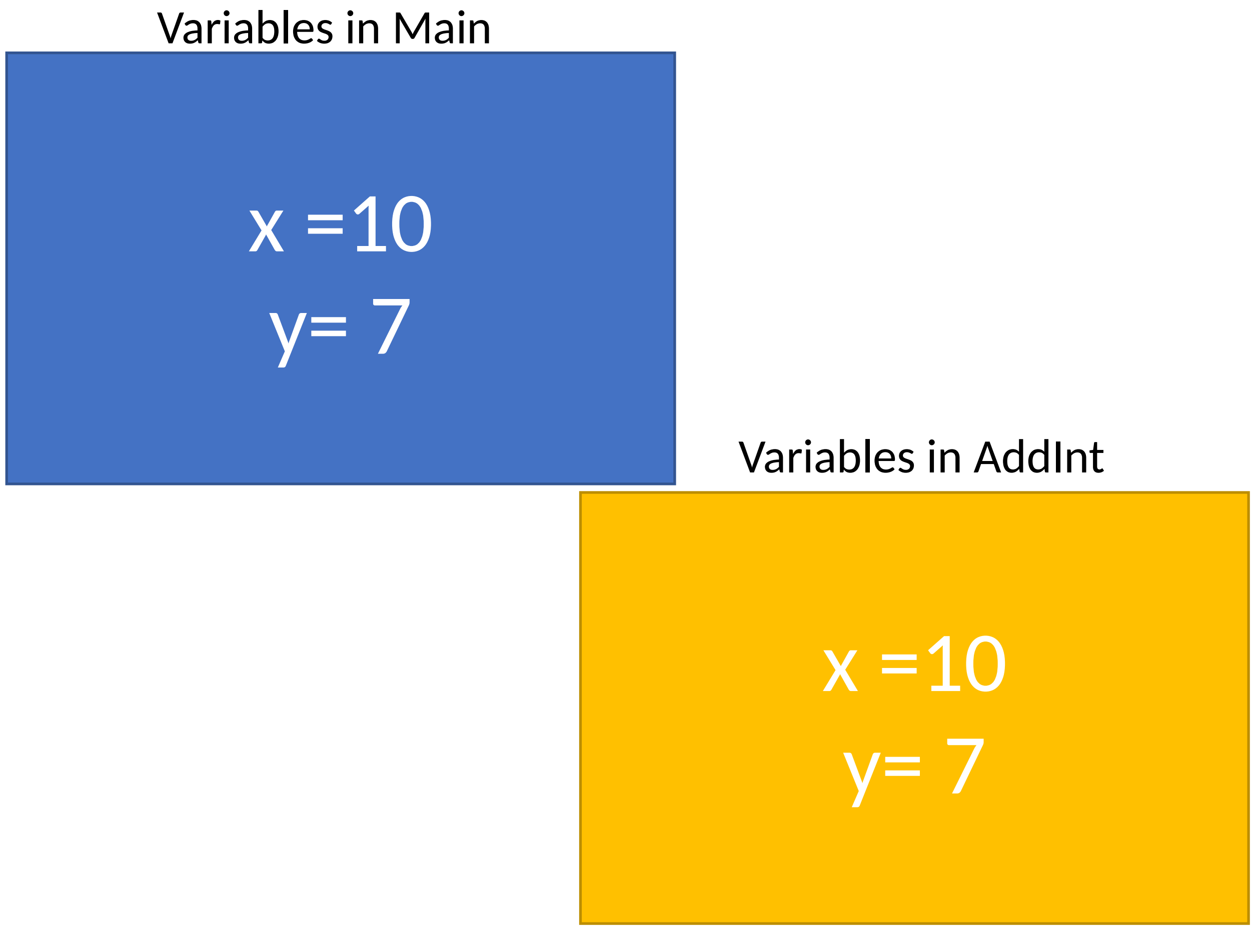
y= 7

Variables in AddInt

What is going on here?



```c
#include <stdio.h>
#include <stdlib.h>

int AddInt(int x, int y) {
    int z = x + y;
    x = 5; //set value of x here
    return z;
}

int main()
{
    int x = 10;
    int y = 7;
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x);
}
```



Variables in Main

x =10

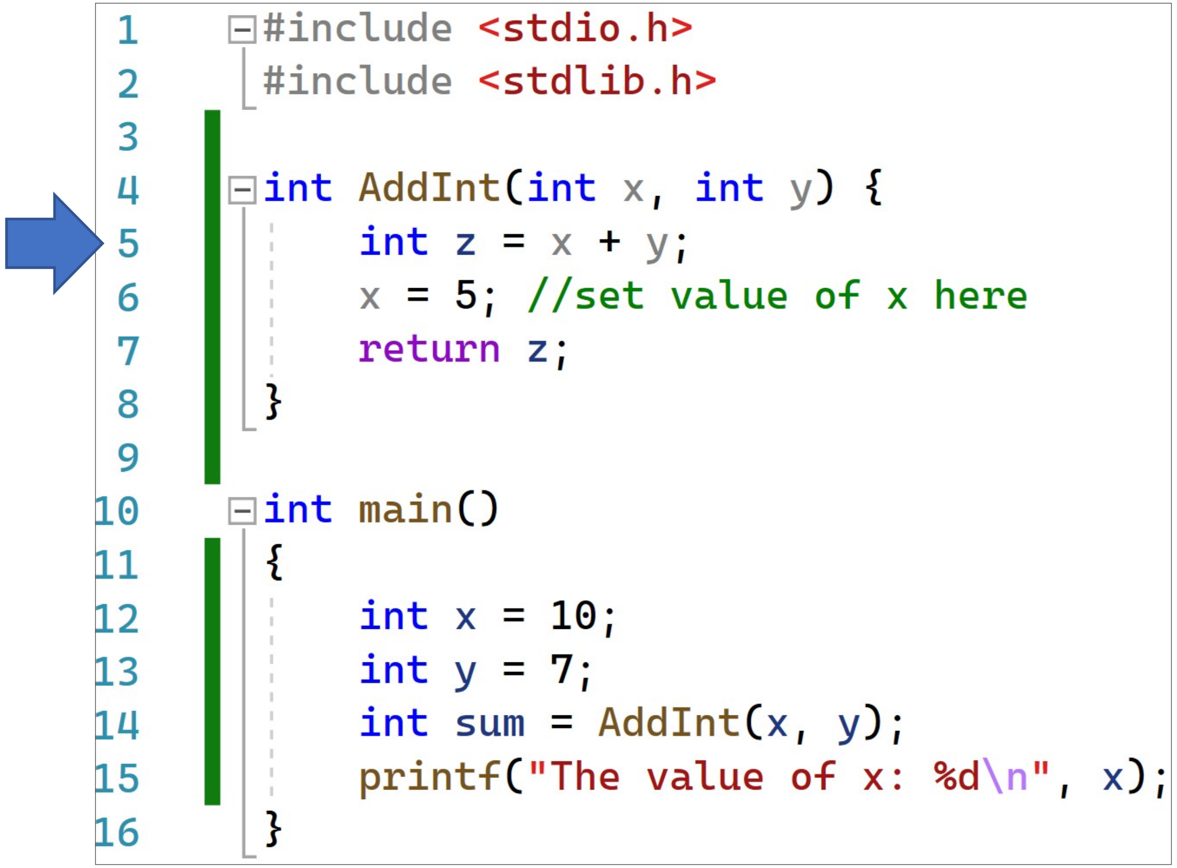
y= 7

Variables in AddInt

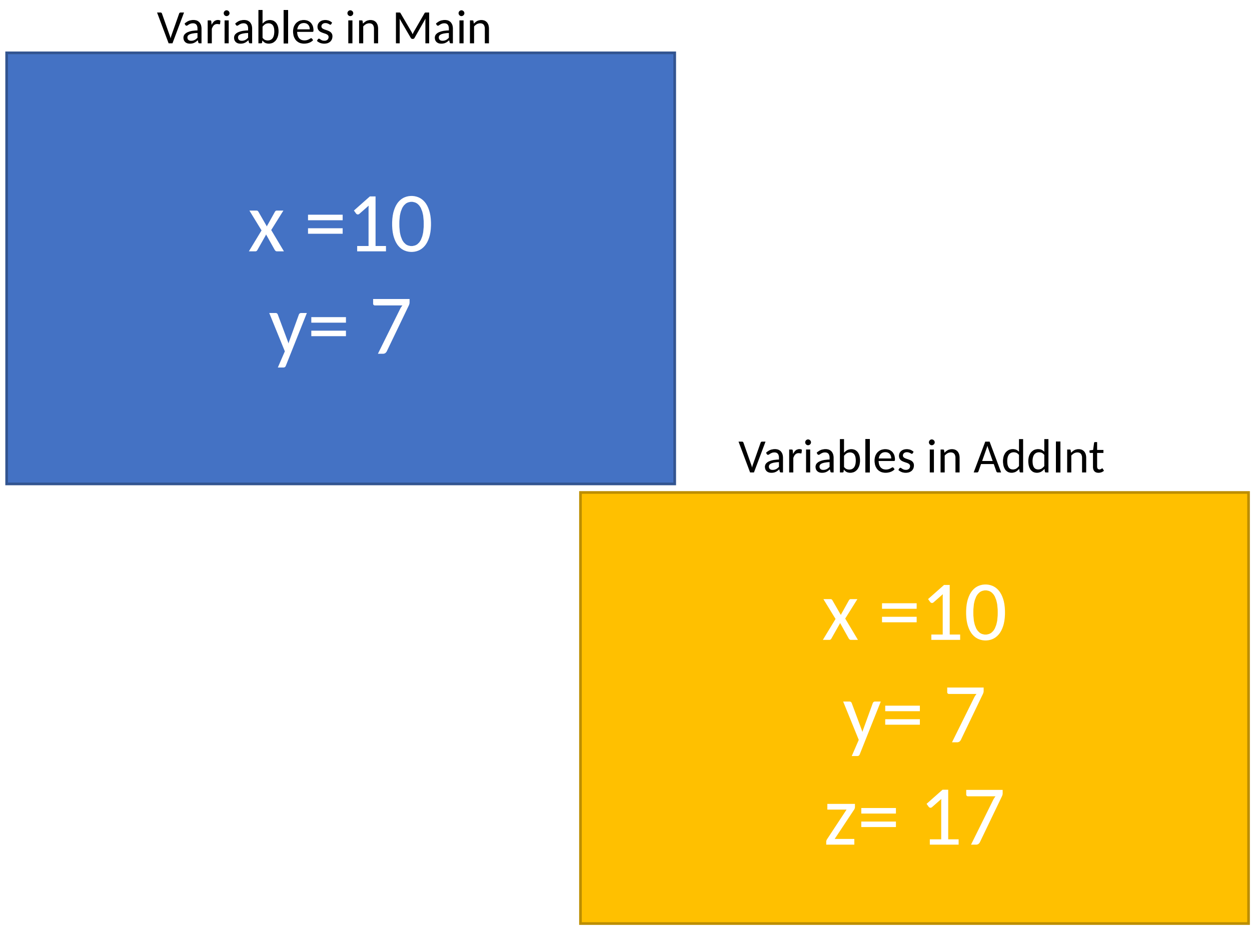
x =10

y= 7

What is going on here?



```c
#include <stdio.h>
#include <stdlib.h>

int AddInt(int x, int y) {
    int z = x + y;
    x = 5; //set value of x here
    return z;
}

int main()
{
    int x = 10;
    int y = 7;
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x);
}
```



Variables in Main

x =10

y= 7

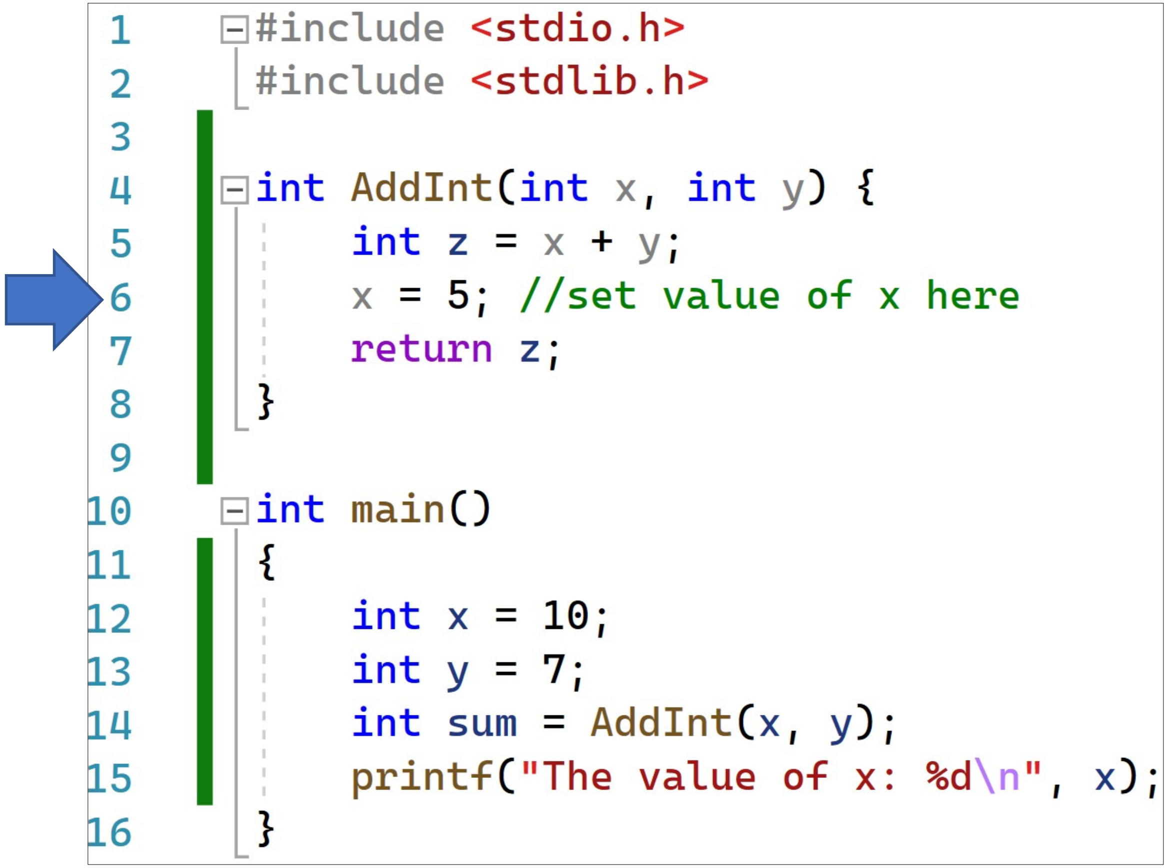
Variables in AddInt

x =10

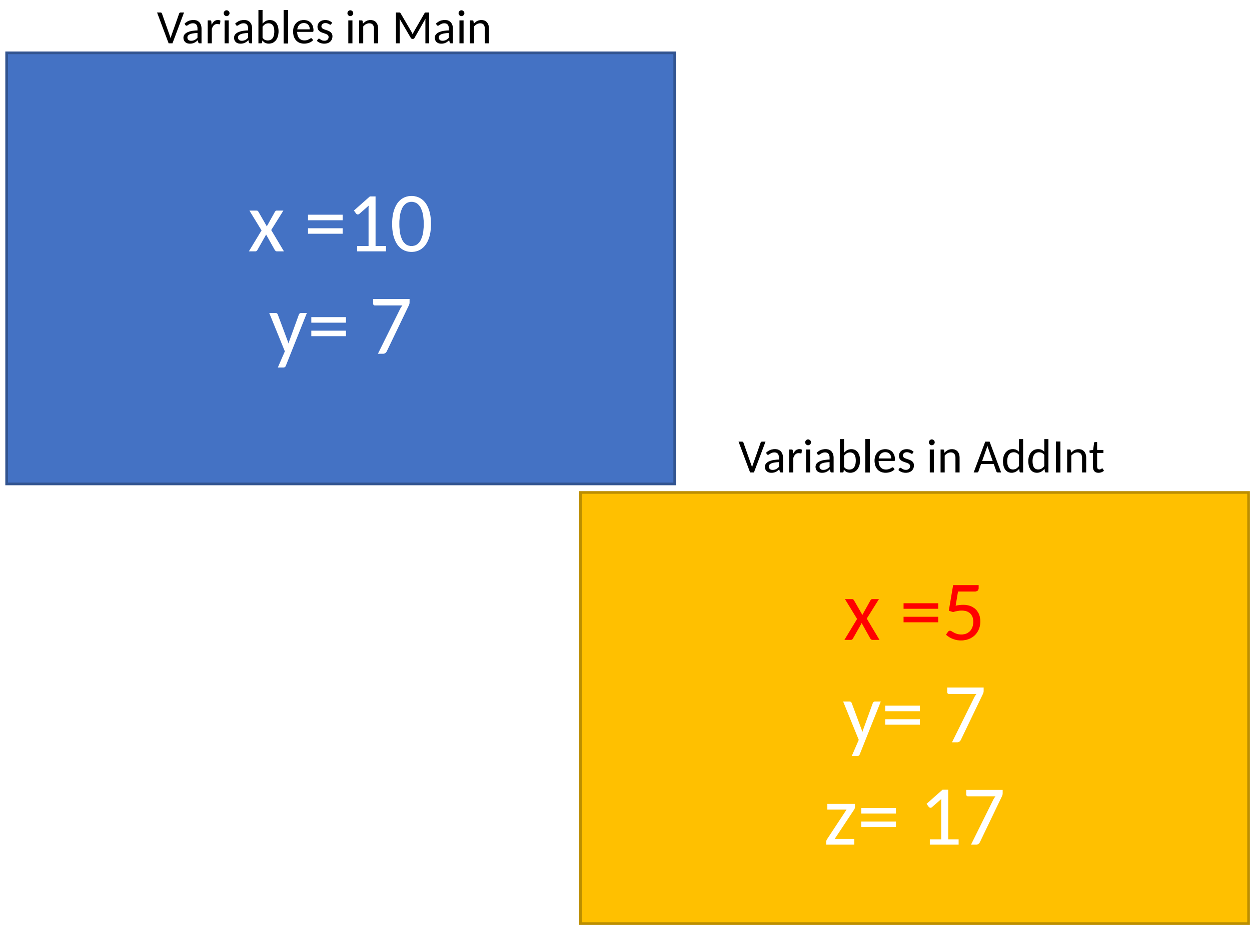
y= 7

z= 17

What is going on here?



```c
#include <stdio.h>
#include <stdlib.h>

int AddInt(int x, int y) {
    int z = x + y;
    x = 5; //set value of x here
    return z;
}

int main()
{
    int x = 10;
    int y = 7;
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x);
}
```



Variables in Main

x =10

y= 7

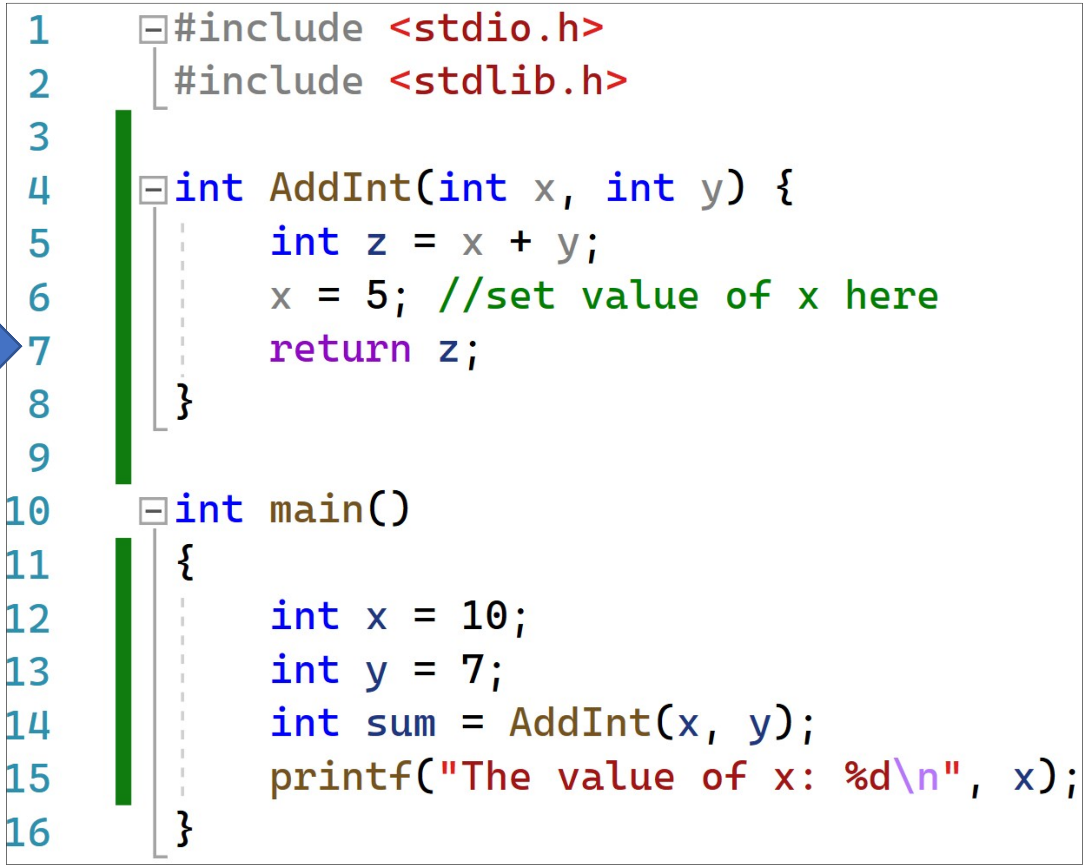
Variables in AddInt

x =5

y= 7

z= 17

What is going on here?



```c
#include <stdio.h>
#include <stdlib.h>

int AddInt(int x, int y) {
    int z = x + y;
    x = 5; //set value of x here
    return z;
}

int main()
{
    int x = 10;
    int y = 7;
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x);
}
```

Variables in Main

x =10

y= 7

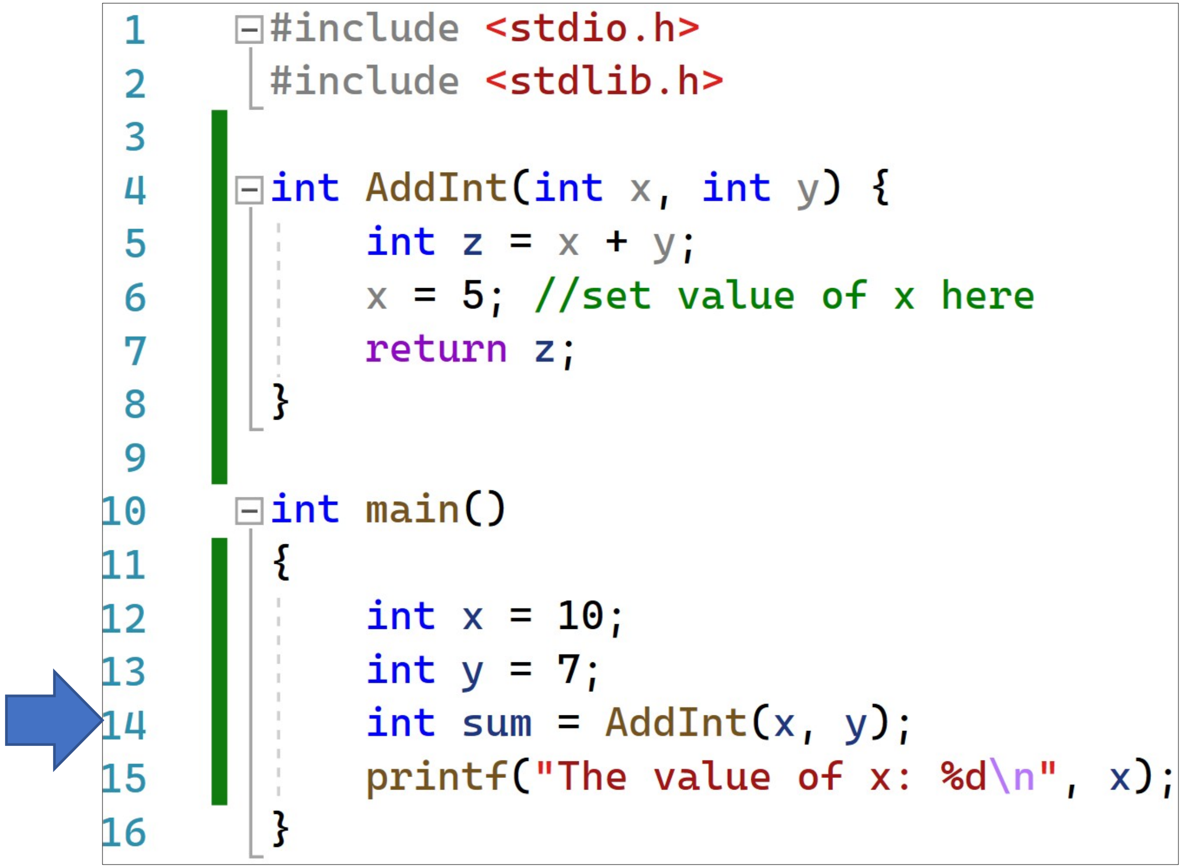
Variables in AddInt

x =5

y= 7

z= 17

What is going on here?



```c
#include <stdio.h>
#include <stdlib.h>

int AddInt(int x, int y) {
    int z = x + y;
    x = 5; //set value of x here
    return z;
}

int main()
{
    int x = 10;
    int y = 7;
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x);
}
```

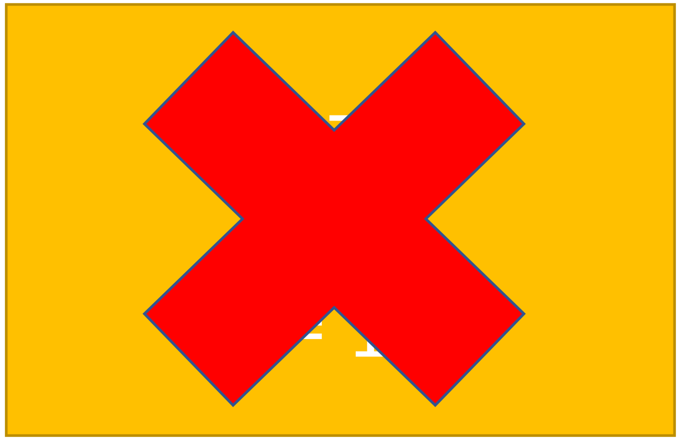
Variables in Main

x =10

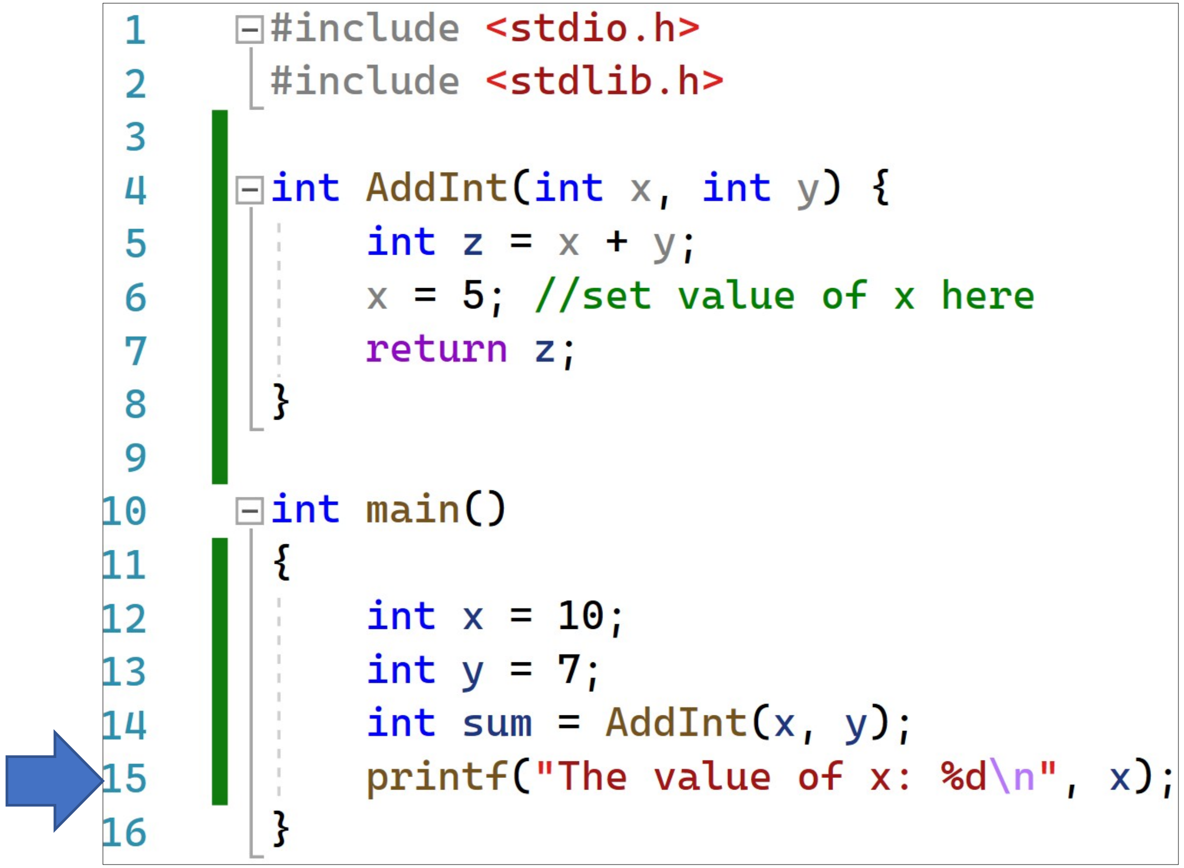
y= 7

sum = 17

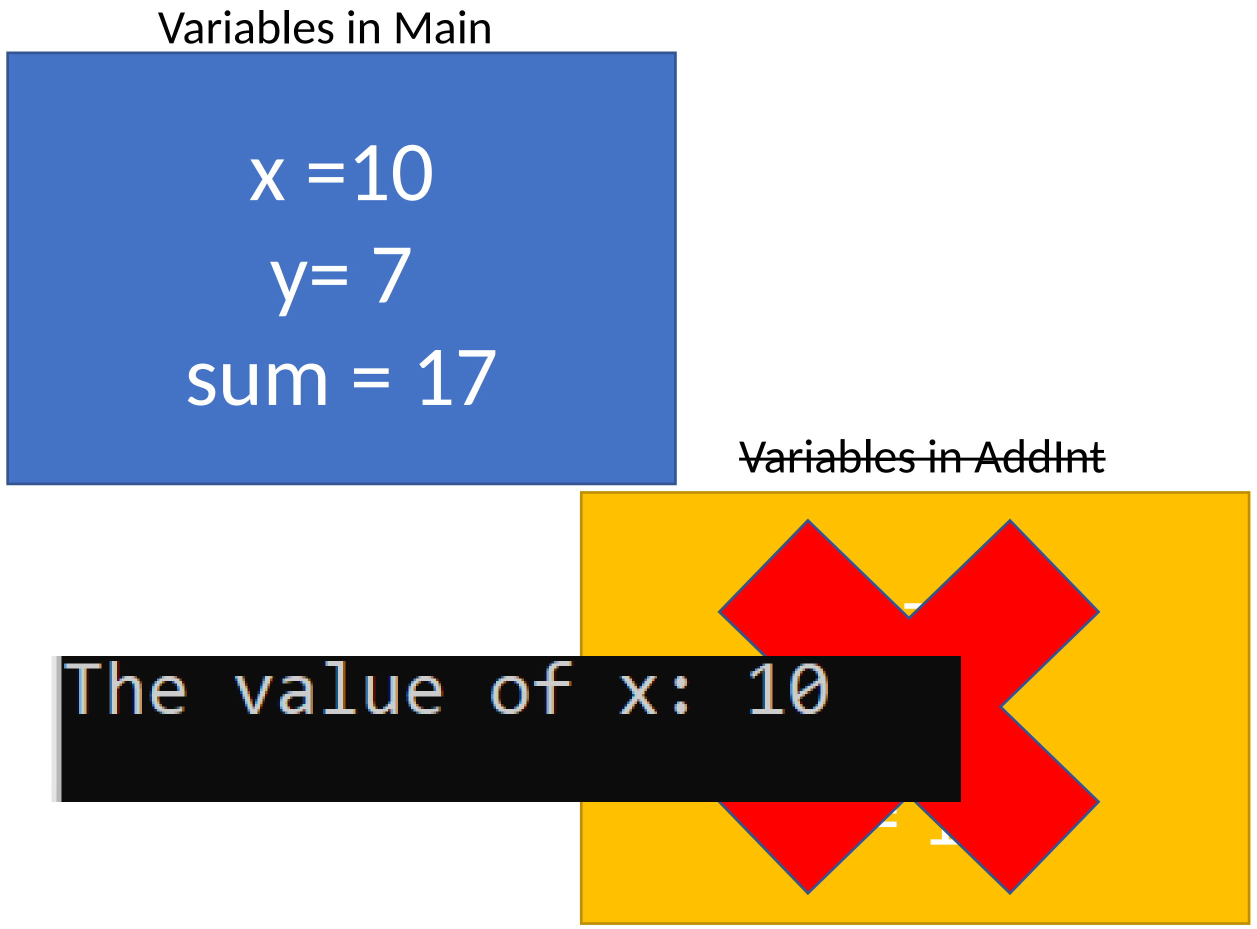
~~Variables in AddInt~~



What is going on here?



```c
#include <stdio.h>
#include <stdlib.h>

int AddInt(int x, int y) {
    int z = x + y;
    x = 5; //set value of x here
    return z;
}

int main()
{
    int x = 10;
    int y = 7;
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x);
}
```



Variables in Main

x =10

y= 7

sum = 17

~~Variables in AddInt~~

```
The value of x: 10
```

Understanding Passing by Value

- Pay careful attention to the previous example.
- In C for PRIMITIVE datatypes, when you pass them to other functions, they are passed as independent copies.
- What is a PRIMITIVE datatype? Int, float, double, long, char…
- Passing by value means that the value is passed to the function, but not the variable itself.
- *What happens for arrays?*

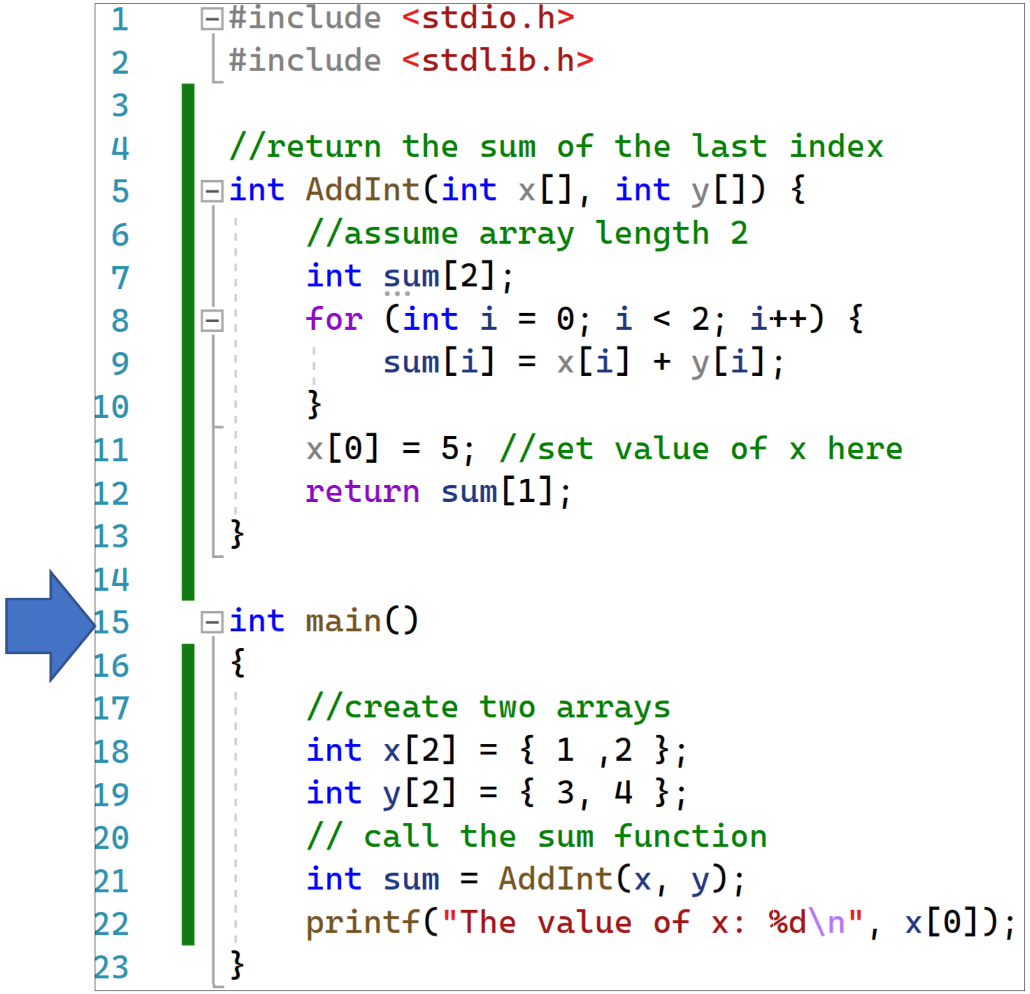
~~1. Arrays in C~~

~~2. Passing By Value~~

3. Passing By Reference

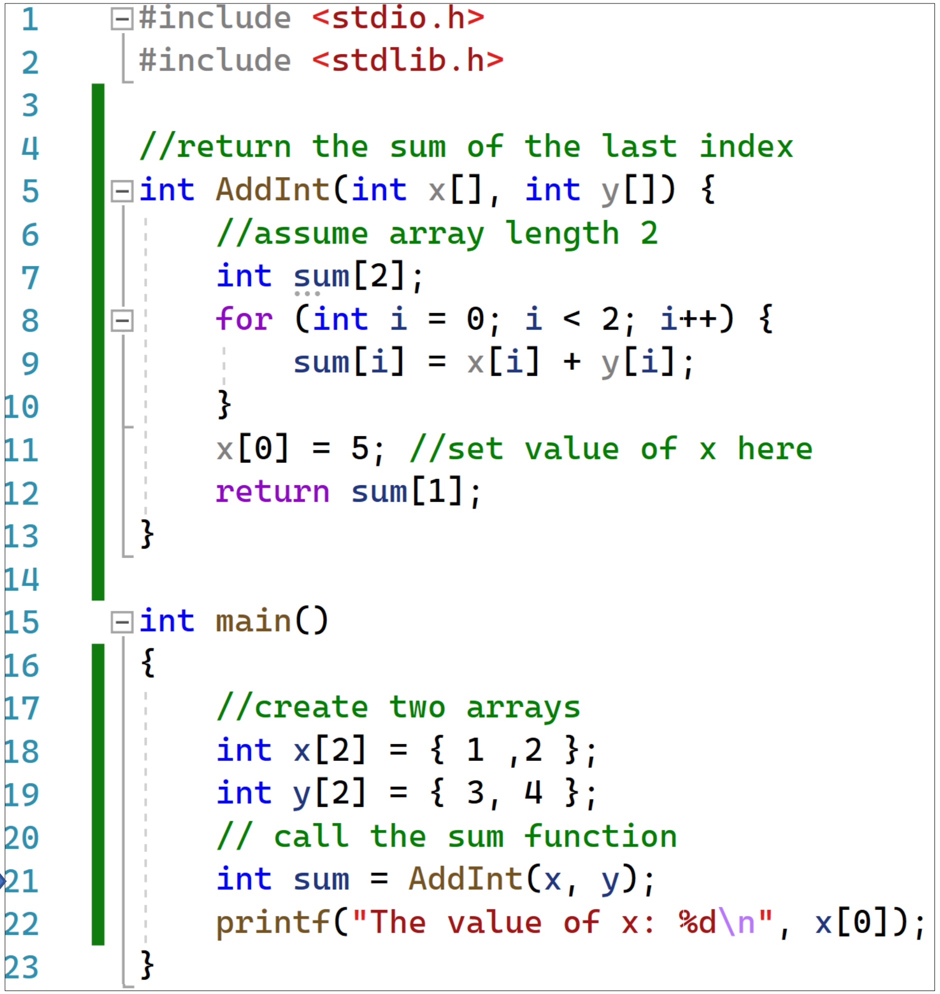
4. Pointers and Memory in C

Passing by reference



```c
#include <stdio.h>
#include <stdlib.h>

//return the sum of the last index
int AddInt(int x[], int y[]) {
    //assume array length 2
    int sum[2];
    for (int i = 0; i < 2; i++) {
        sum[i] = x[i] + y[i];
    }
    x[0] = 5; //set value of x here
    return sum[1];
}

int main()
{
    //create two arrays
    int x[2] = { 1 ,2 };
    int y[2] = { 3, 4 };
    // call the sum function
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x[0]);
}
```

Passing by reference



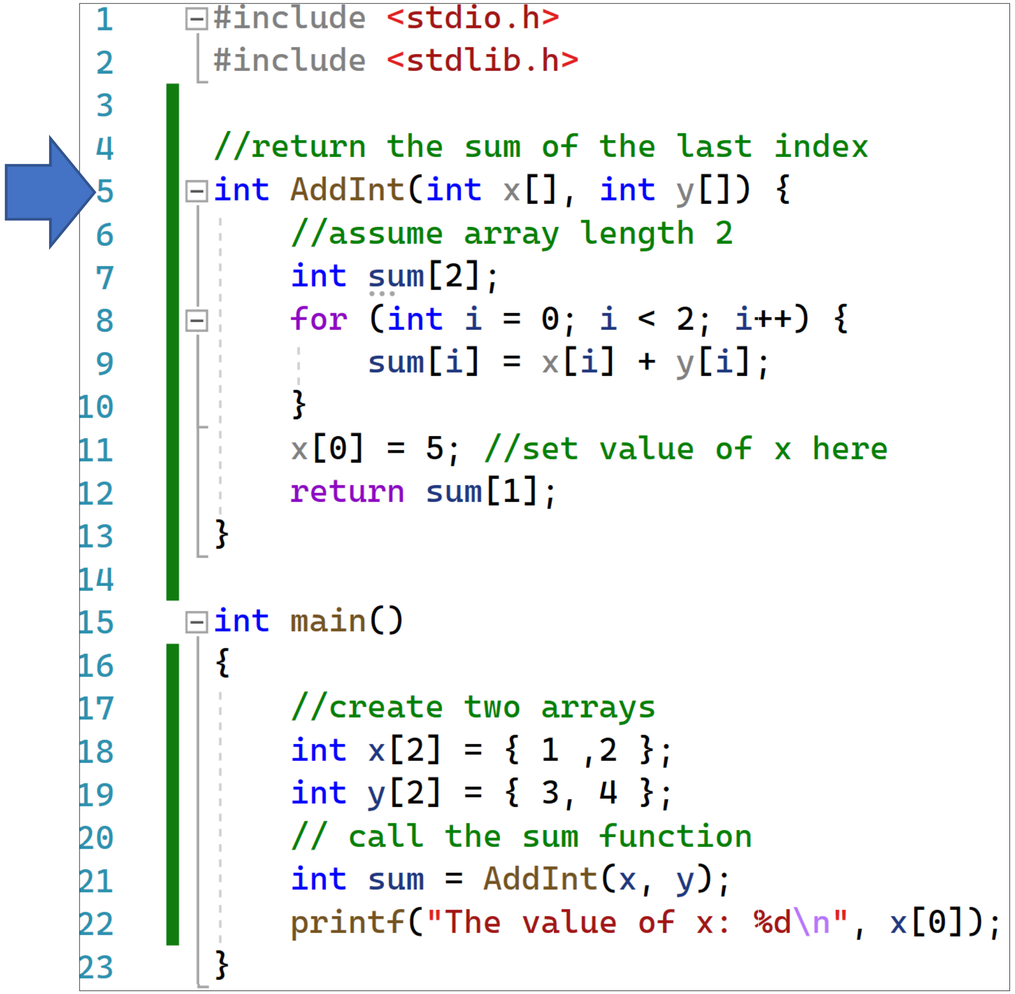
```c
#include <stdio.h>
#include <stdlib.h>

//return the sum of the last index
int AddInt(int x[], int y[]) {
    //assume array length 2
    int sum[2];
    for (int i = 0; i < 2; i++) {
        sum[i] = x[i] + y[i];
    }
    x[0] = 5; //set value of x here
    return sum[1];
}

int main()
{
    //create two arrays
    int x[2] = { 1 ,2 };
    int y[2] = { 3, 4 };
    // call the sum function
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x[0]);
}
```

Variables in Main

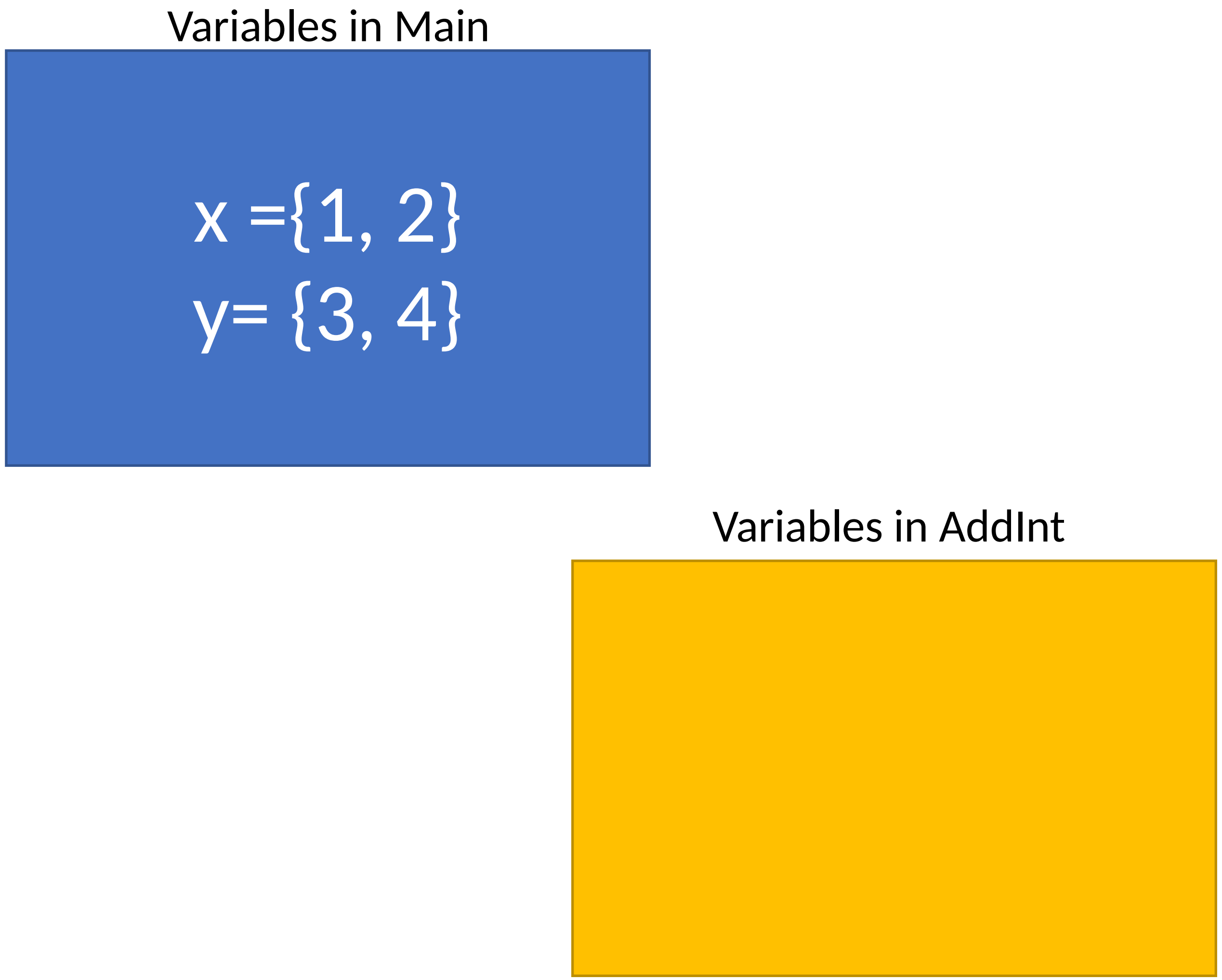
x ={1, 2}

y= {3, 4}

Passing by reference



```
#include <stdio.h>
#include <stdlib.h>

//return the sum of the last index
int AddInt(int x[], int y[]) {
    //assume array length 2
    int sum[2];
    for (int i = 0; i < 2; i++) {
        sum[i] = x[i] + y[i];
    }
    x[0] = 5; //set value of x here
    return sum[1];
}

int main()
{
    //create two arrays
    int x[2] = { 1 ,2 };
    int y[2] = { 3, 4 };
    // call the sum function
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x[0]);
}
```



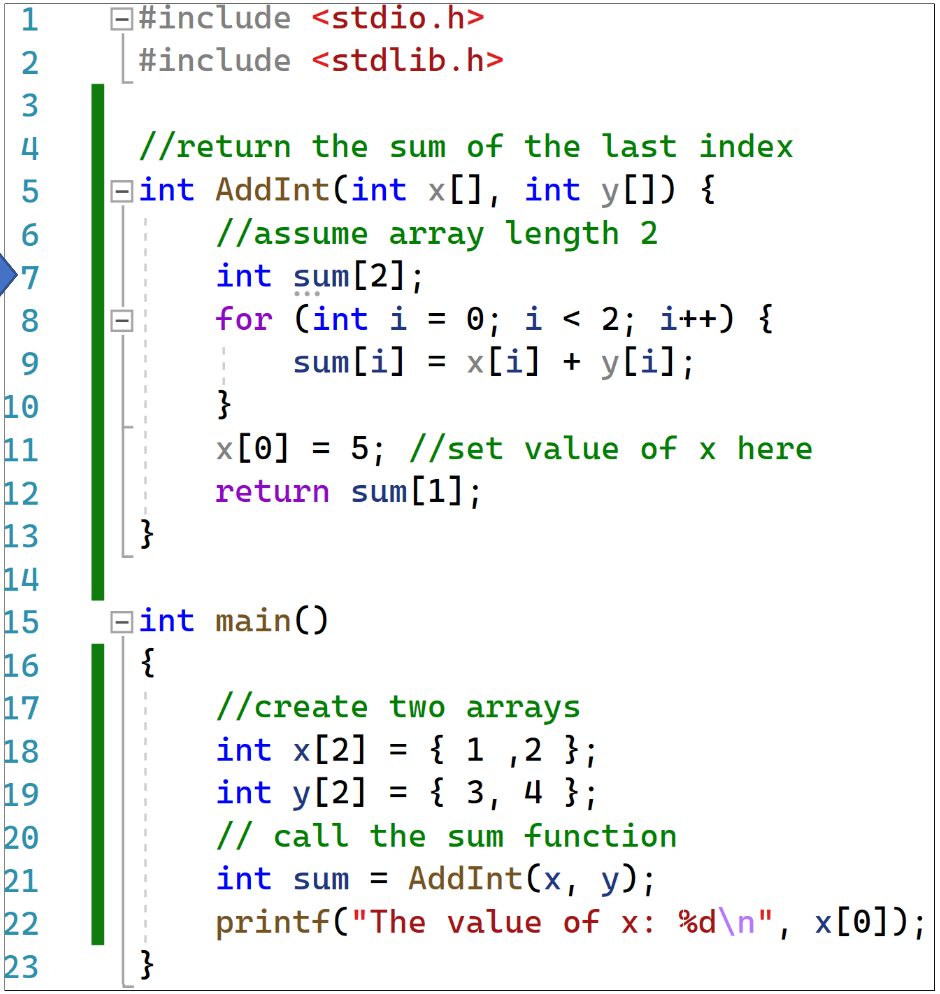
Variables in Main

x ={1, 2}

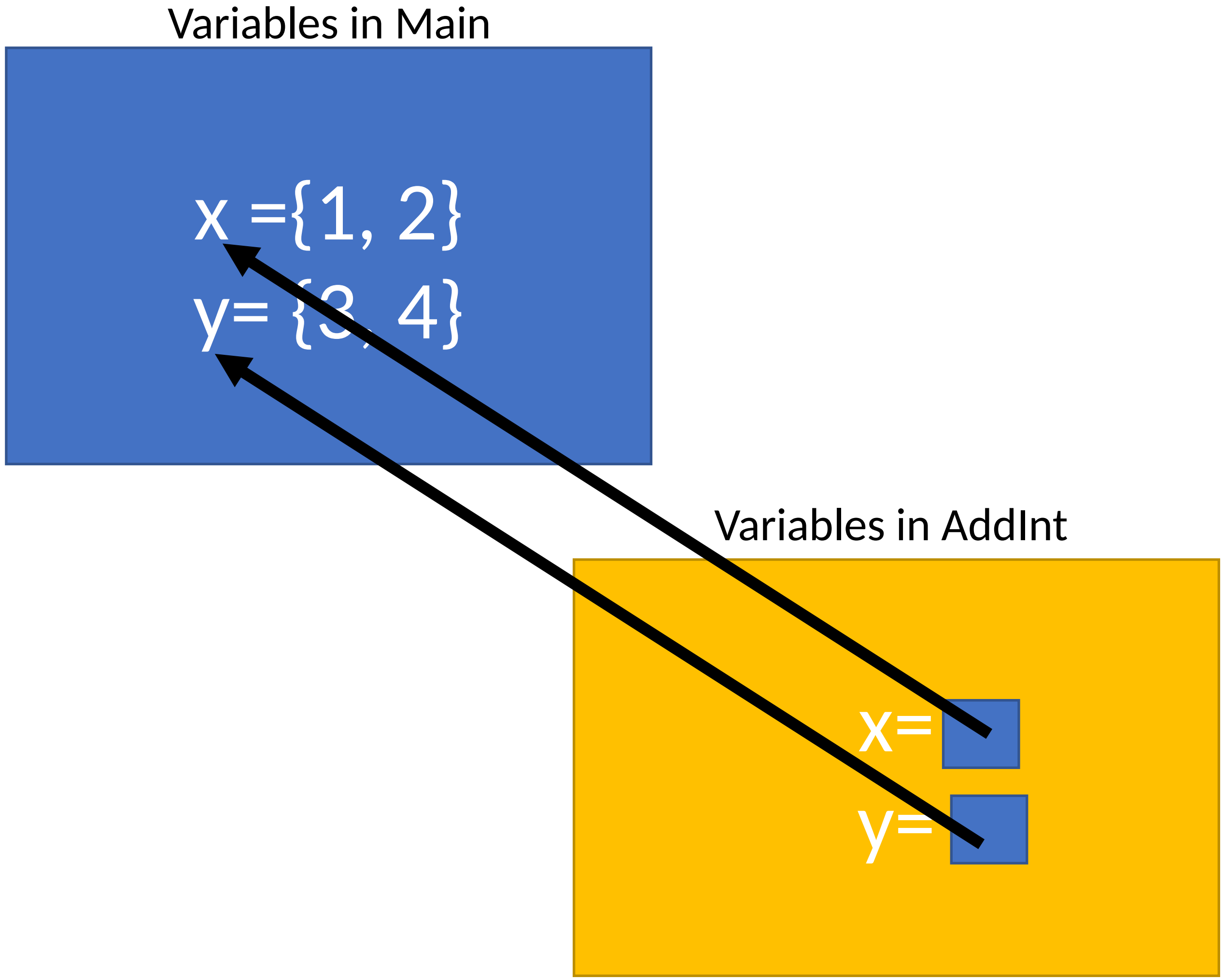
y= {3, 4}

Variables in AddInt

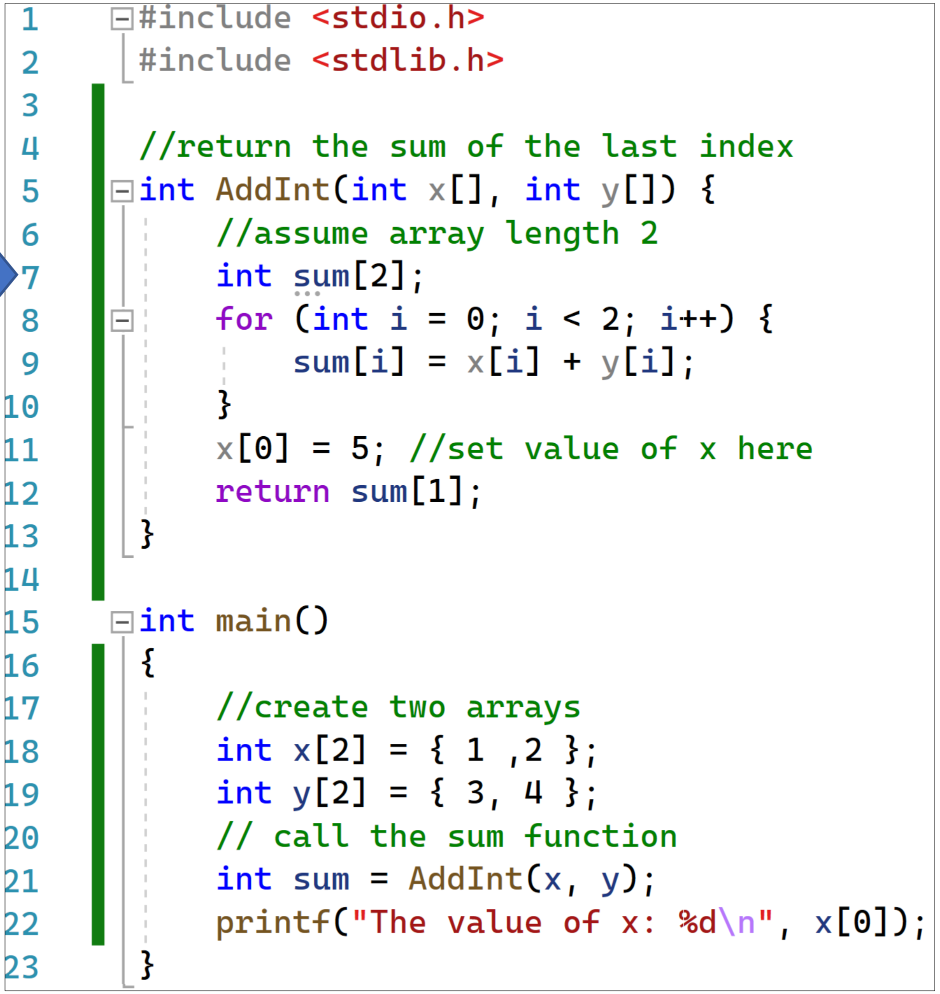
Passing by reference



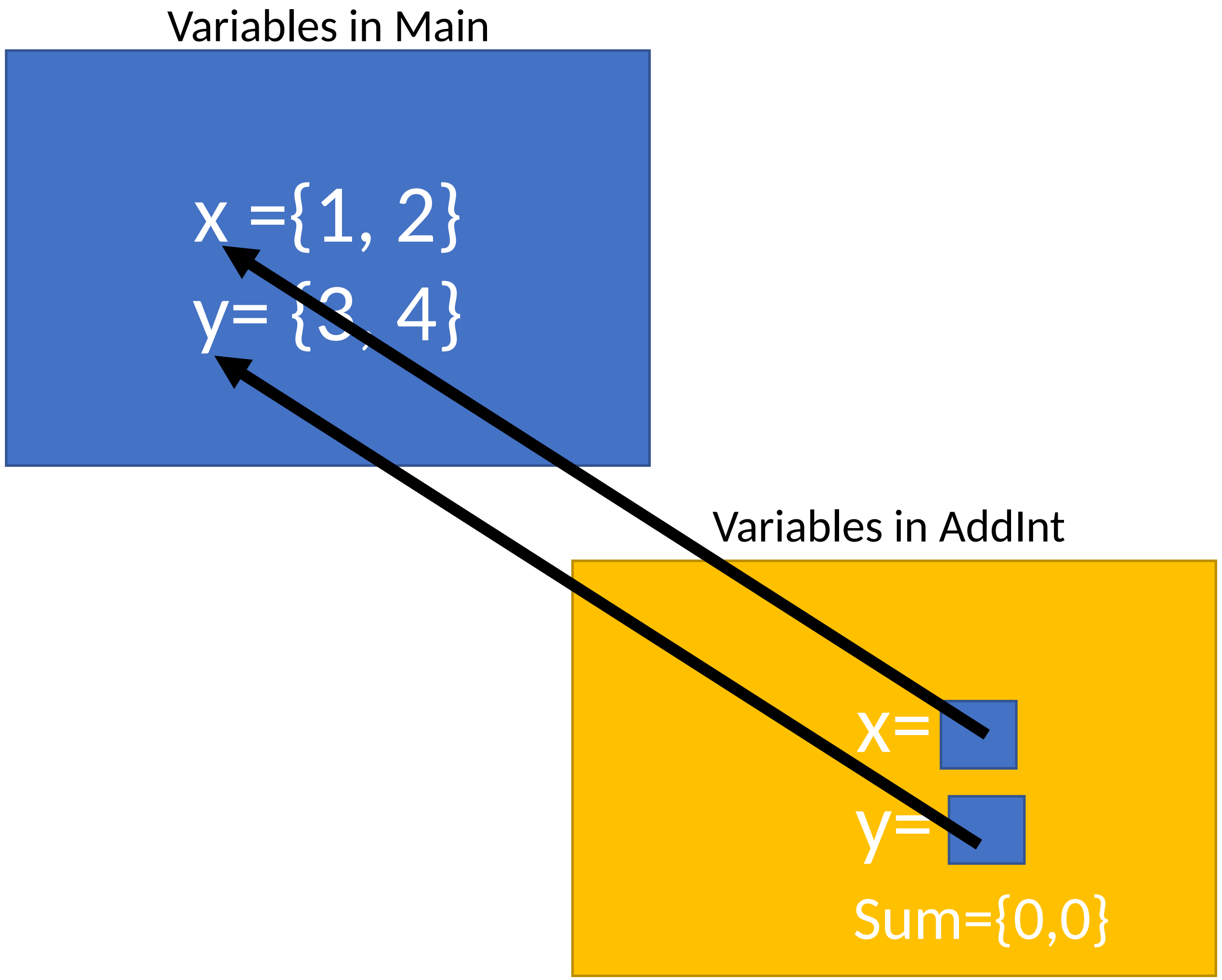
```c
#include <stdio.h>
#include <stdlib.h>

//return the sum of the last index
int AddInt(int x[], int y[]) {
    //assume array length 2
    int sum[2];
    for (int i = 0; i < 2; i++) {
        sum[i] = x[i] + y[i];
    }
    x[0] = 5; //set value of x here
    return sum[1];
}

int main()
{
    //create two arrays
    int x[2] = { 1 ,2 };
    int y[2] = { 3, 4 };
    // call the sum function
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x[0]);
}
```



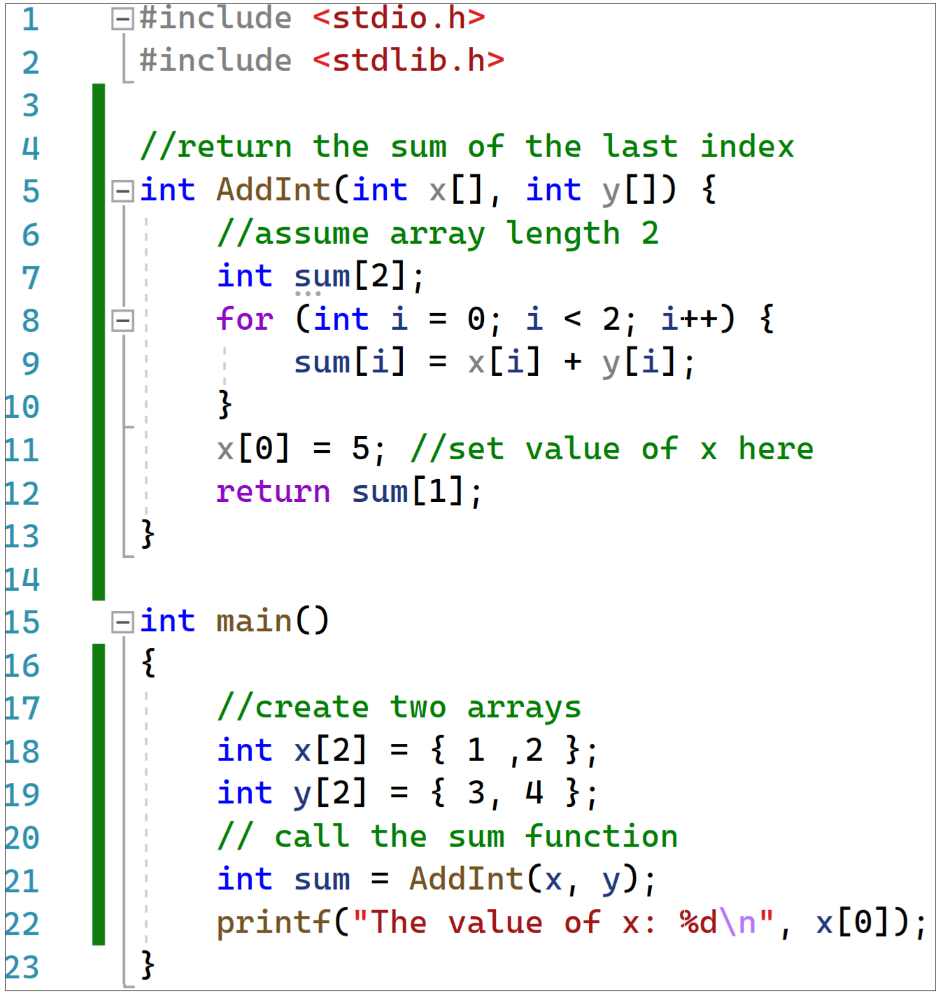
Passing by reference



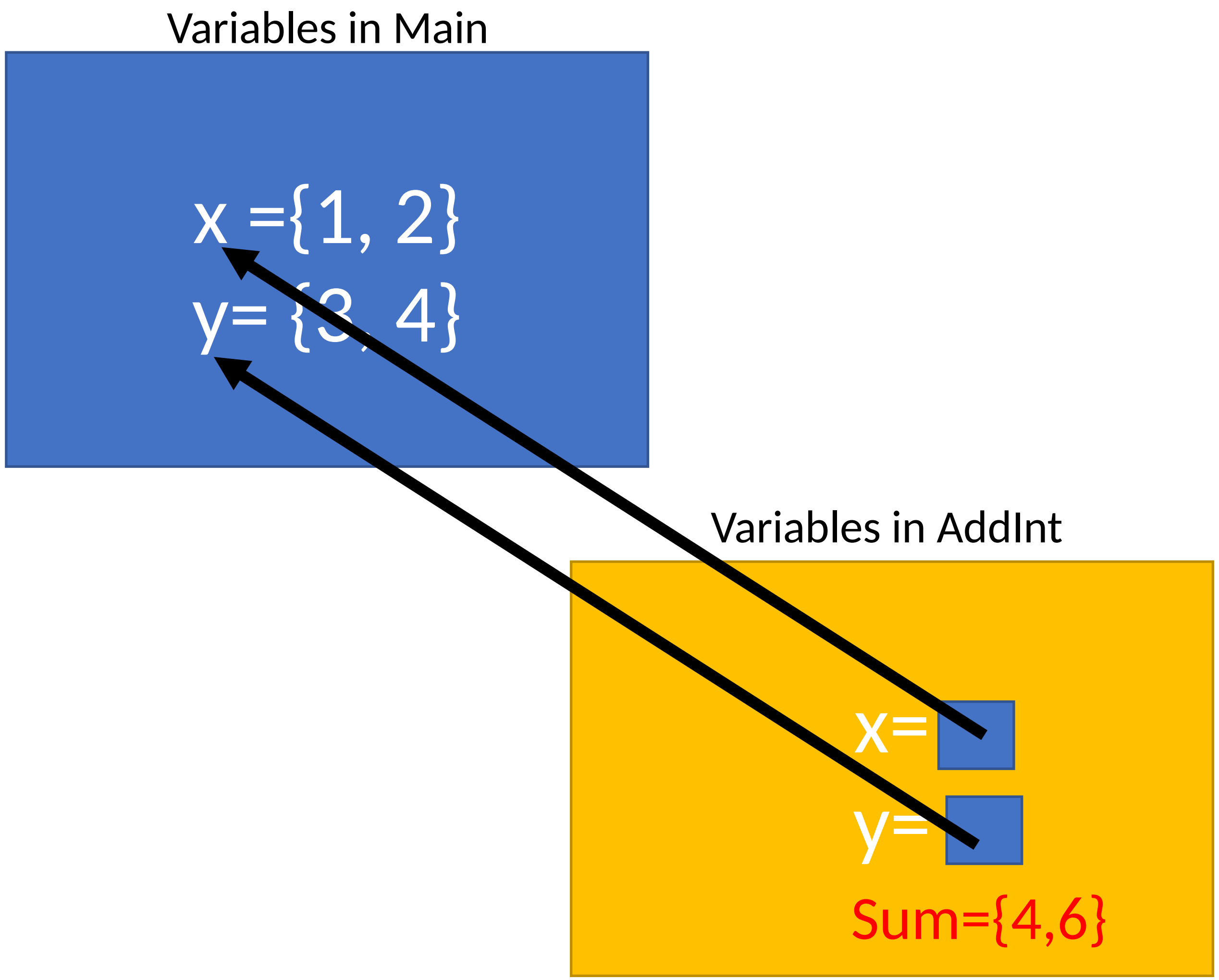
```c
#include <stdio.h>
#include <stdlib.h>

//return the sum of the last index
int AddInt(int x[], int y[]) {
    //assume array length 2
    int sum[2];
    for (int i = 0; i < 2; i++) {
        sum[i] = x[i] + y[i];
    }
    x[0] = 5; //set value of x here
    return sum[1];
}

int main()
{
    //create two arrays
    int x[2] = { 1 ,2 };
    int y[2] = { 3, 4 };
    // call the sum function
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x[0]);
}
```



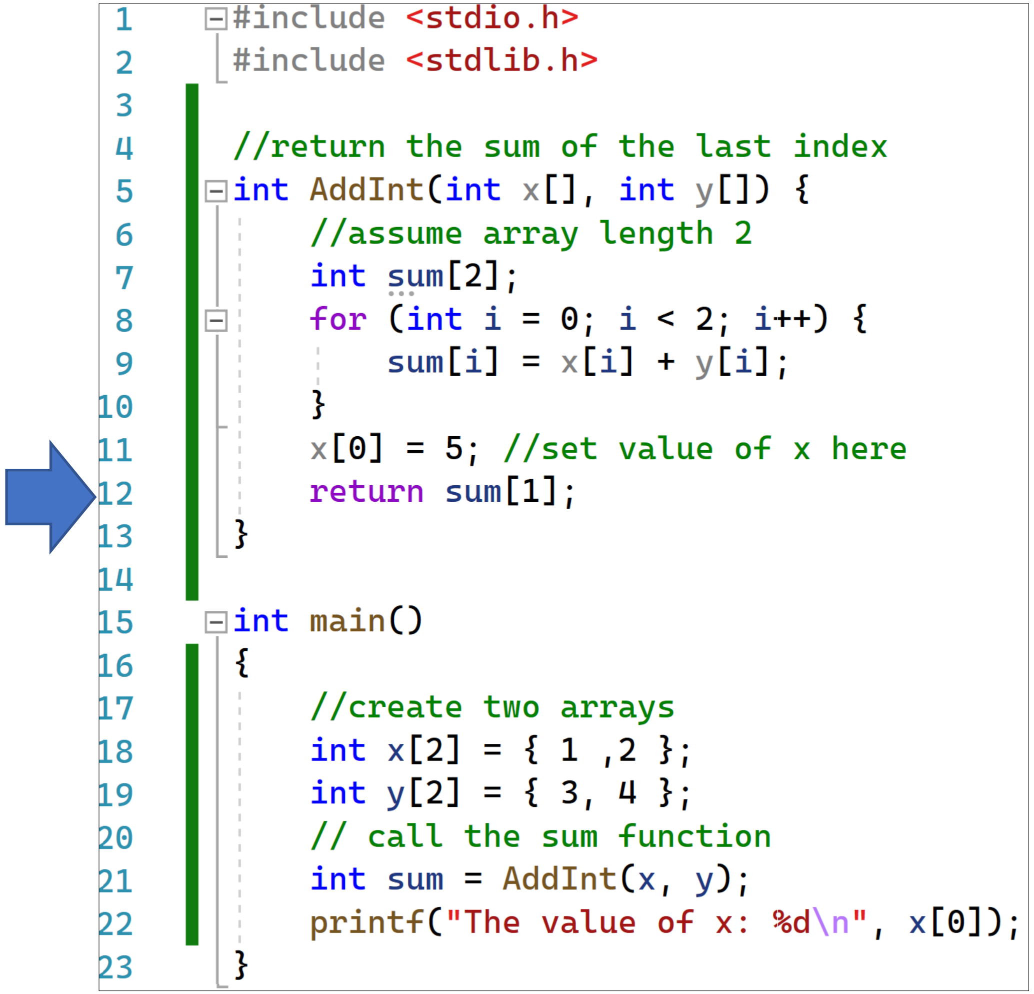
Passing by reference



```c
#include <stdio.h>
#include <stdlib.h>

//return the sum of the last index
int AddInt(int x[], int y[]) {
    //assume array length 2
    int sum[2];
    for (int i = 0; i < 2; i++) {
        sum[i] = x[i] + y[i];
    }
    x[0] = 5; //set value of x here
    return sum[1];
}

int main()
{
    //create two arrays
    int x[2] = { 1 ,2 };
    int y[2] = { 3, 4 };
    // call the sum function
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x[0]);
}
```

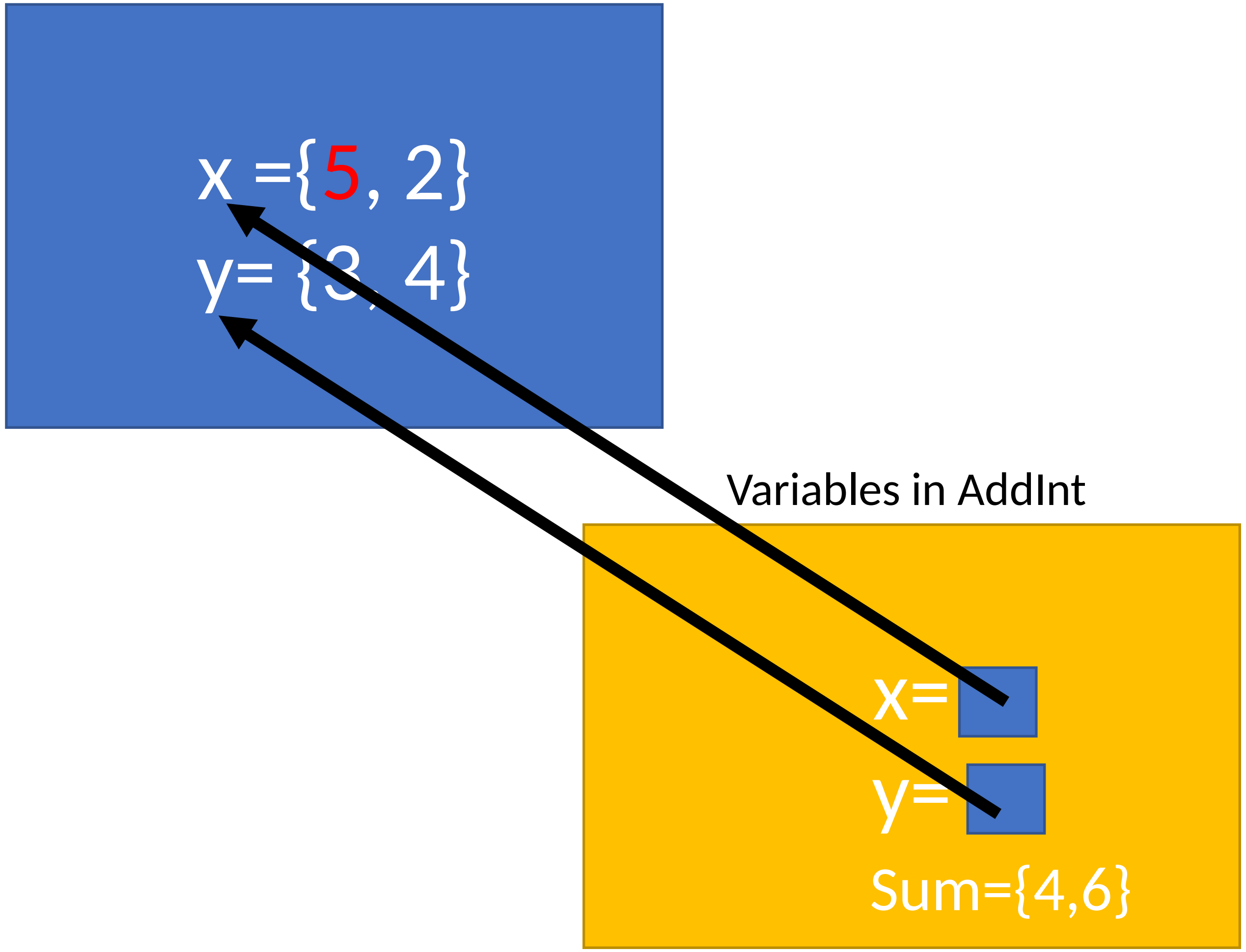


Passing by reference



```
#include <stdio.h>
#include <stdlib.h>

//return the sum of the last index
int AddInt(int x[], int y[]) {
    //assume array length 2
    int sum[2];
    for (int i = 0; i < 2; i++) {
        sum[i] = x[i] + y[i];
    }
    x[0] = 5; //set value of x here
    return sum[1];
}

int main()
{
    //create two arrays
    int x[2] = { 1 ,2 };
    int y[2] = { 3, 4 };
    // call the sum function
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x[0]);
}
```

Variables in Main



x ={5, 2}

y= {3, 4}

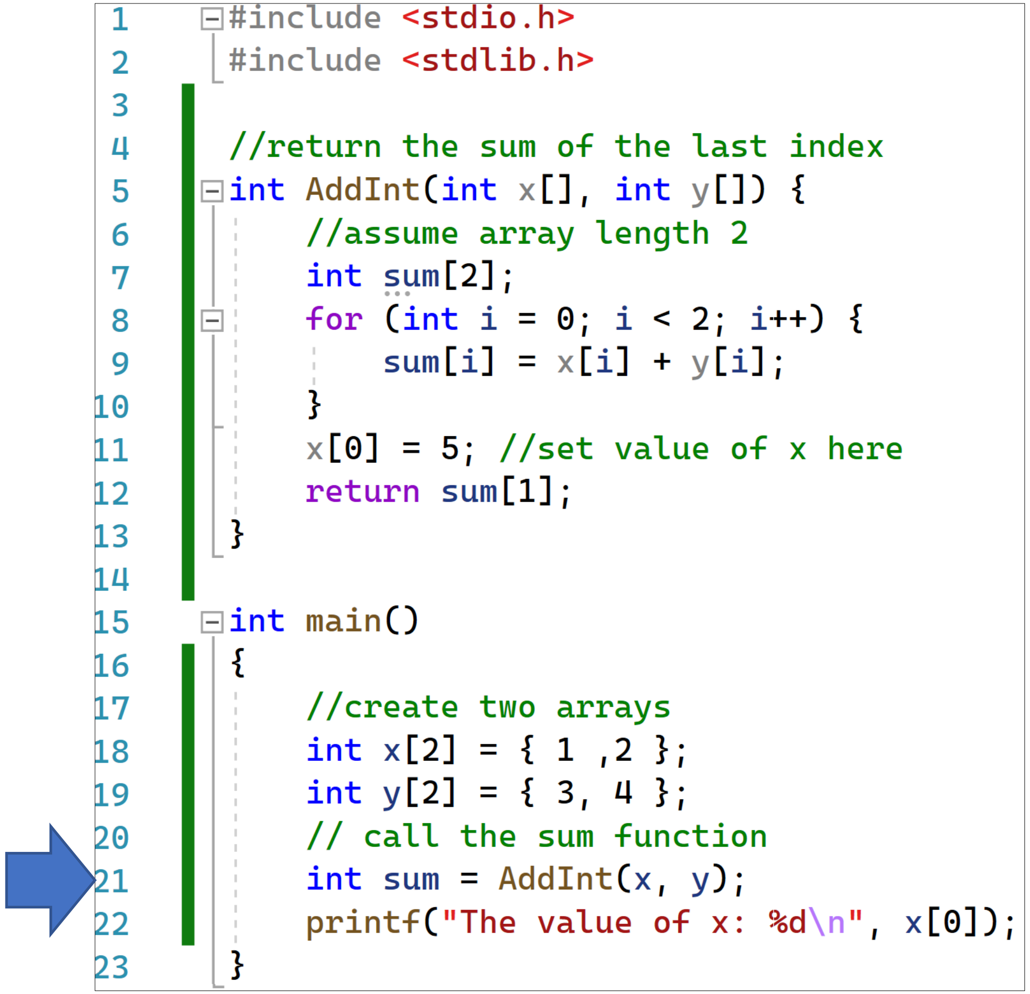
Variables in AddInt

x=

y=

Sum={4,6}

Passing by reference



```c
#include <stdio.h>
#include <stdlib.h>

//return the sum of the last index
int AddInt(int x[], int y[]) {
    //assume array length 2
    int sum[2];
    for (int i = 0; i < 2; i++) {
        sum[i] = x[i] + y[i];
    }
    x[0] = 5; //set value of x here
    return sum[1];
}

int main()
{
    //create two arrays
    int x[2] = { 1 ,2 };
    int y[2] = { 3, 4 };
    // call the sum function
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x[0]);
}
```

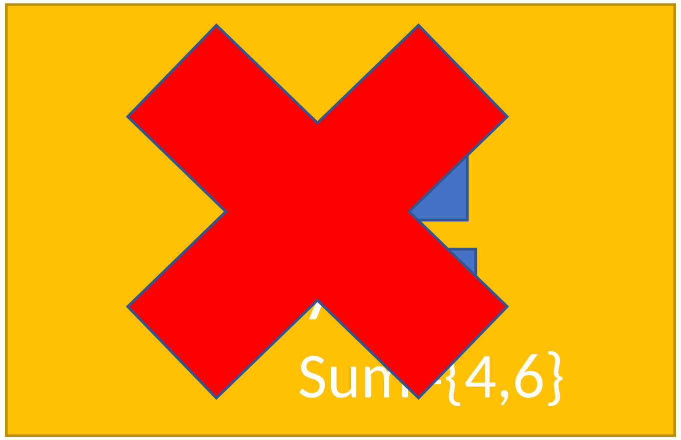
Variables in Main

x ={5, 2}

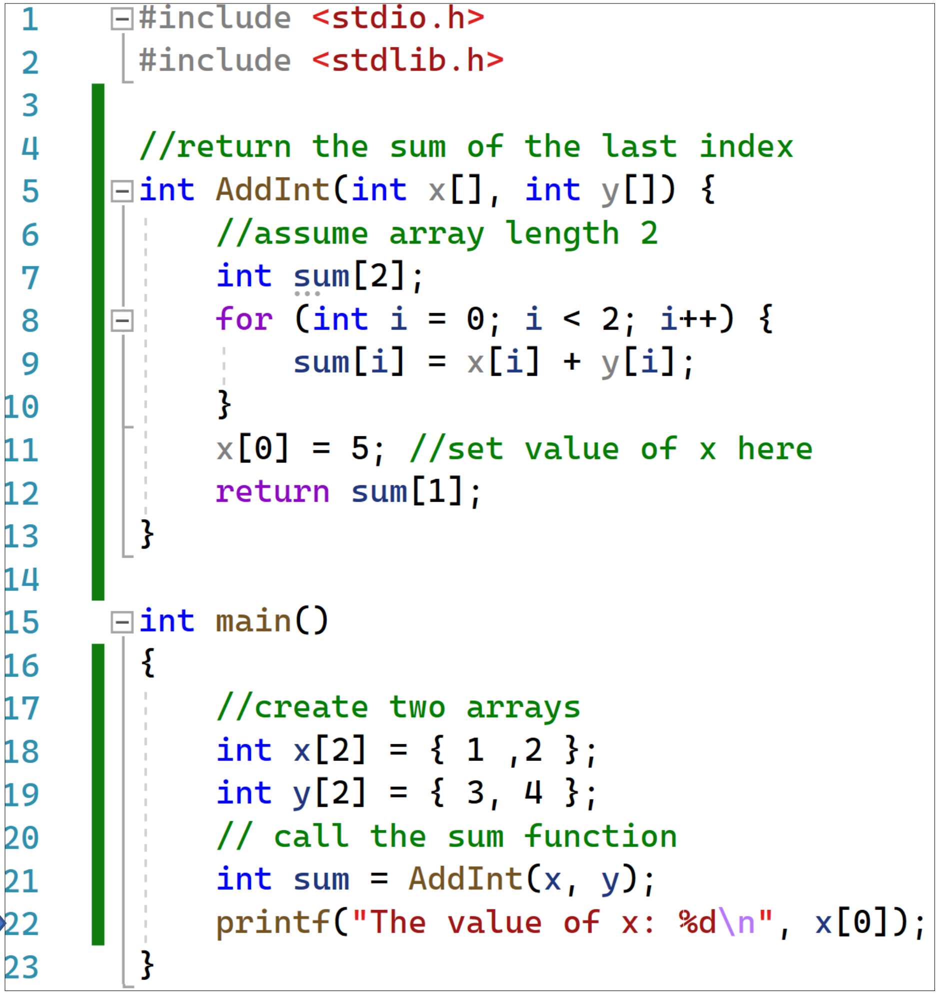
y= {3, 4}

sum =6

~~Variables in AddInt~~



Passing by reference



```c
#include <stdio.h>
#include <stdlib.h>

//return the sum of the last index
int AddInt(int x[], int y[]) {
    //assume array length 2
    int sum[2];
    for (int i = 0; i < 2; i++) {
        sum[i] = x[i] + y[i];
    }
    x[0] = 5; //set value of x here
    return sum[1];
}

int main()
{
    //create two arrays
    int x[2] = { 1 ,2 };
    int y[2] = { 3, 4 };
    // call the sum function
    int sum = AddInt(x, y);
    printf("The value of x: %d\n", x[0]);
}
```

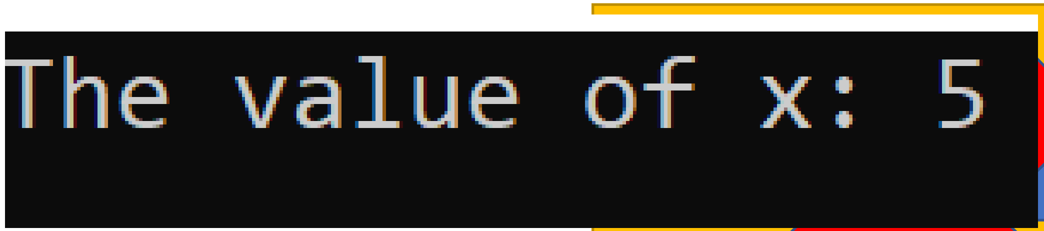
Variables in Main

x ={5, 2}

y= {3, 4}

sum =6

~~Variables in AddInt~~



```
The value of x: 5
```

Sum={4,6}

Understanding Passing by Reference

- Pay careful attention to the previous example.
- Arrays are NOT PRIMITIVE datatypes.
- When you pass arrays to a method you DO NOT get an independent copy. You get a reference to where the original array is stored in memory.
- A bit confused?

Why did the film Inception have this iconic quote?

- There are two possibilities:

1. They were talking about dreams (unlikely).

2. They were talking about understanding pointers and memory in C programming.

~~1. Arrays in C~~

~~2. Passing By Value~~

~~3. Passing By Reference~~

4. Pointers and Memory in C

Pointers and Memory in C

- Let’s talk about where you live.
- Assume you have a house.
- Where is the house?
- At an address.
- When discussing where you live two pieces of information are important.
- **The address** of your house, and **the house** itself.

Pointers and Memory in C

- For C programming two things are important: *The value of a variable* and *where that variable lives in memory*.

- Your house = A variables value
- House Address = the address in computer memory

- In C we call a reference to the address in computer memory **a pointer**.

Variables and Memory

- The memory is an array of bytes
- Every byte in memory is numbered: the address!
  - An address is just an unsigned integer
- Every variable is kept in memory, and is associated with two numbers:

-The address

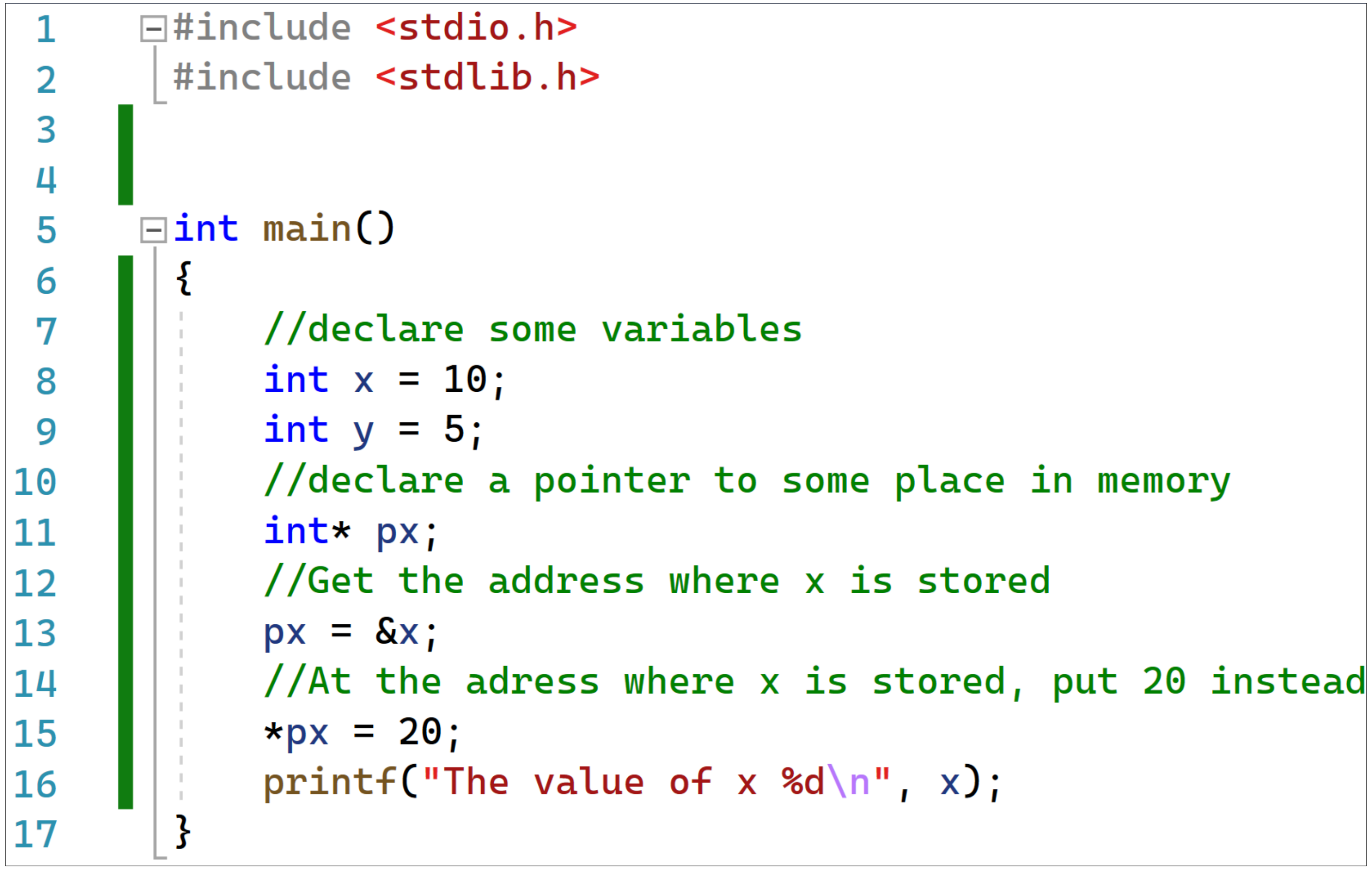
-The value stored at that address

Referencing and dereferencing

- Two new operators

```
    &       Reference: "get" the address of something
    *       Dereference: "use" the address
```

Pointer Example using “&”

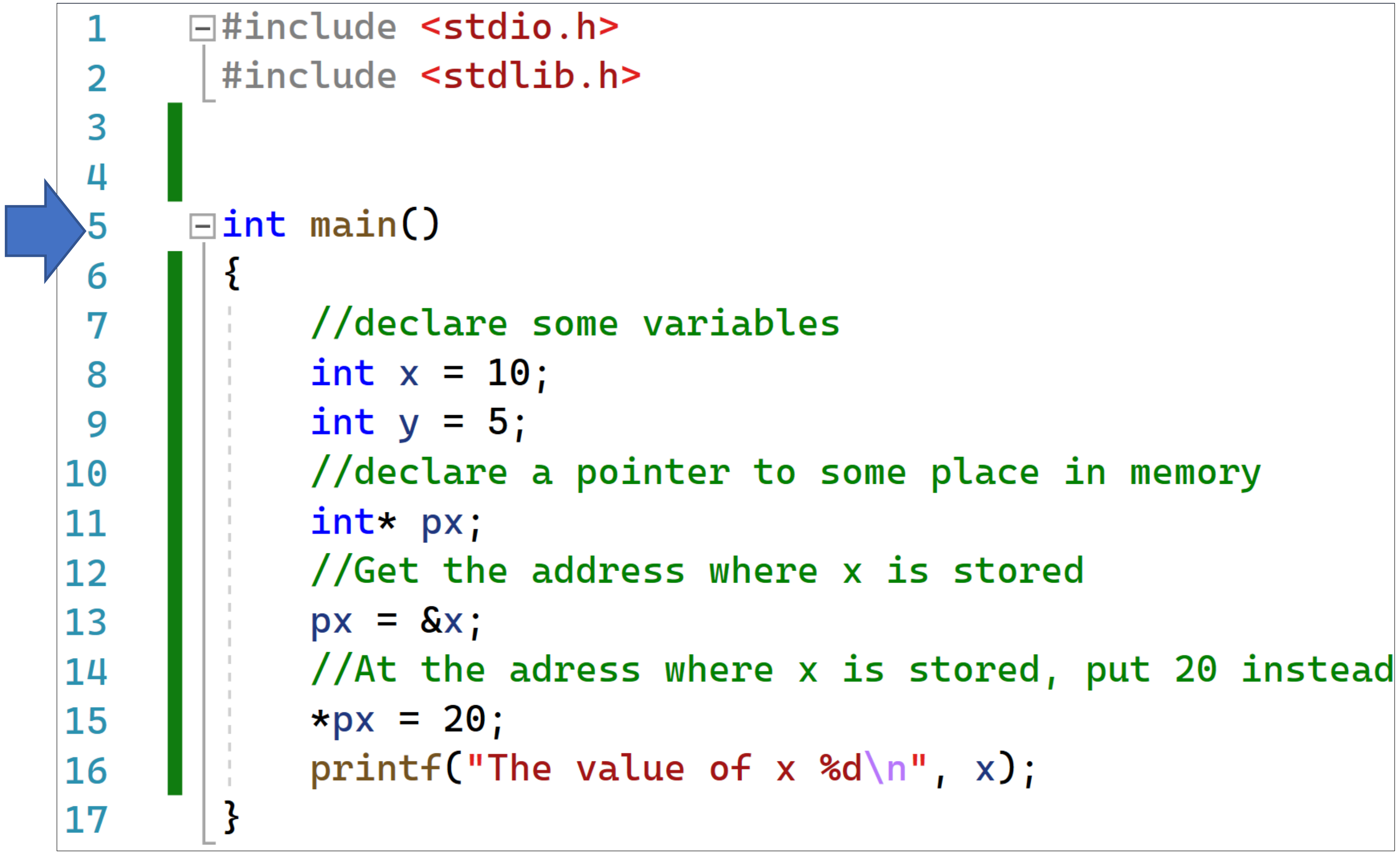


```c
#include <stdio.h>
#include <stdlib.h>


int main()
{
    //declare some variables
    int x = 10;
    int y = 5;
    //declare a pointer to some place in memory
    int* px;
    //Get the address where x is stored
    px = &x;
    //At the adress where x is stored, put 20 instead
    *px = 20;
    printf("The value of x %d\n", x);
}
```

Computer Memory

Address	Value
1028	
1024	
1020	
1016	
1012	
1008	
1004	
1000	

Pointer Example using “&”

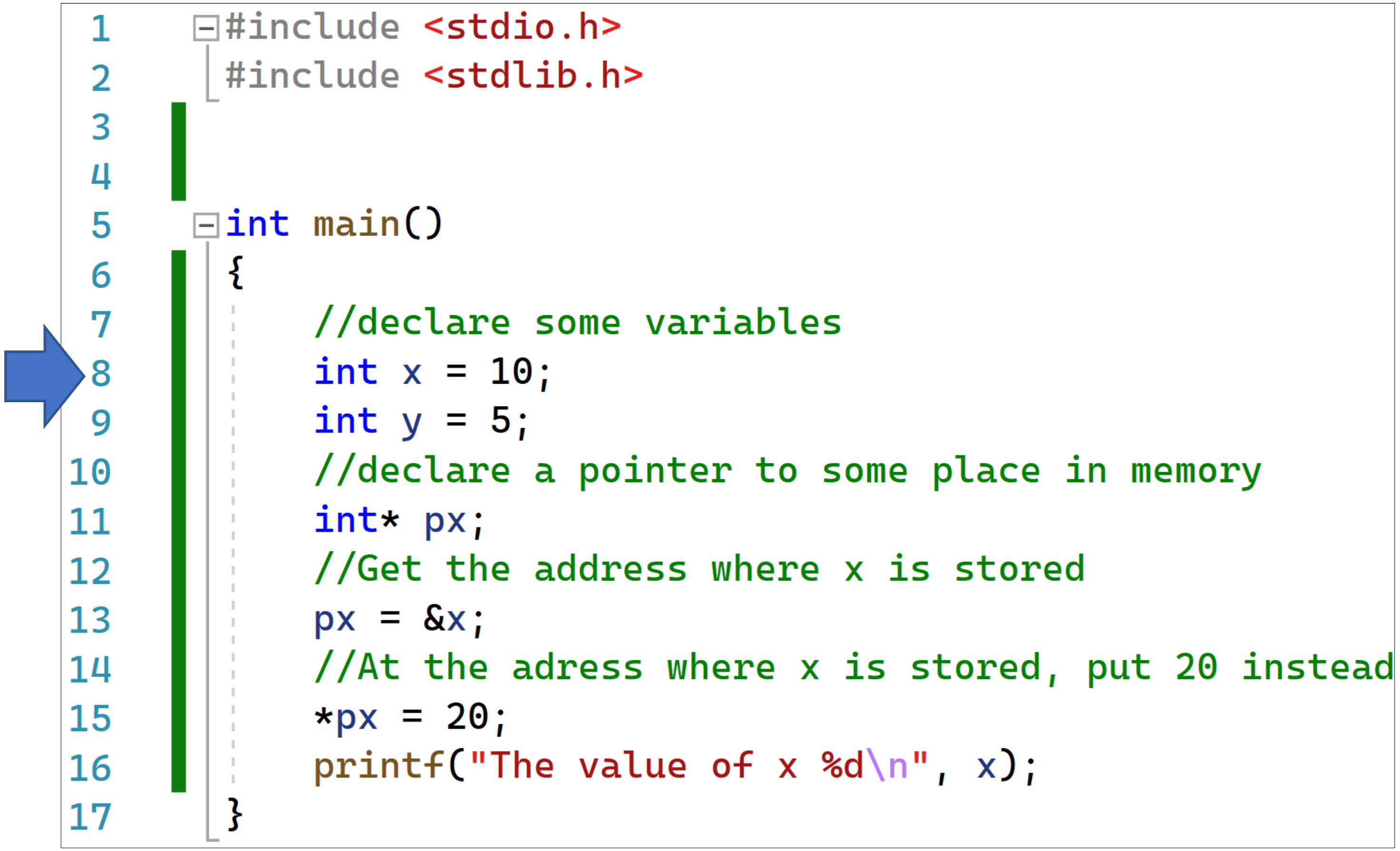


```c
#include <stdio.h>
#include <stdlib.h>


int main()
{
    //declare some variables
    int x = 10;
    int y = 5;
    //declare a pointer to some place in memory
    int* px;
    //Get the address where x is stored
    px = &x;
    //At the adress where x is stored, put 20 instead
    *px = 20;
    printf("The value of x %d\n", x);
}
```

Computer Memory

Address	Value
1028	
1024	
1020	
1016	
1012	
1008	
1004	
1000	

Pointer Example using “&”

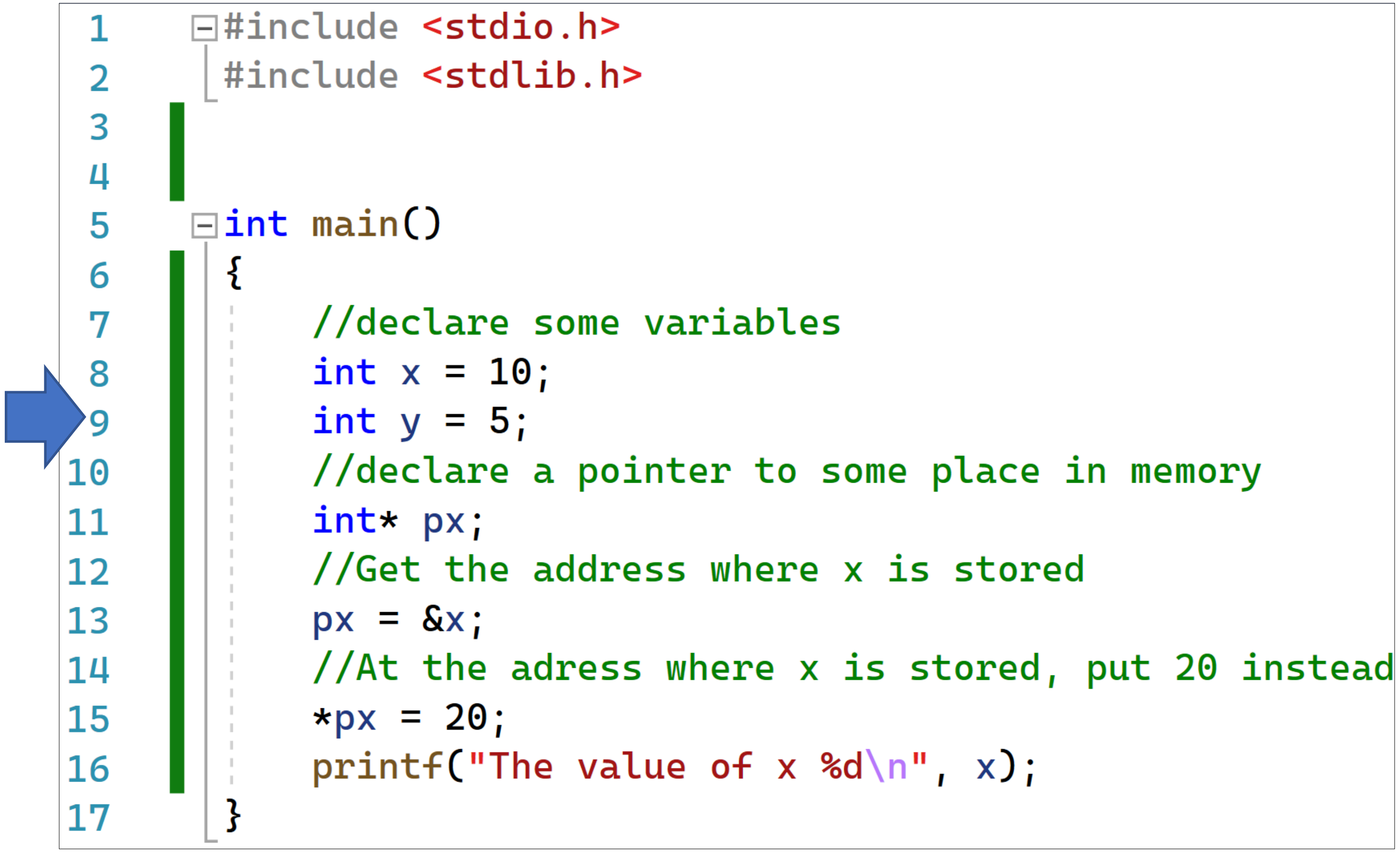


```c
#include <stdio.h>
#include <stdlib.h>


int main()
{
    //declare some variables
    int x = 10;
    int y = 5;
    //declare a pointer to some place in memory
    int* px;
    //Get the address where x is stored
    px = &x;
    //At the adress where x is stored, put 20 instead
    *px = 20;
    printf("The value of x %d\n", x);
}
```

Computer Memory

	Address	Value
	1028	
	1024	
x	1020	10
	1016	
	1012	
	1008	
	1004	
	1000	

Pointer Example using “&”

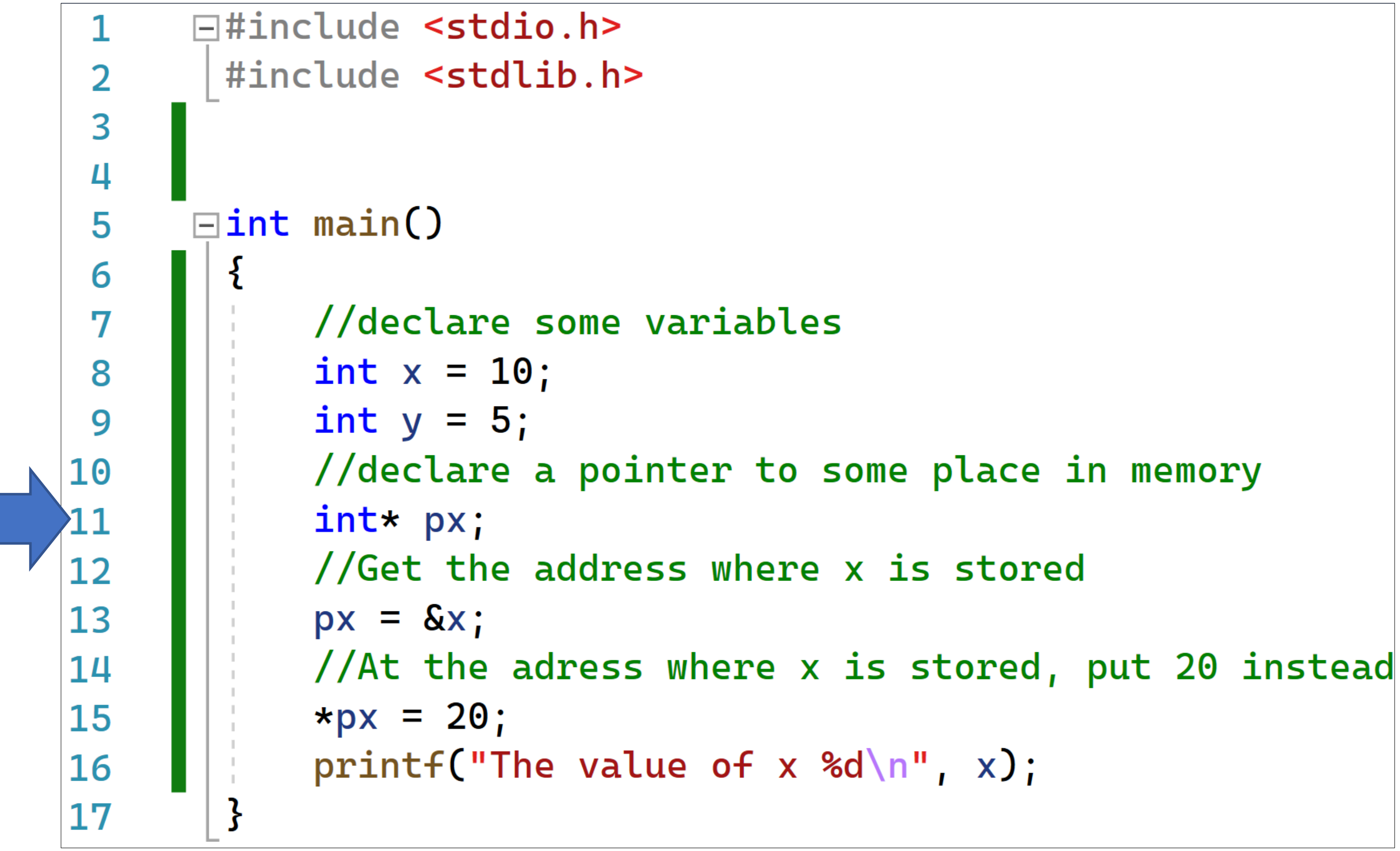


```c
#include <stdio.h>
#include <stdlib.h>


int main()
{
    //declare some variables
    int x = 10;
    int y = 5;
    //declare a pointer to some place in memory
    int* px;
    //Get the address where x is stored
    px = &x;
    //At the adress where x is stored, put 20 instead
    *px = 20;
    printf("The value of x %d\n", x);
}
```

Computer Memory

	Address	Value
	1028	
	1024	
x	1020	10
y	1016	5
	1012	
	1008	
	1004	
	1000	

Pointer Example using “&”

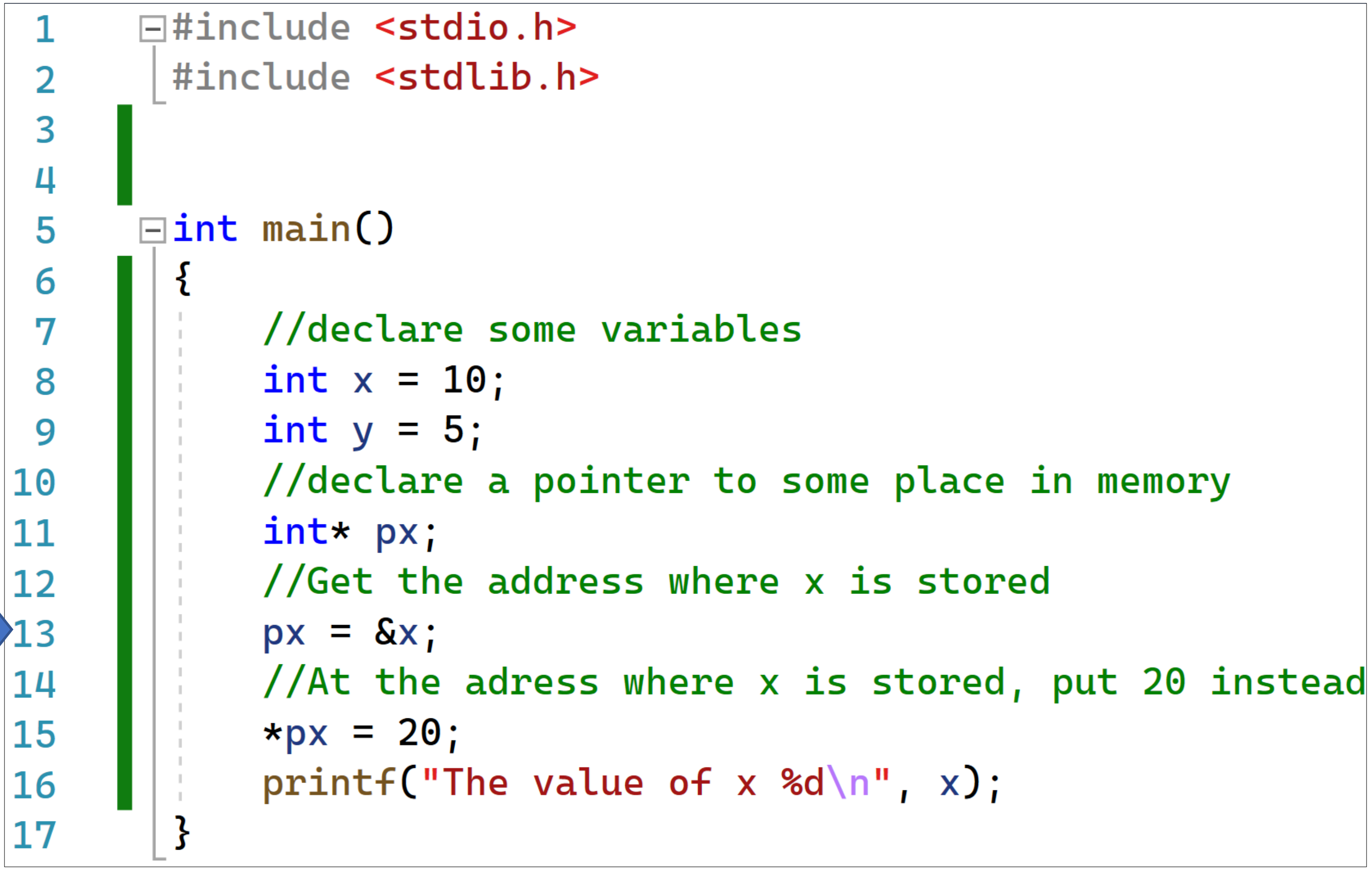


```c
#include <stdio.h>
#include <stdlib.h>


int main()
{
    //declare some variables
    int x = 10;
    int y = 5;
    //declare a pointer to some place in memory
    int* px;
    //Get the address where x is stored
    px = &x;
    //At the adress where x is stored, put 20 instead
    *px = 20;
    printf("The value of x %d\n", x);
}
```

Computer Memory

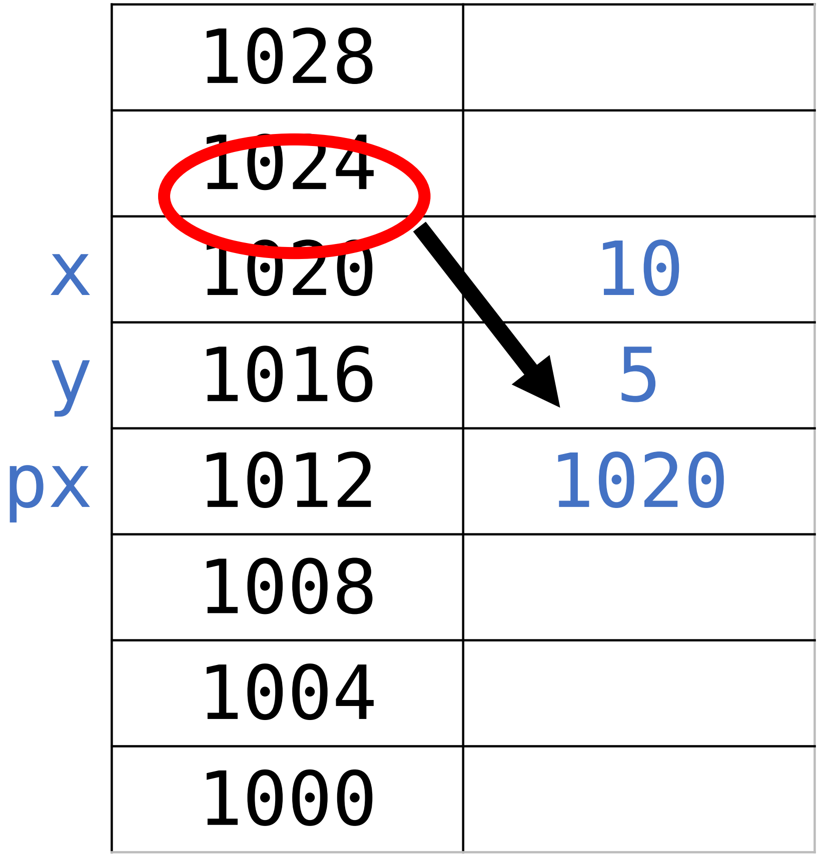
	Address	Value
	1028	
	1024	
x	1020	10
y	1016	5
px	1012	?
	1008	
	1004	
	1000	

Pointer Example using “&”



```c
#include <stdio.h>
#include <stdlib.h>


int main()
{
    //declare some variables
    int x = 10;
    int y = 5;
    //declare a pointer to some place in memory
    int* px;
    //Get the address where x is stored
    px = &x;
    //At the adress where x is stored, put 20 instead
    *px = 20;
    printf("The value of x %d\n", x);
}
```

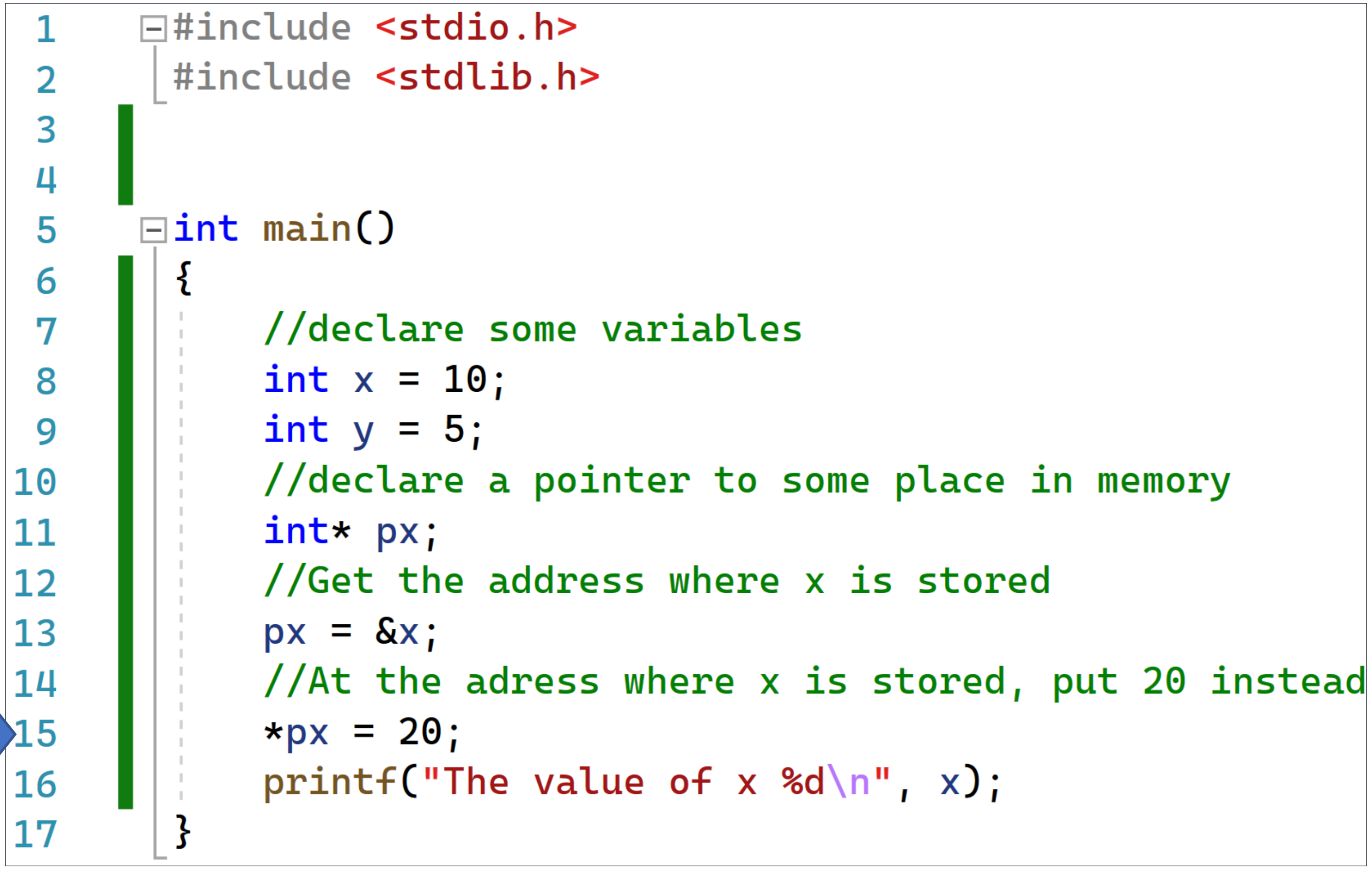
Computer Memory



	Address	Value
	1028	
	1024	
x	1020	10
y	1016	5
px	1012	1020
	1008	
	1004	
	1000	

The & symbol is saying take the address of where x is stored and store that address value in px

Pointer Example using “&”



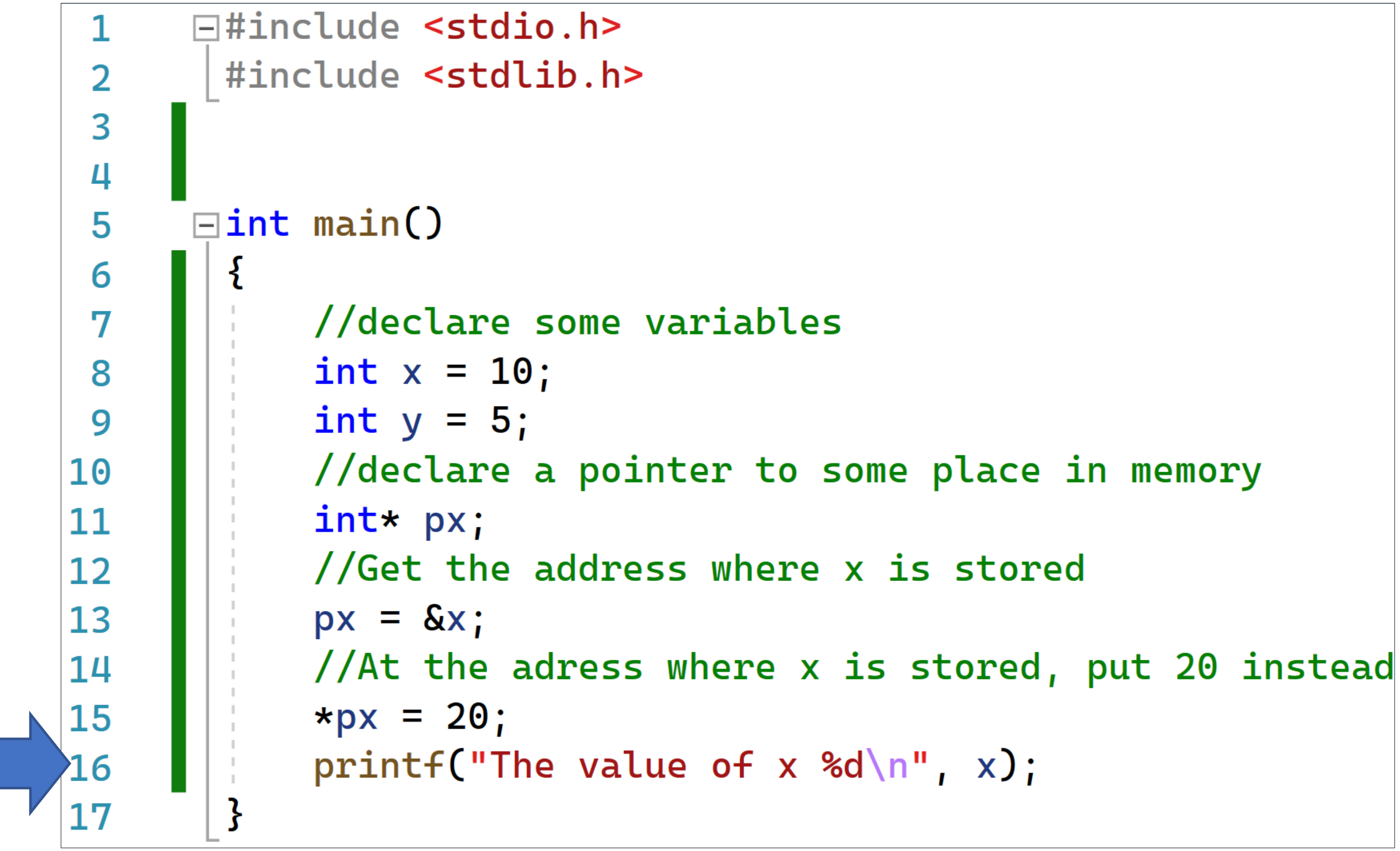
```c
#include <stdio.h>
#include <stdlib.h>


int main()
{
    //declare some variables
    int x = 10;
    int y = 5;
    //declare a pointer to some place in memory
    int* px;
    //Get the address where x is stored
    px = &x;
    //At the adress where x is stored, put 20 instead
    *px = 20;
    printf("The value of x %d\n", x);
}
```

Computer Memory

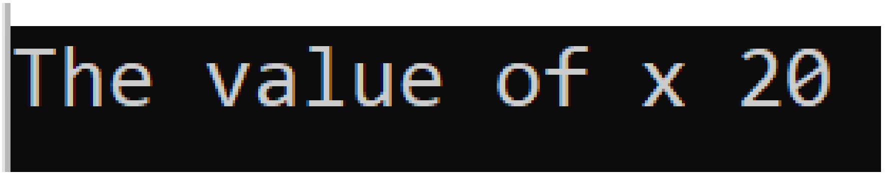
	Address	Value
	1028	
	1024	
x	1020	20
y	1016	5
px	1012	1020
	1008	
	1004	
	1000	

The * symbol is saying take what is stored at the address that px of is pointing to and put 20

Pointer Example using “&”



```c
#include <stdio.h>
#include <stdlib.h>


int main()
{
    //declare some variables
    int x = 10;
    int y = 5;
    //declare a pointer to some place in memory
    int* px;
    //Get the address where x is stored
    px = &x;
    //At the adress where x is stored, put 20 instead
    *px = 20;
    printf("The value of x %d\n", x);
}
```



```
The value of x 20
```

Computer Memory

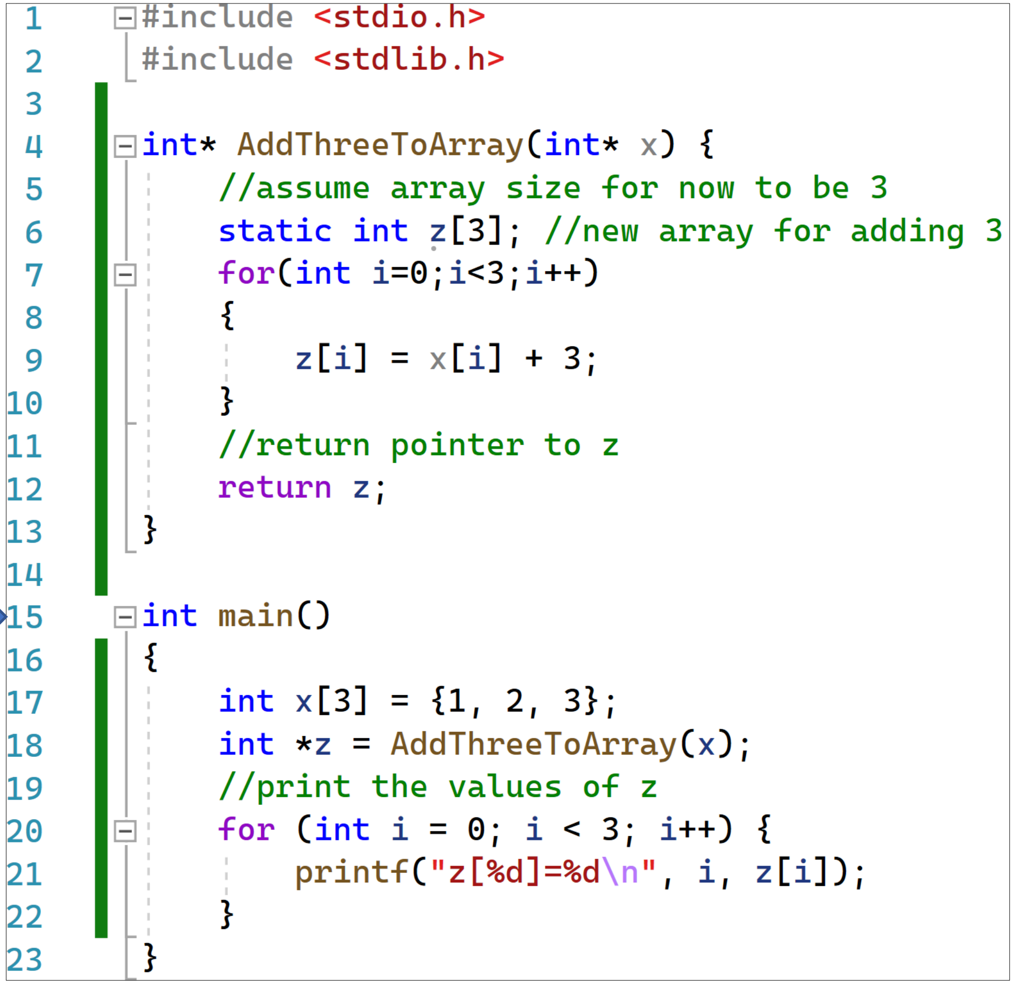
	Address	Value
	1028	
	1024	
x	1020	20
y	1016	5
px	1012	1020
	1008	
	1004	
	1000	

How would we use pointer with arrays?

- Previously we showed how we could pass arrays as inputs to functions.
- What if we want to pass arrays as outputs?
- We can use pointers. Since a function can only return “one variable” in C, we’ll return a pointer to the start of the array!

Return the pointer instead of the entire array:
It ain't much, but it's honest work

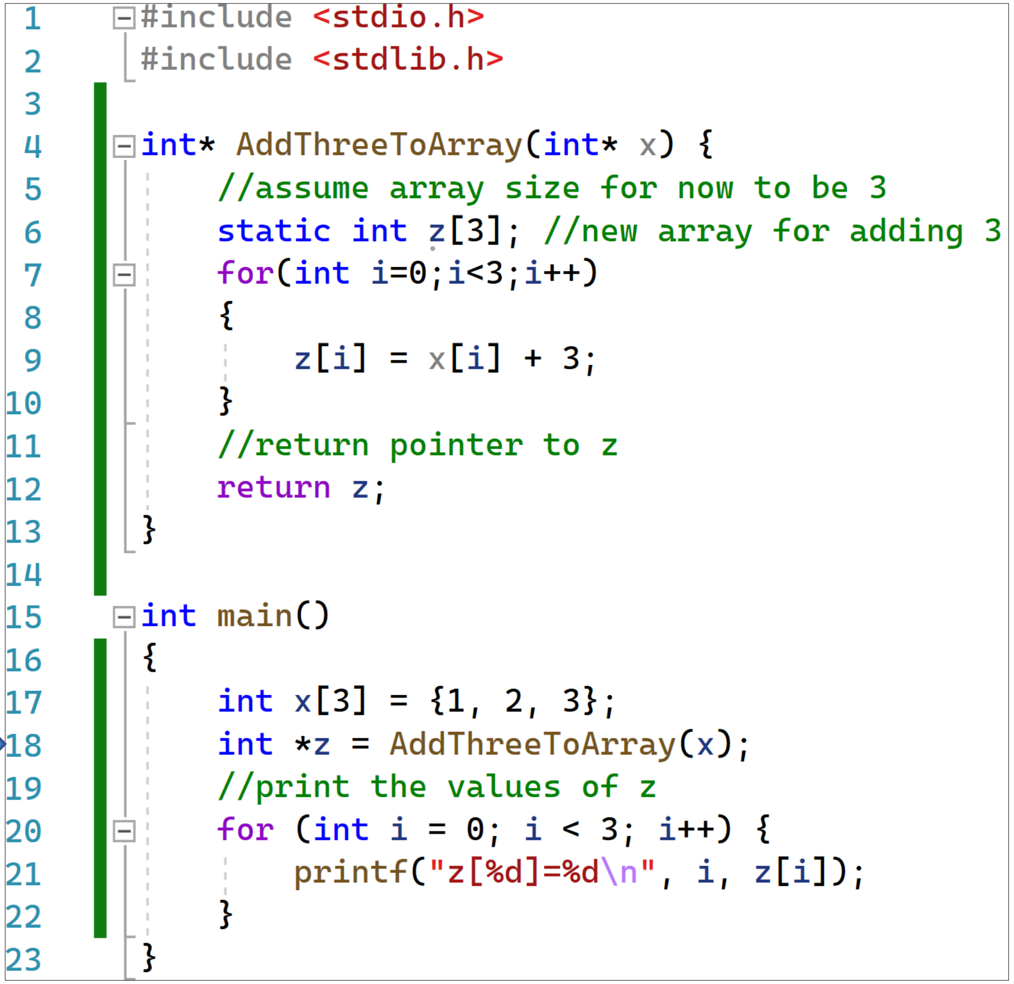
Pointer Example: Returning an Array



```c
#include <stdio.h>
#include <stdlib.h>

int* AddThreeToArray(int* x) {
    //assume array size for now to be 3
    static int z[3]; //new array for adding 3
    for(int i=0;i<3;i++)
    {
        z[i] = x[i] + 3;
    }
    //return pointer to z
    return z;
}

int main()
{
    int x[3] = {1, 2, 3};
    int *z = AddThreeToArray(x);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("z[%d]=%d\n", i, z[i]);
    }
}
```

Variables in Main

Pointer Example: Returning an Array

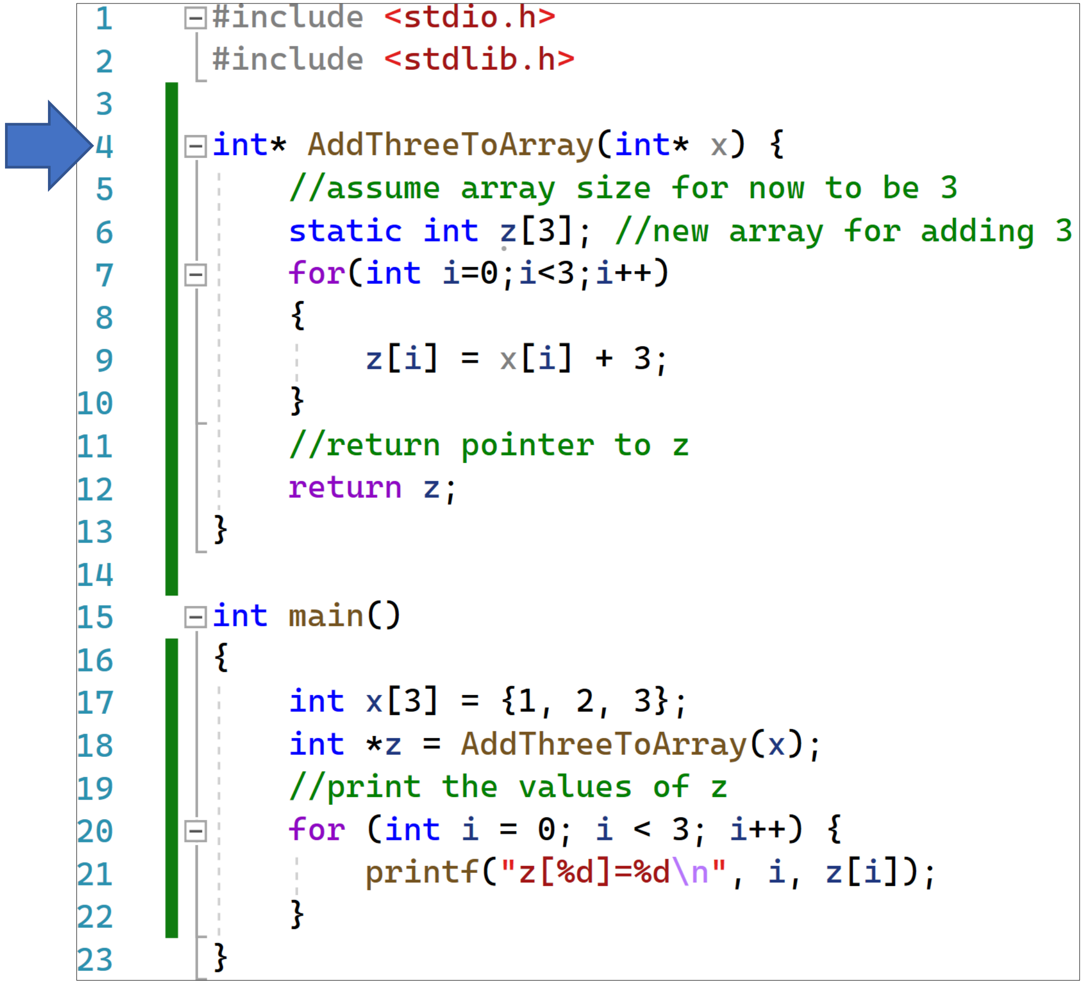


```c
#include <stdio.h>
#include <stdlib.h>

int* AddThreeToArray(int* x) {
    //assume array size for now to be 3
    static int z[3]; //new array for adding 3
    for(int i=0;i<3;i++)
    {
        z[i] = x[i] + 3;
    }
    //return pointer to z
    return z;
}

int main()
{
    int x[3] = {1, 2, 3};
    int *z = AddThreeToArray(x);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("z[%d]=%d\n", i, z[i]);
    }
}
```

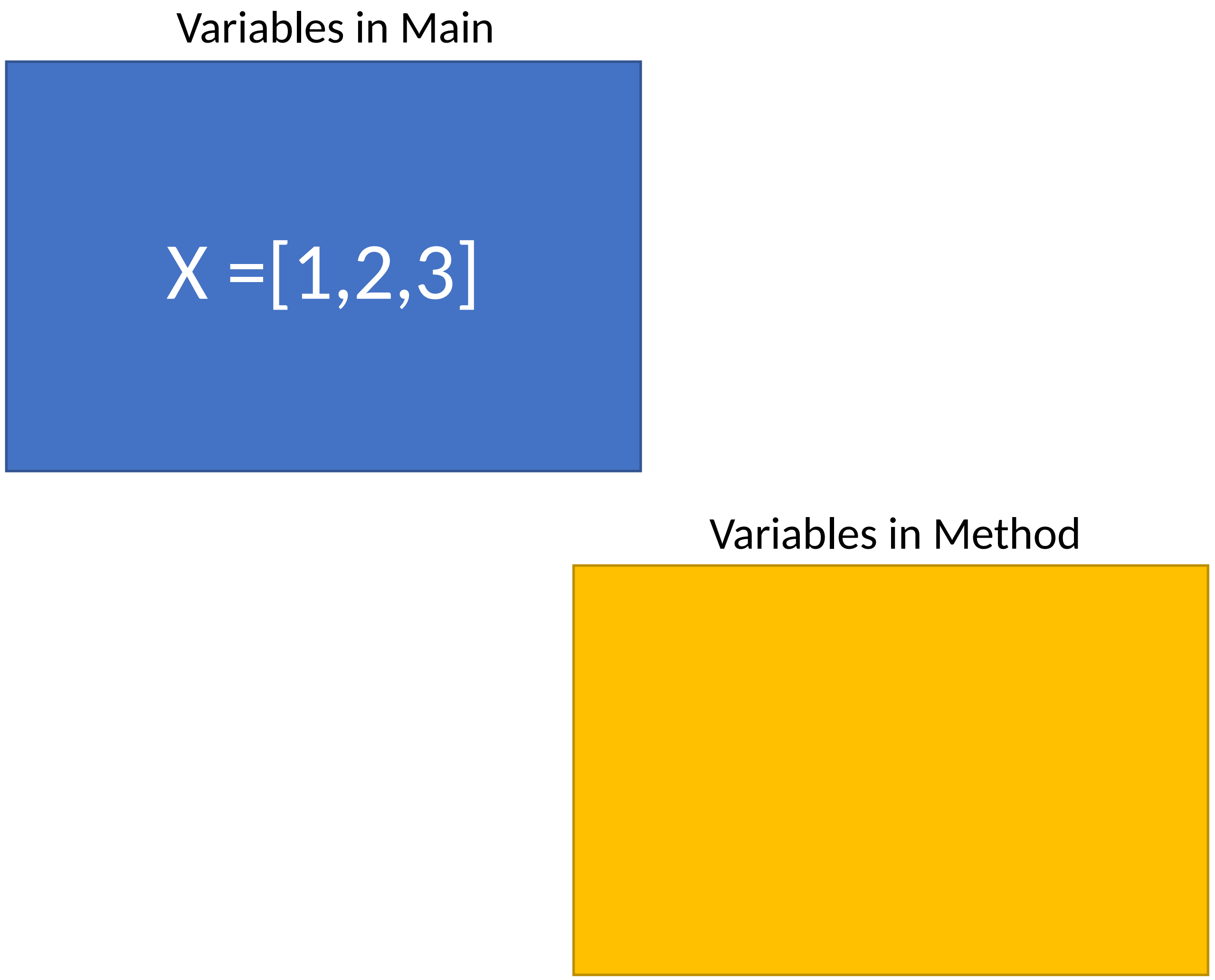
Variables in Main

X =[1,2,3]

Pointer Example: Returning an Array



```c
#include <stdio.h>
#include <stdlib.h>

int* AddThreeToArray(int* x) {
    //assume array size for now to be 3
    static int z[3]; //new array for adding 3
    for(int i=0;i<3;i++)
    {
        z[i] = x[i] + 3;
    }
    //return pointer to z
    return z;
}

int main()
{
    int x[3] = {1, 2, 3};
    int *z = AddThreeToArray(x);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("z[%d]=%d\n", i, z[i]);
    }
}
```

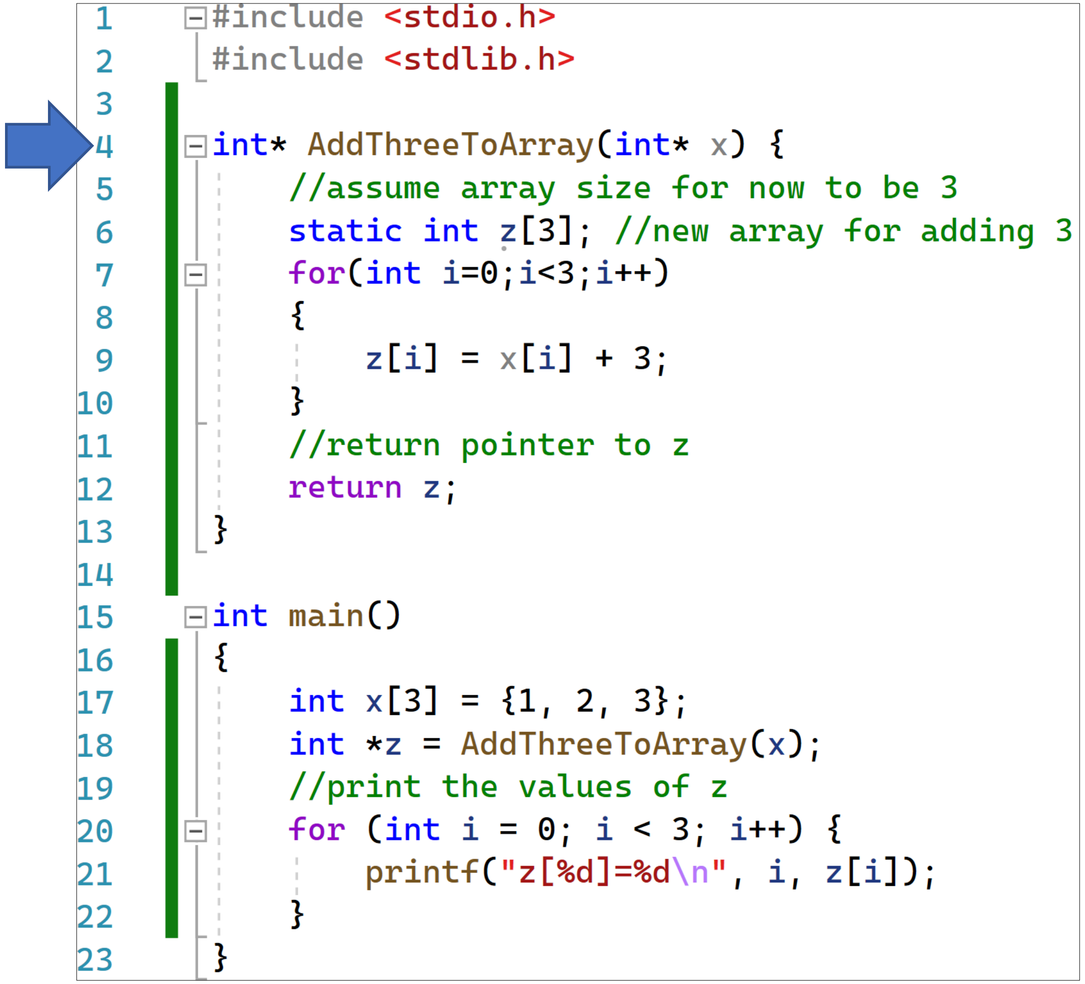


Variables in Main

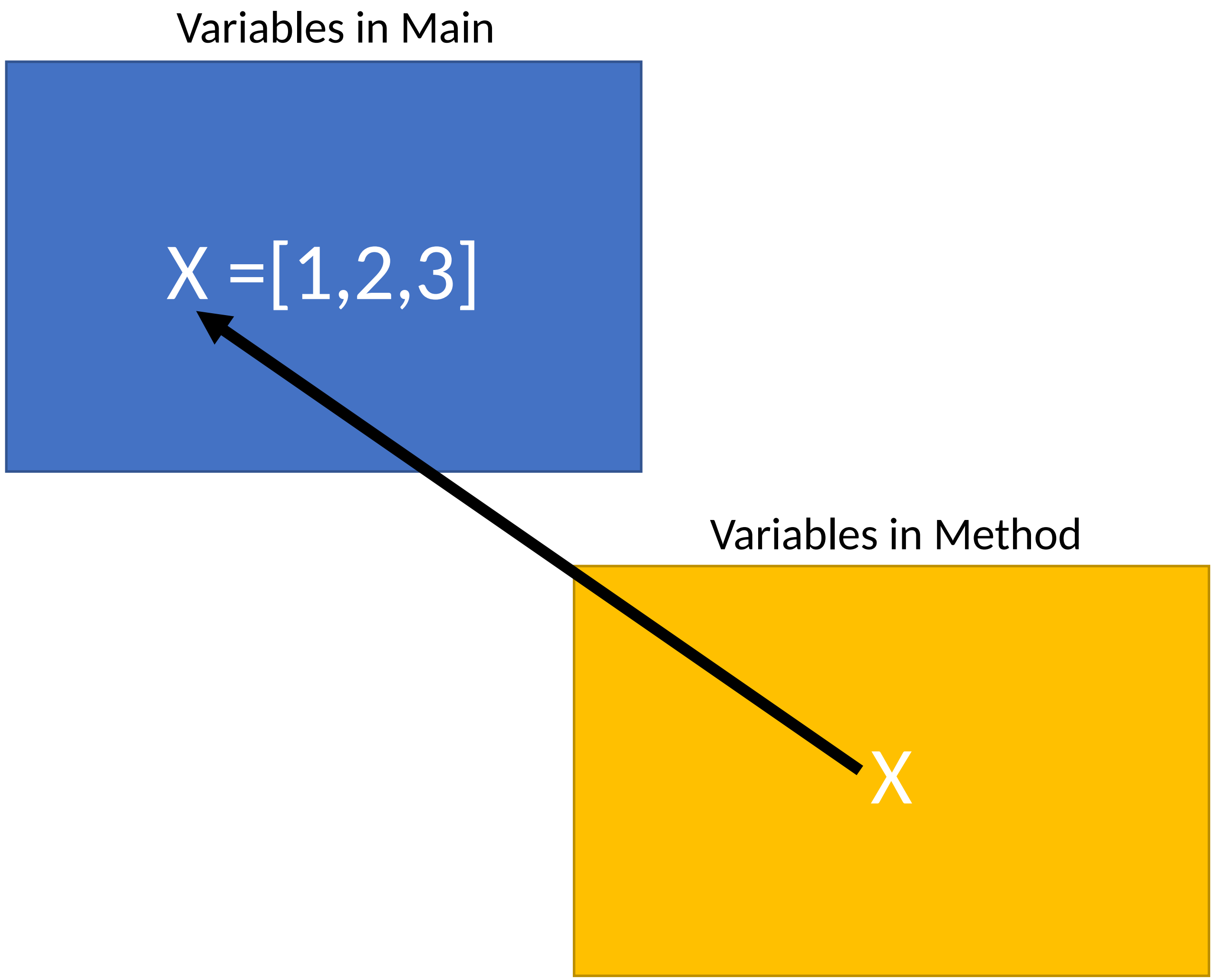
X =[1,2,3]

Variables in Method

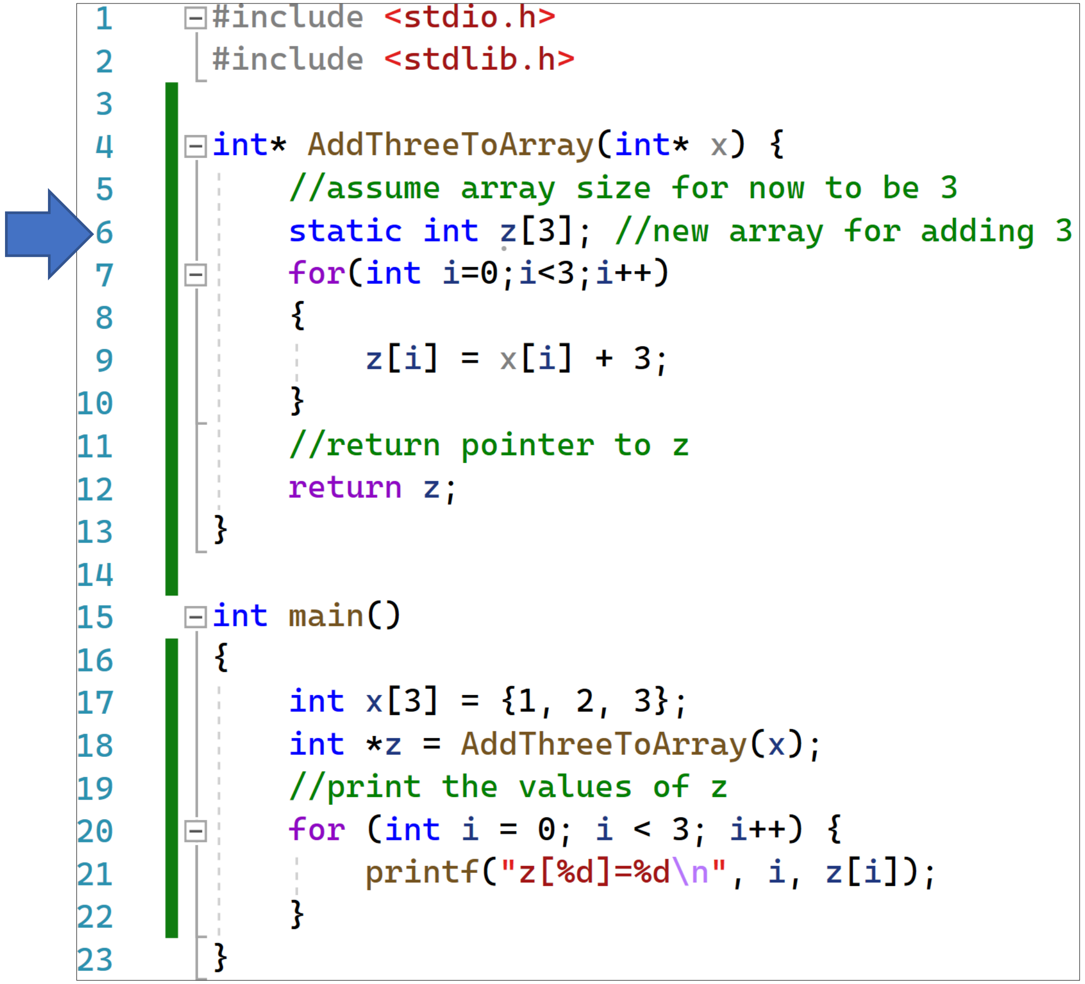
Pointer Example: Returning an Array



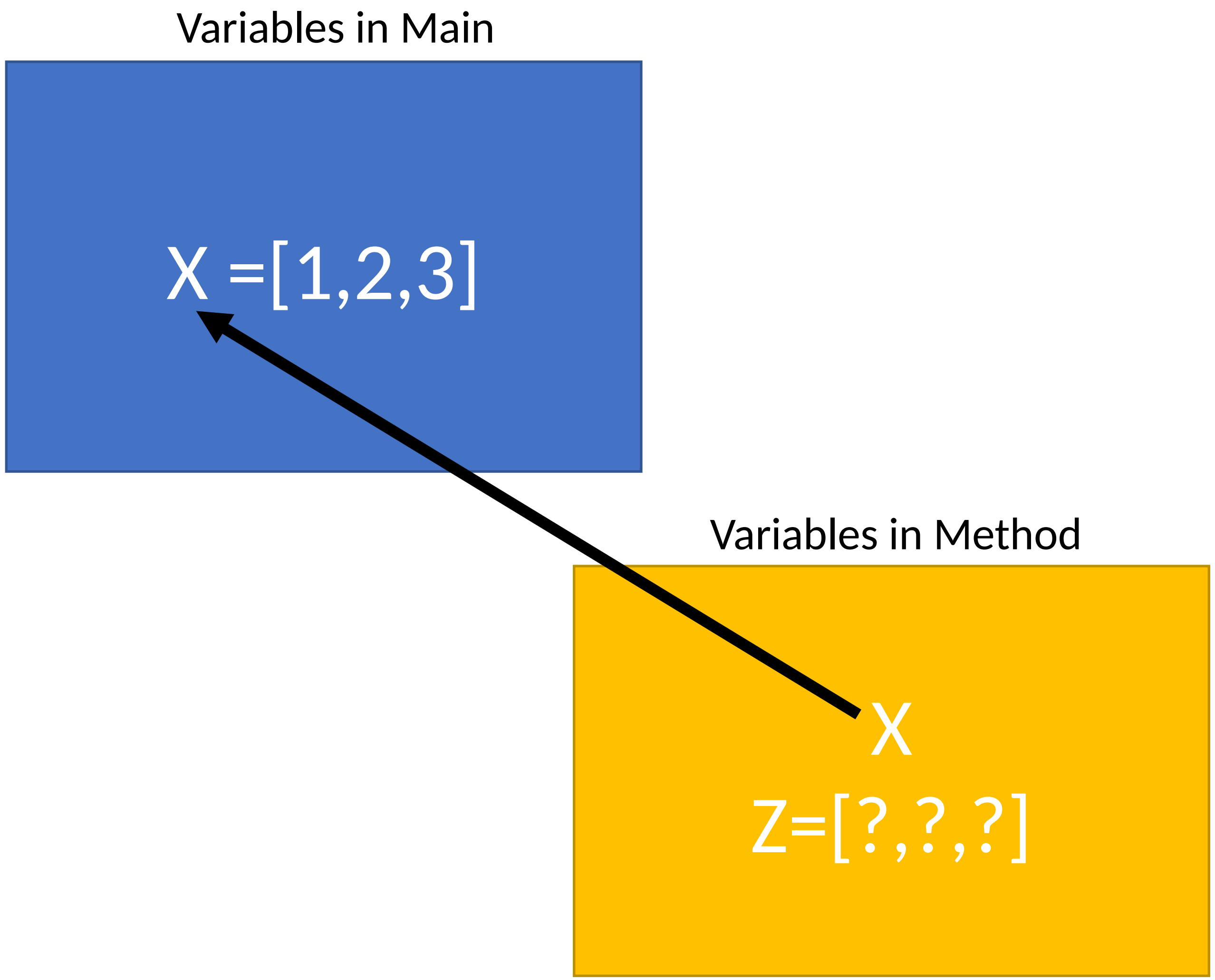
```c
#include <stdio.h>
#include <stdlib.h>

int* AddThreeToArray(int* x) {
    //assume array size for now to be 3
    static int z[3]; //new array for adding 3
    for(int i=0;i<3;i++)
    {
        z[i] = x[i] + 3;
    }
    //return pointer to z
    return z;
}

int main()
{
    int x[3] = {1, 2, 3};
    int *z = AddThreeToArray(x);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("z[%d]=%d\n", i, z[i]);
    }
}
```



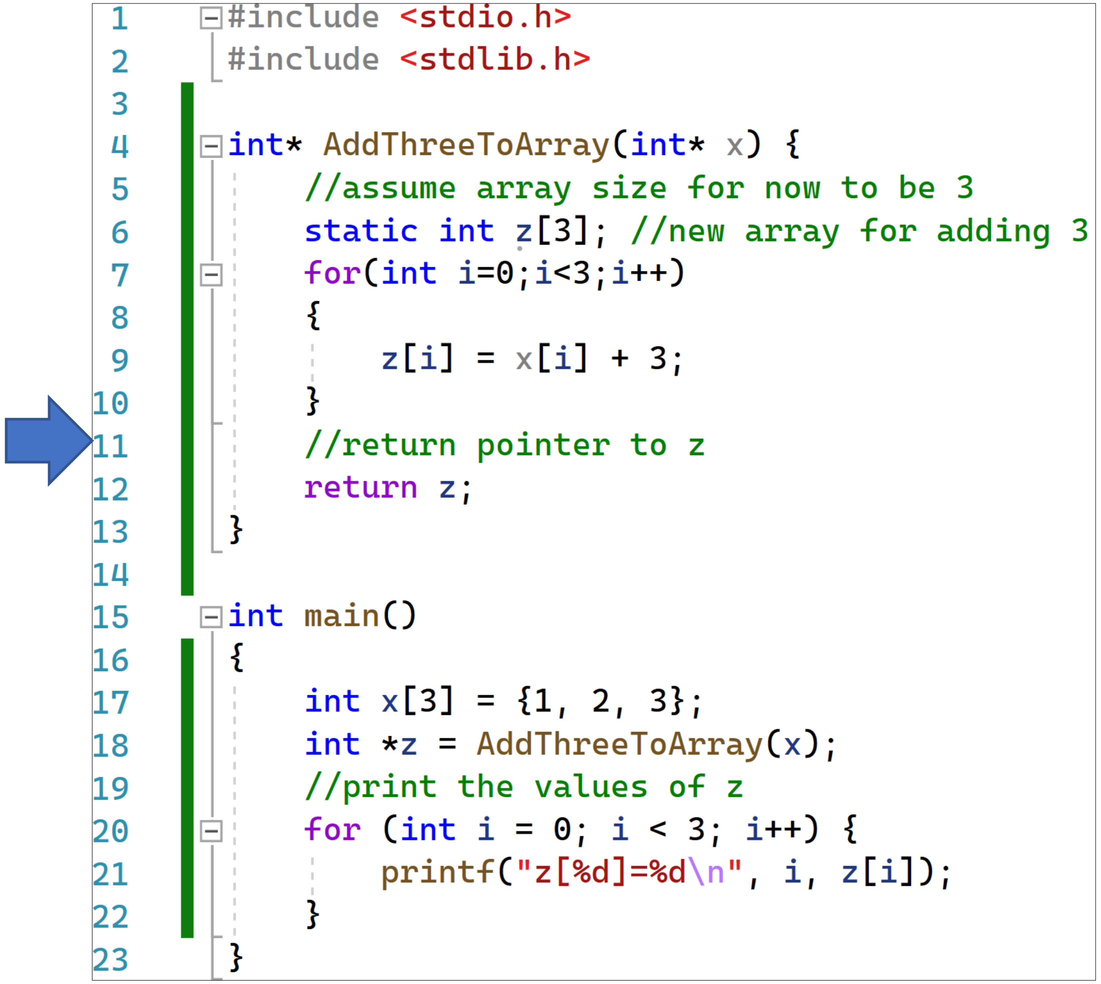
Pointer Example: Returning an Array



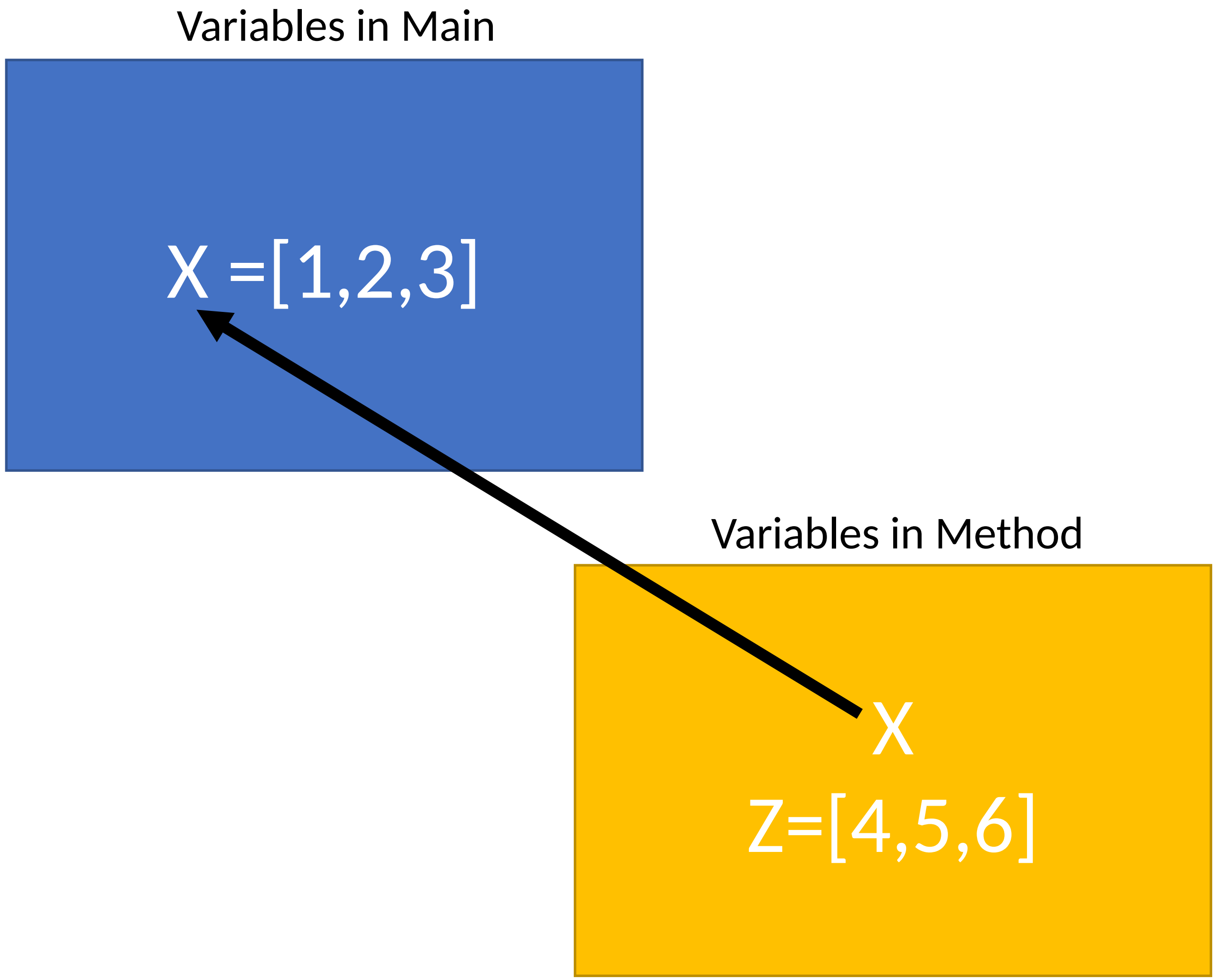
```c
#include <stdio.h>
#include <stdlib.h>

int* AddThreeToArray(int* x) {
    //assume array size for now to be 3
    static int z[3]; //new array for adding 3
    for(int i=0;i<3;i++)
    {
        z[i] = x[i] + 3;
    }
    //return pointer to z
    return z;
}

int main()
{
    int x[3] = {1, 2, 3};
    int *z = AddThreeToArray(x);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("z[%d]=%d\n", i, z[i]);
    }
}
```



Pointer Example: Returning an Array



```c
#include <stdio.h>
#include <stdlib.h>

int* AddThreeToArray(int* x) {
    //assume array size for now to be 3
    static int z[3]; //new array for adding 3
    for(int i=0;i<3;i++)
    {
        z[i] = x[i] + 3;
    }
    //return pointer to z
    return z;
}

int main()
{
    int x[3] = {1, 2, 3};
    int *z = AddThreeToArray(x);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("z[%d]=%d\n", i, z[i]);
    }
}
```



Variables in Main

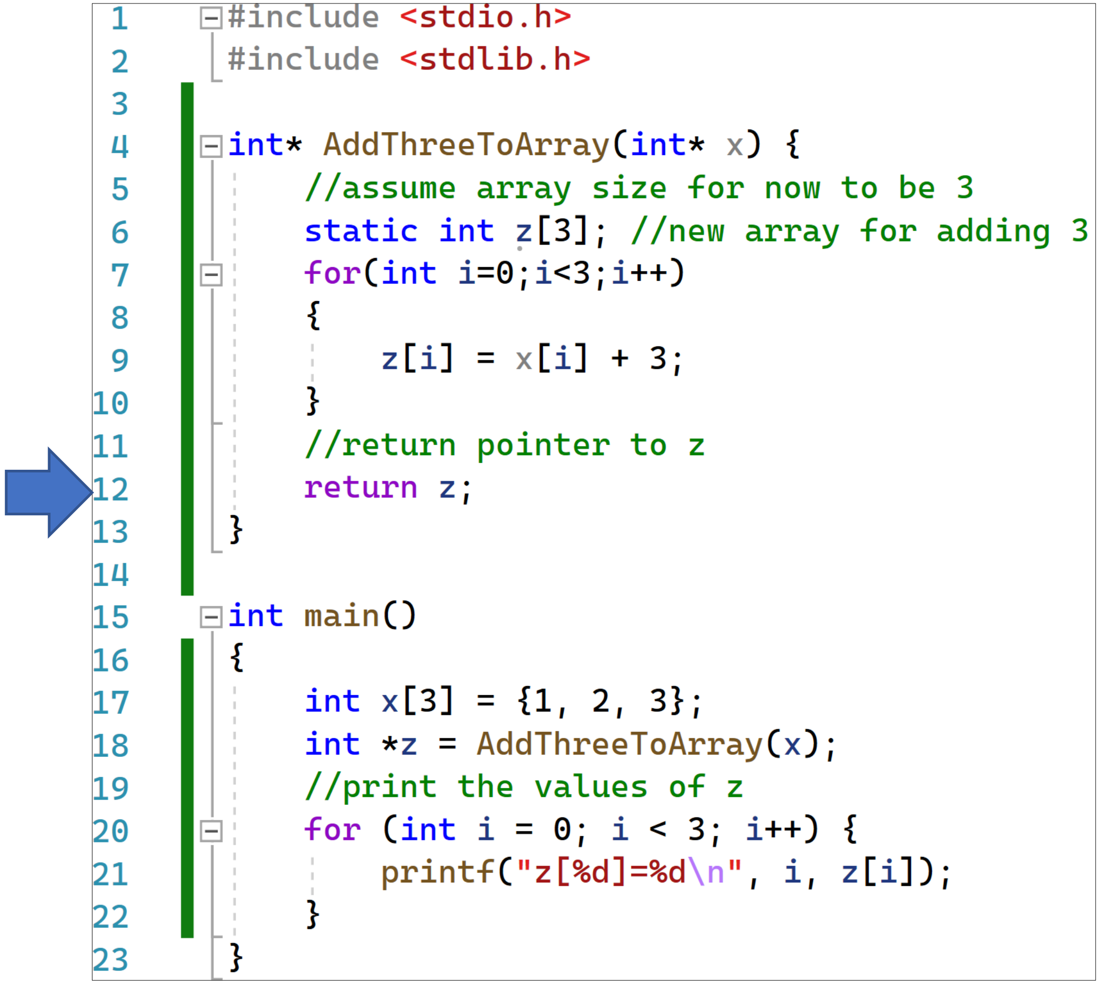
X =[1,2,3]

Variables in Method

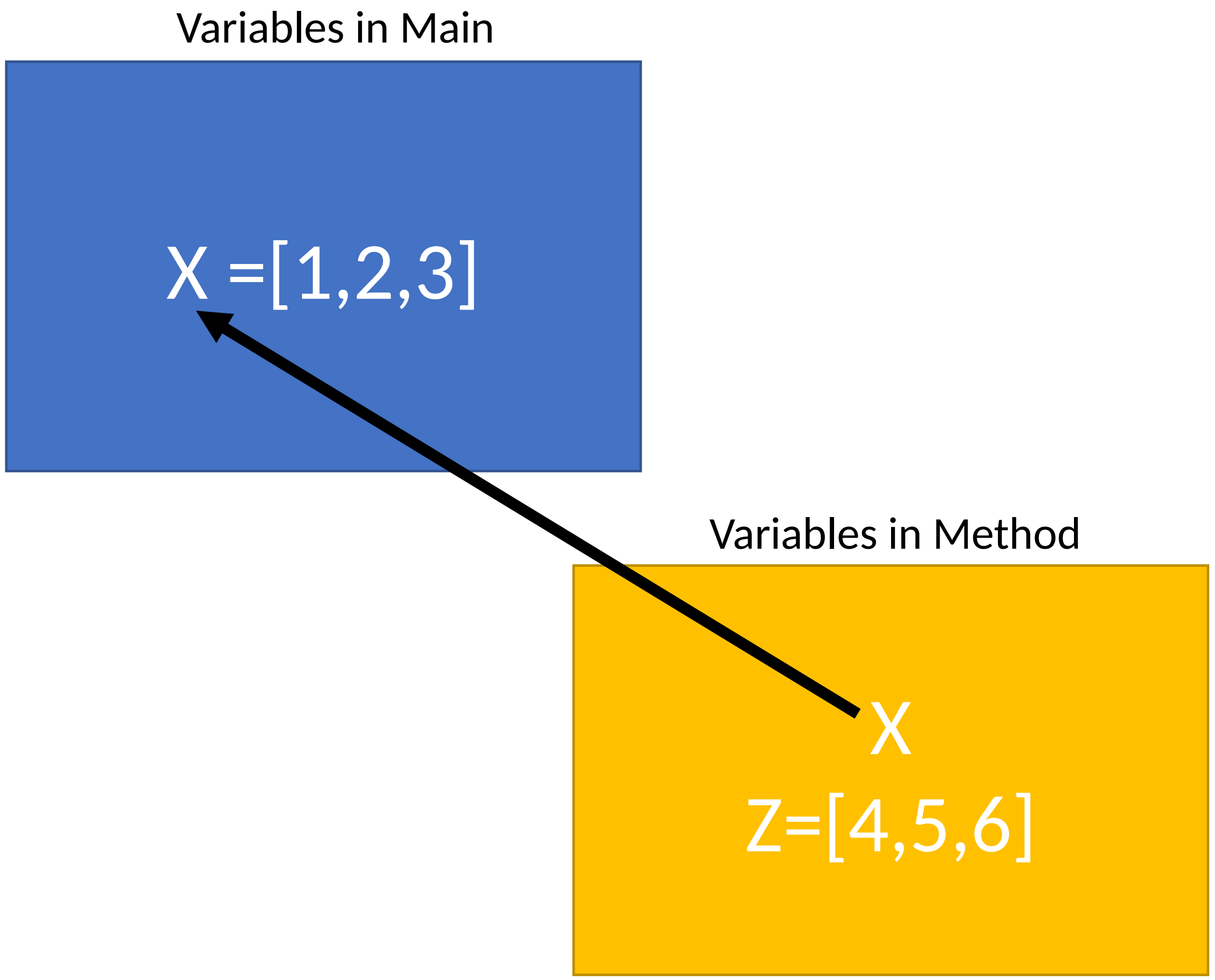
X

Z=[4,5,6]

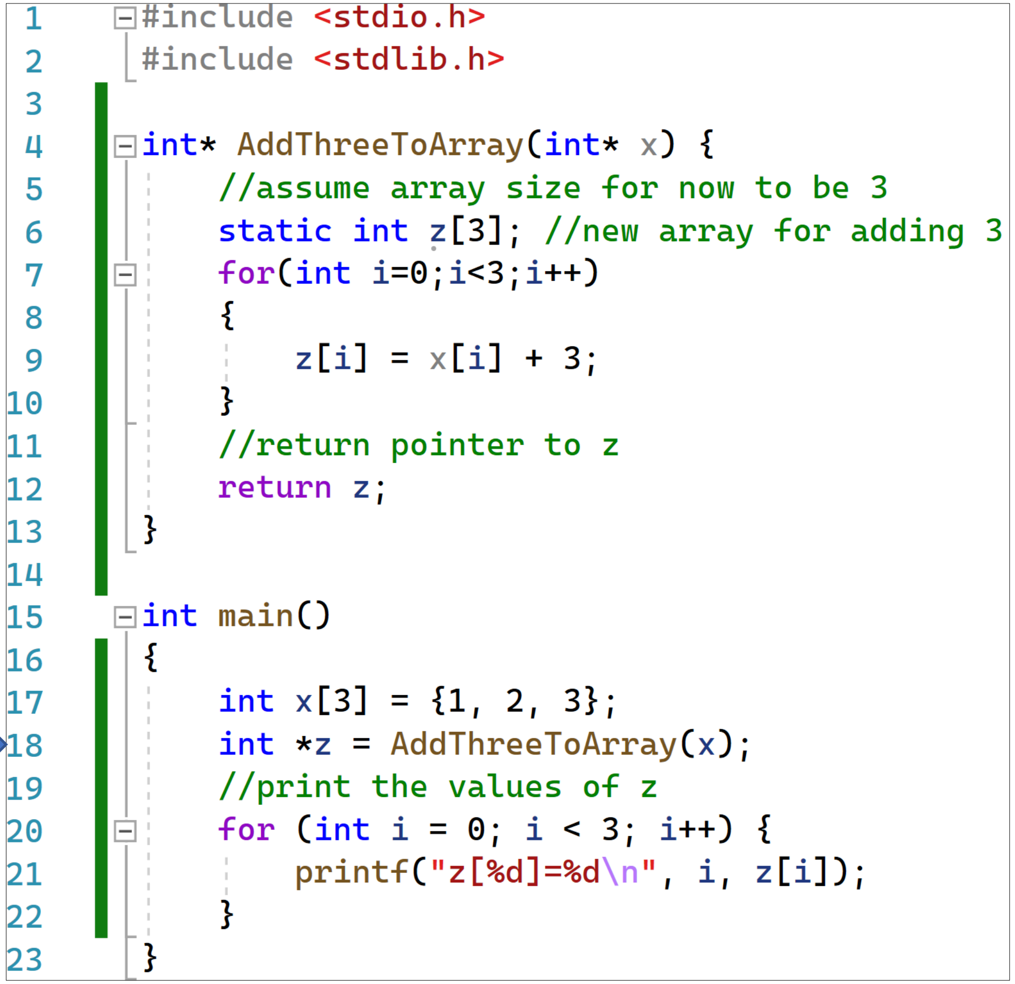
Pointer Example: Returning an Array



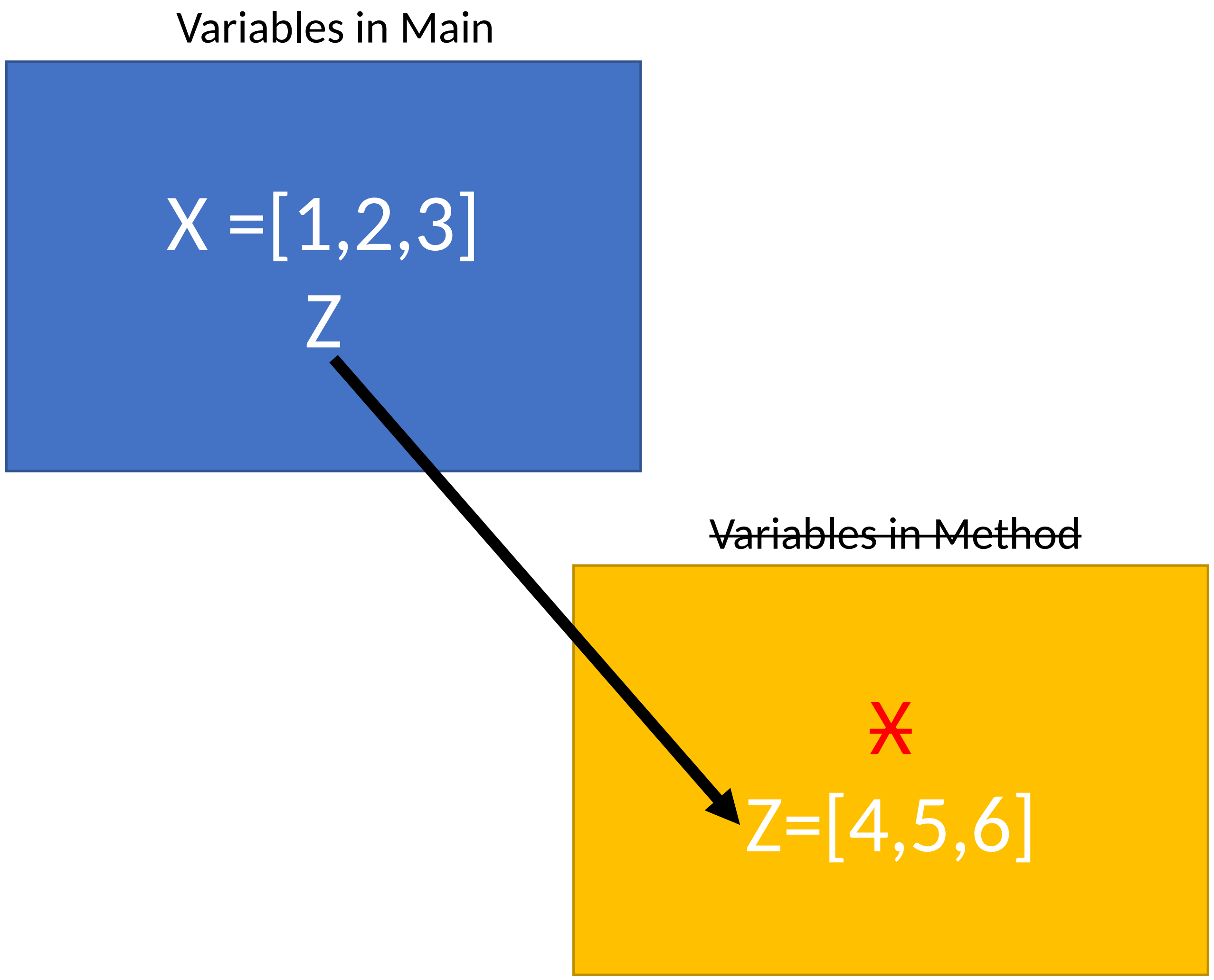
```
#include <stdio.h>
#include <stdlib.h>

int* AddThreeToArray(int* x) {
    //assume array size for now to be 3
    static int z[3]; //new array for adding 3
    for(int i=0;i<3;i++)
    {
        z[i] = x[i] + 3;
    }
    //return pointer to z
    return z;
}

int main()
{
    int x[3] = {1, 2, 3};
    int *z = AddThreeToArray(x);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("z[%d]=%d\n", i, z[i]);
    }
}
```



Pointer Example: Returning an Array



```c
#include <stdio.h>
#include <stdlib.h>

int* AddThreeToArray(int* x) {
    //assume array size for now to be 3
    static int z[3]; //new array for adding 3
    for(int i=0;i<3;i++)
    {
        z[i] = x[i] + 3;
    }
    //return pointer to z
    return z;
}

int main()
{
    int x[3] = {1, 2, 3};
    int *z = AddThreeToArray(x);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("z[%d]=%d\n", i, z[i]);
    }
}
```



Variables in Main

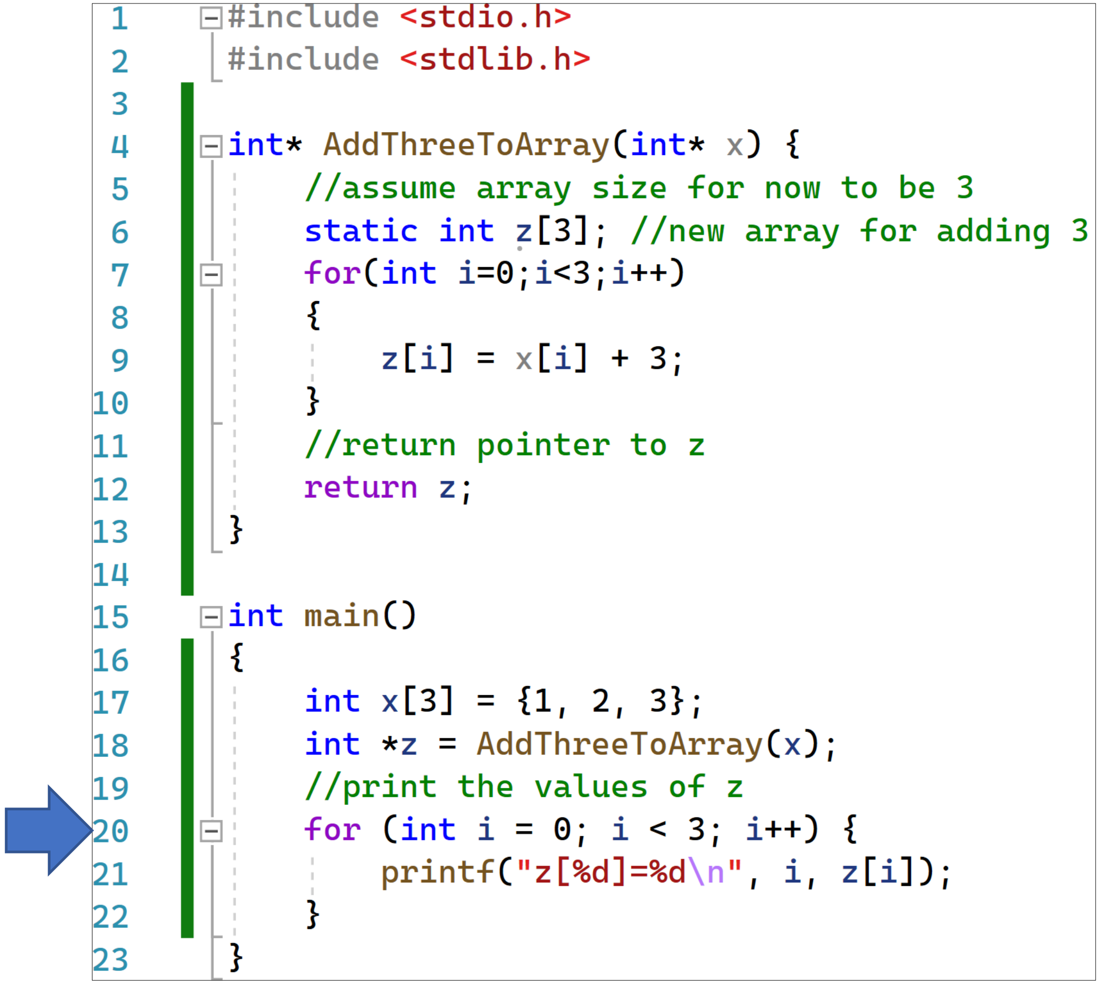
X =[1,2,3]

Z

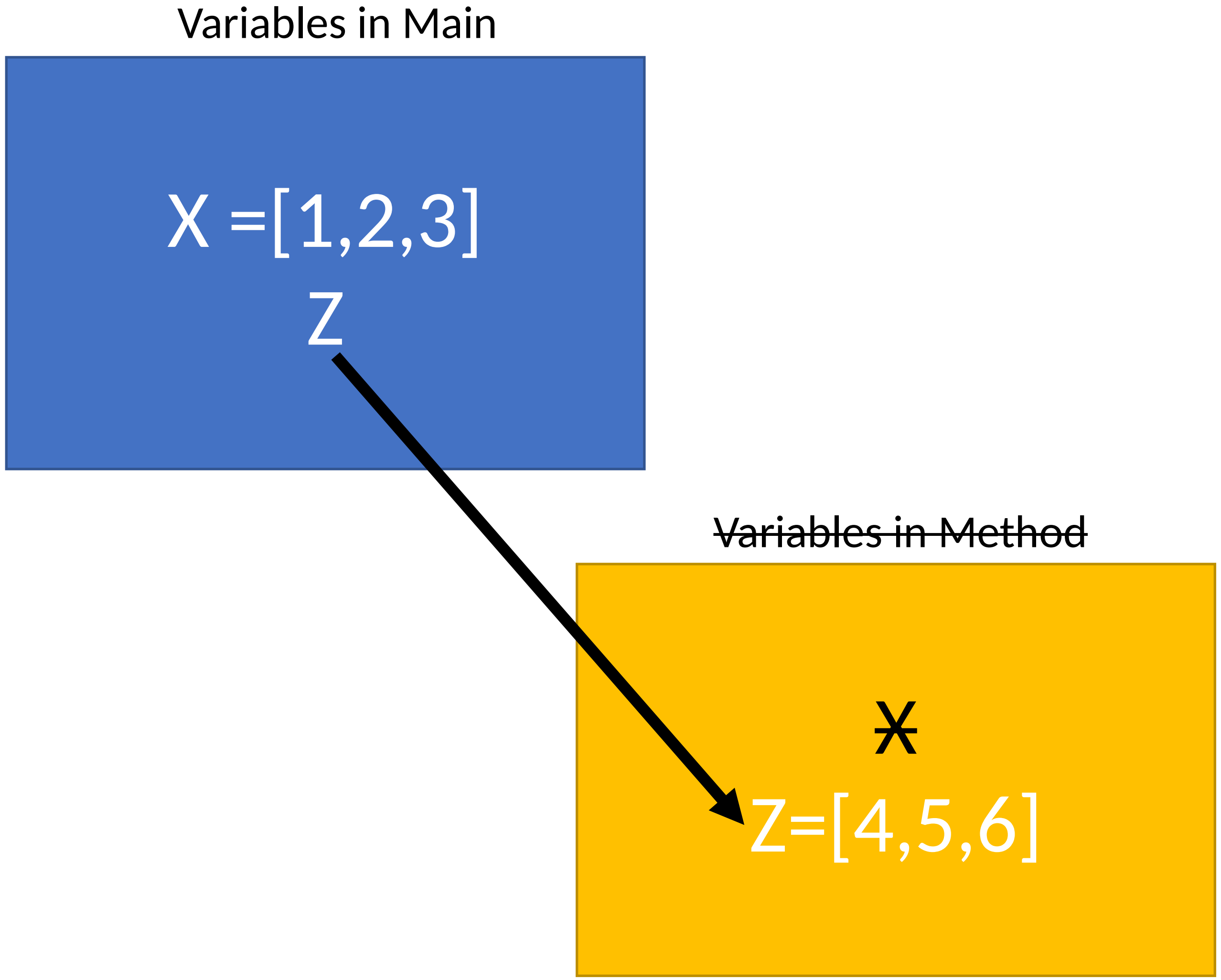
~~Variables in Method~~

Z=[4,5,6]

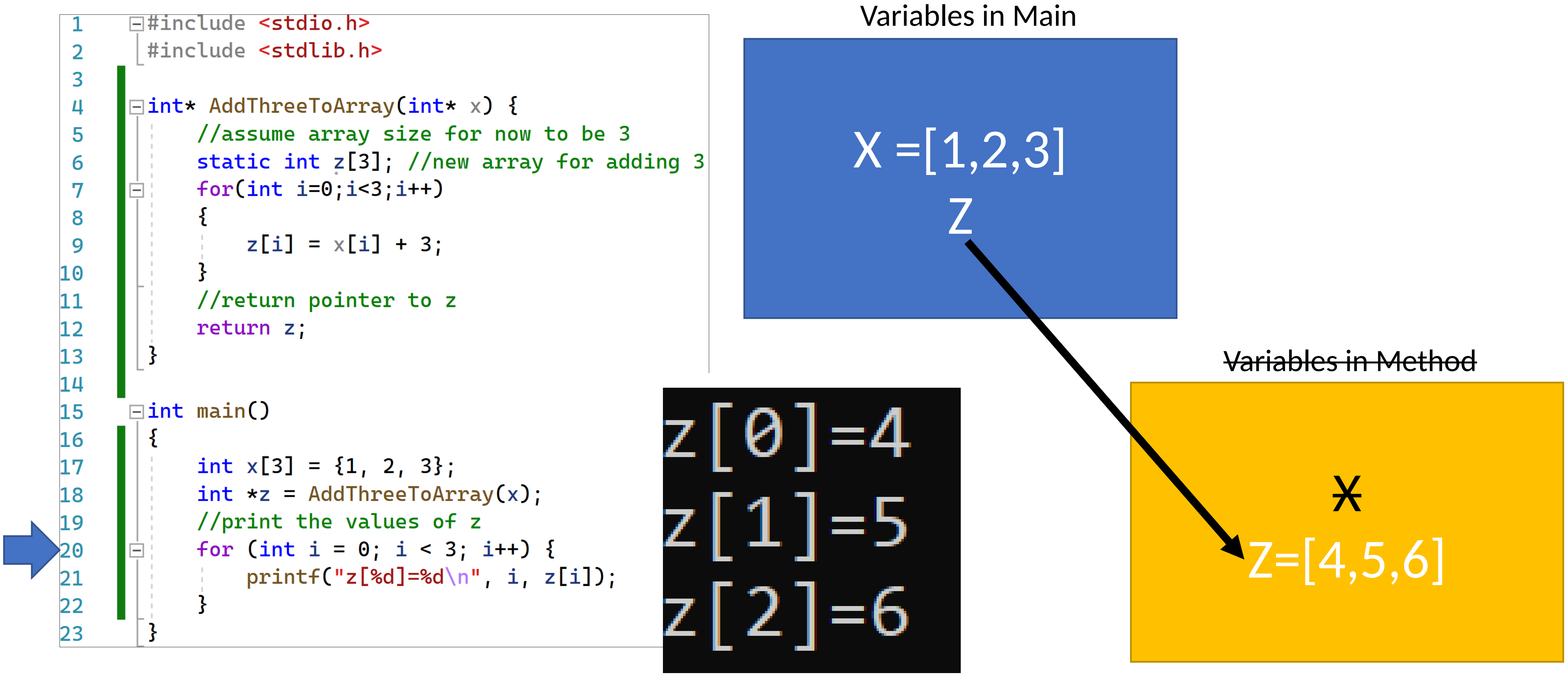
Pointer Example: Returning an Array



```c
#include <stdio.h>
#include <stdlib.h>

int* AddThreeToArray(int* x) {
    //assume array size for now to be 3
    static int z[3]; //new array for adding 3
    for(int i=0;i<3;i++)
    {
        z[i] = x[i] + 3;
    }
    //return pointer to z
    return z;
}

int main()
{
    int x[3] = {1, 2, 3};
    int *z = AddThreeToArray(x);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("z[%d]=%d\n", i, z[i]);
    }
}
```



Pointer Example: Returning an Array



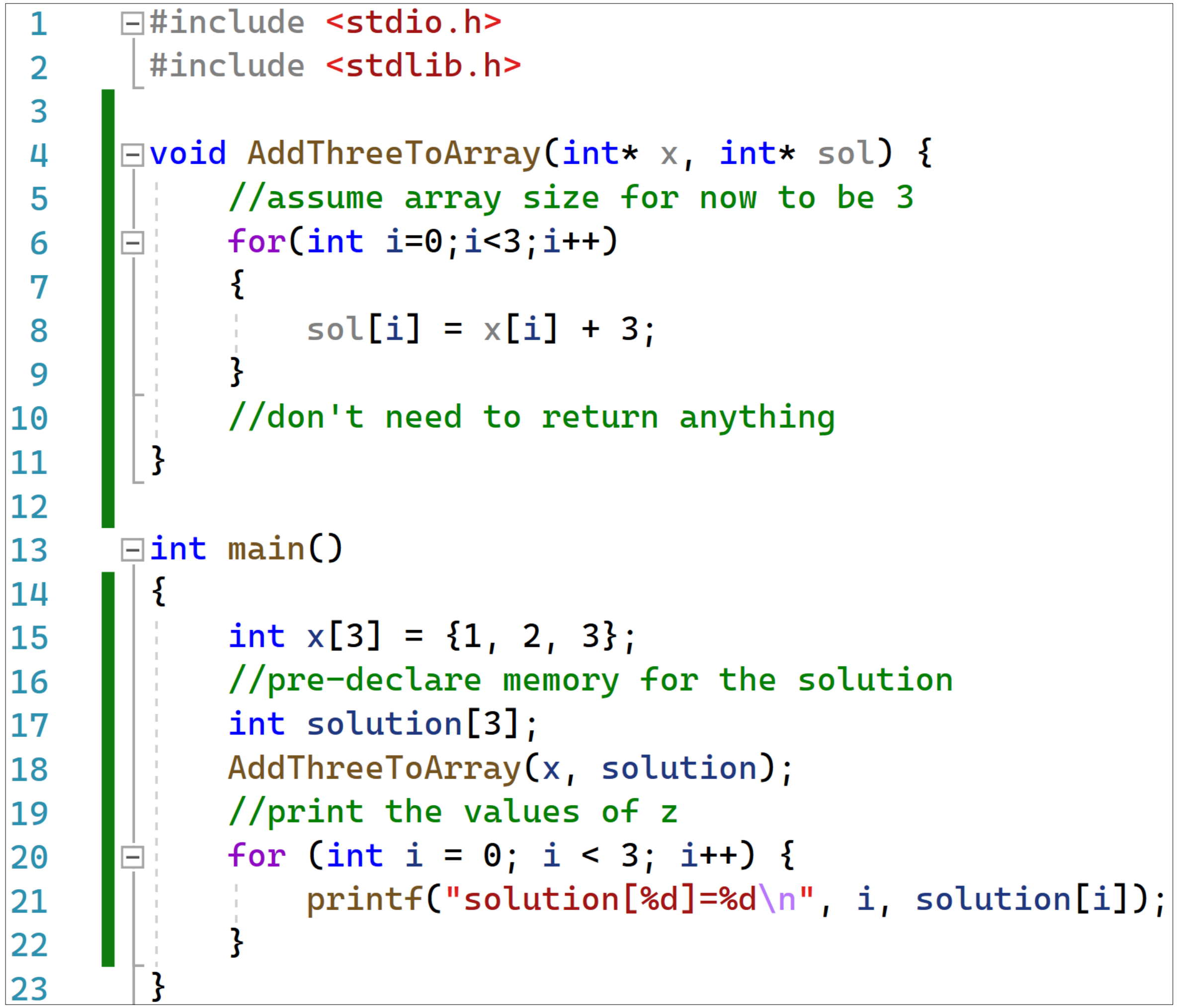
```c
#include <stdio.h>
#include <stdlib.h>

int* AddThreeToArray(int* x) {
    //assume array size for now to be 3
    static int z[3]; //new array for adding 3
    for(int i=0;i<3;i++)
    {
        z[i] = x[i] + 3;
    }
    //return pointer to z
    return z;
}

int main()
{
    int x[3] = {1, 2, 3};
    int *z = AddThreeToArray(x);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("z[%d]=%d\n", i, z[i]);
    }
}
```

```
z[0]=4
z[1]=5
z[2]=6
```

Are there any problems with this approach?

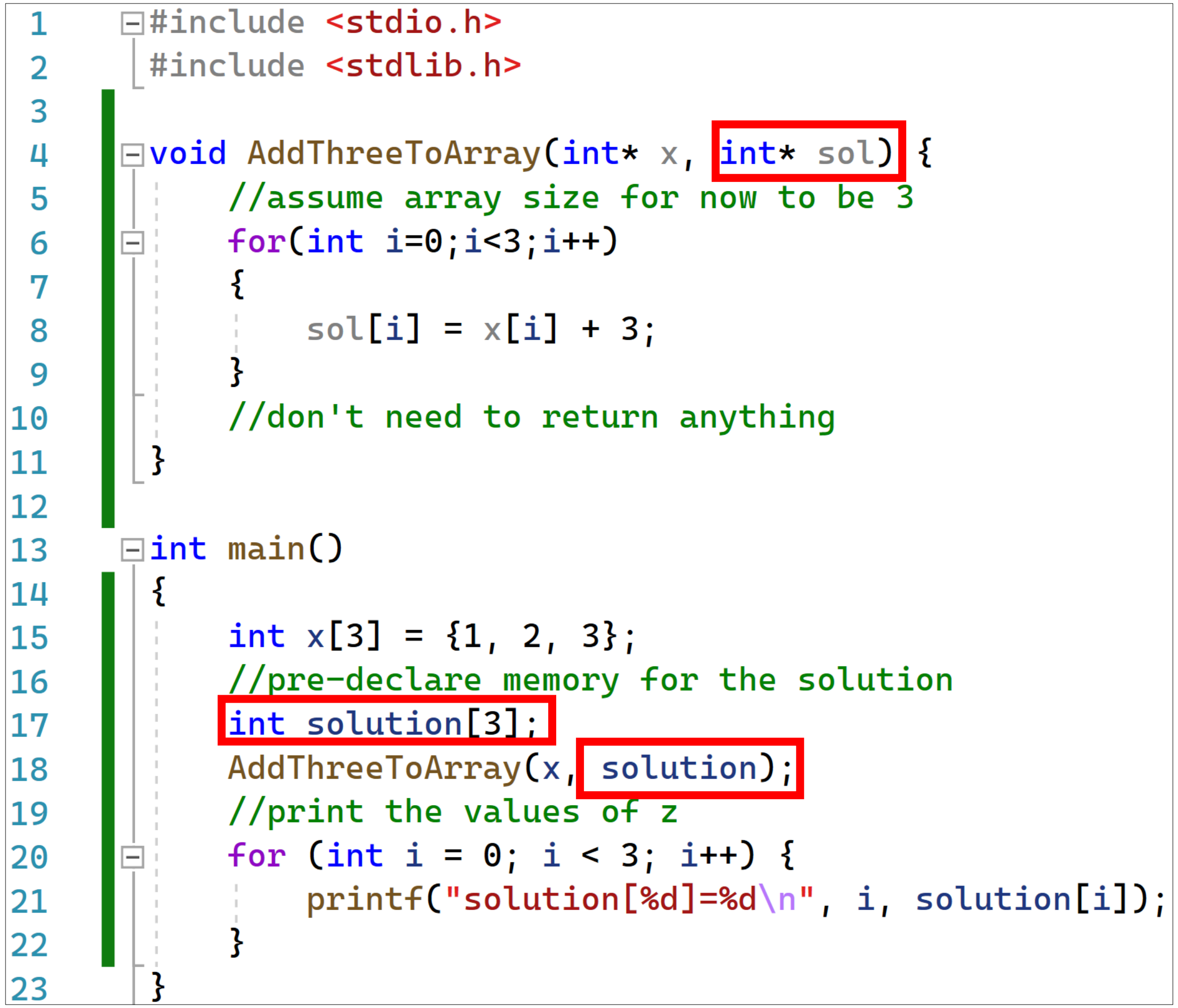
- What happens if we want to use the function multiple times?
- We only have static variable so every time z will get overwritten.

Fixing the Array Return Code in C



```c
#include <stdio.h>
#include <stdlib.h>

void AddThreeToArray(int* x, int* sol) {
    //assume array size for now to be 3
    for(int i=0;i<3;i++)
    {
        sol[i] = x[i] + 3;
    }
    //don't need to return anything
}

int main()
{
    int x[3] = {1, 2, 3};
    //pre-declare memory for the solution
    int solution[3];
    AddThreeToArray(x, solution);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("solution[%d]=%d\n", i, solution[i]);
    }
}
```

Fixing the Array Return Code in C



```c
#include <stdio.h>
#include <stdlib.h>

void AddThreeToArray(int* x, int* sol) {
    //assume array size for now to be 3
    for(int i=0;i<3;i++)
    {
        sol[i] = x[i] + 3;
    }
    //don't need to return anything
}

int main()
{
    int x[3] = {1, 2, 3};
    //pre-declare memory for the solution
    int solution[3];
    AddThreeToArray(x, solution);
    //print the values of z
    for (int i = 0; i < 3; i++) {
        printf("solution[%d]=%d\n", i, solution[i]);
    }
}
```

Fixing the Array Return Code 2

*When we want to do array manipulation with functions, is the previous solution the **ONLY** way to fix the code ?*

- The short answer: Yes (with the tools you have learned so far).
- The long answer: No. There are other ways but we'll need to delve into some new features of C in an exciting future lecture!

A few misc but related pointer/array topics

Pointer declarations

- Word to the wise…
  - The following declarations are equivalent

```
int*    p;
int  *  p;
int    *p;
```

- They all declare…
  - p to be a pointer to an integer
- But
  - First one makes the above statement clear
  - Second one is “non-committing”
  - Third says that what p points to is an integer (classic C style)

Pitfalls of Pointer Declarations

- Consider the following declarations:

```
int  *a, b;
int* c, d;
int  e, *f;
int  *g, *h;
```

What are the types of the variables?

a, c, f, g, h are int *

b, d, e are int

Multidimensional arrays

```
// declaration and initialization
int h[2][3] = { {0, 1, 2}, {10, 11, 12} };
```

Array layout in memory:
(assuming an int has four bytes)

- Row 0 first, then Row 1, …
- In each row: column 0 first, then column 1, …

Visualization of the Array

	0	1	2
0	0	1	2
1	10	11	12

	Address	Value
	1024	
h[1][2]	1020	12
h[1][1]	1016	11
h[1][0]	1012	10
h[0][2]	1008	2
h[0][1]	1004	1
h[0][0]	1000	0
	996	

Lecture Conclusions

- In C we have different types of variables: primitives and non-primitives.
- When variables are given as input to a function, primitives are copied by value, non-primitives are copied by reference.
- In C we have variables and their address in memory. The two operators to deals with this are “*” and “&”.

Figure Sources

1. https://wompampsupport.azureedge.net/fetchimage?siteId=7575&v=2&jpgQuality=100&width=700&url=https%3A%2F%2Fi.kym-cdn.com%2Fentries%2Ficons%2Ffacebook%2F000%2F026%2F366%2Fpather.jpg
2. https://i.kym-cdn.com/photos/images/newsfeed/000/531/557/a88.jpg
3. https://loveincorporated.blob.core.windows.net/contentimages/main/2ba923f2-25b7-403b-b3c3-95ac2016c7bb-shire-hobbit-home.jpg
4. https://s3.amazonaws.com/pix.iemoji.com/images/emoji/apple/ios-12/256/backhand-index-pointing-right.png
5. https://i.kym-cdn.com/entries/icons/original/000/028/021/work.jpg
6. https://i.imgur.com/i9JNNyJ.jpg