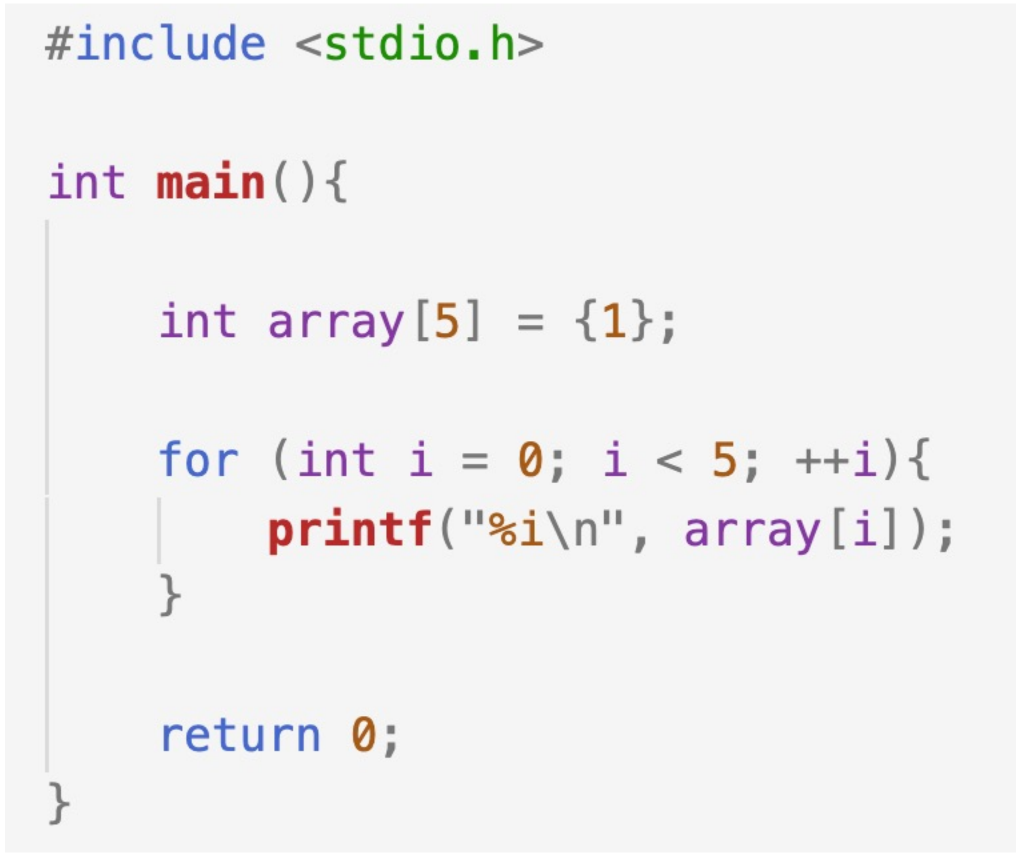
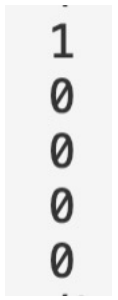
Review

Predict the output

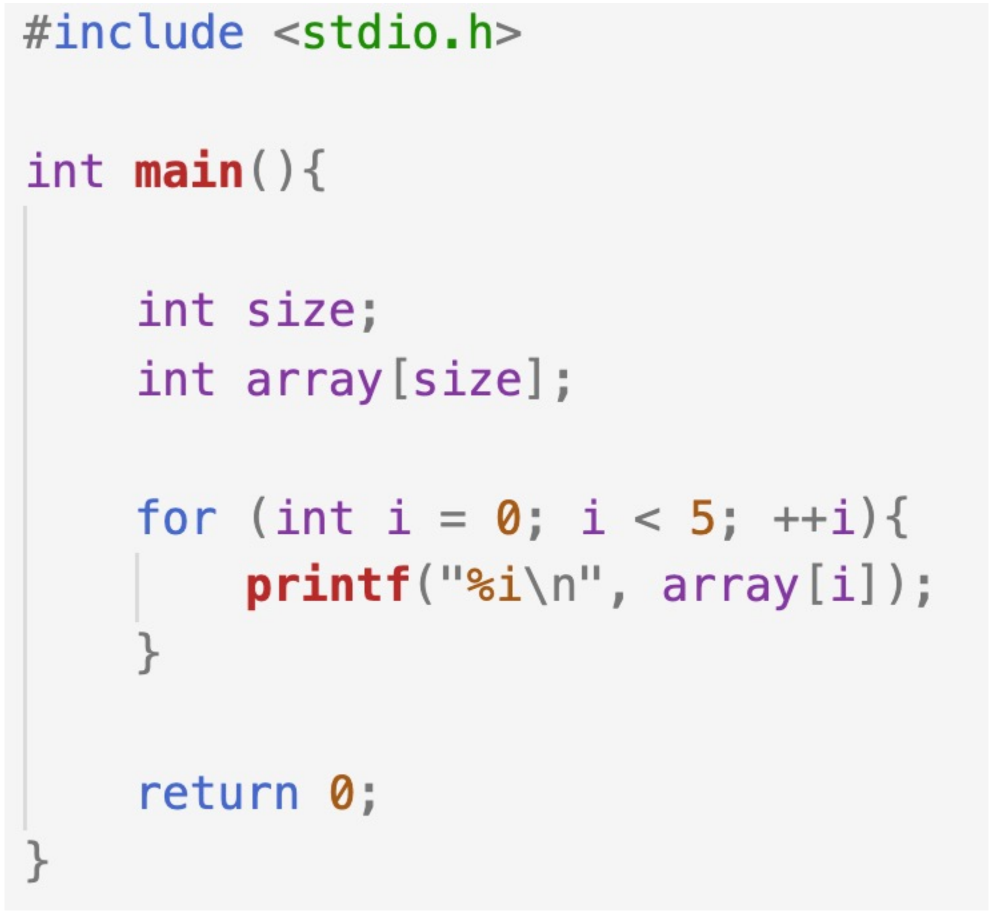


```c
#include <stdio.h>

int main(){

    int array[5] = {1};

    for (int i = 0; i < 5; ++i){
        printf("%i\n", array[i]);
    }

    return 0;
}
```



```
1
0
0
0
0
```

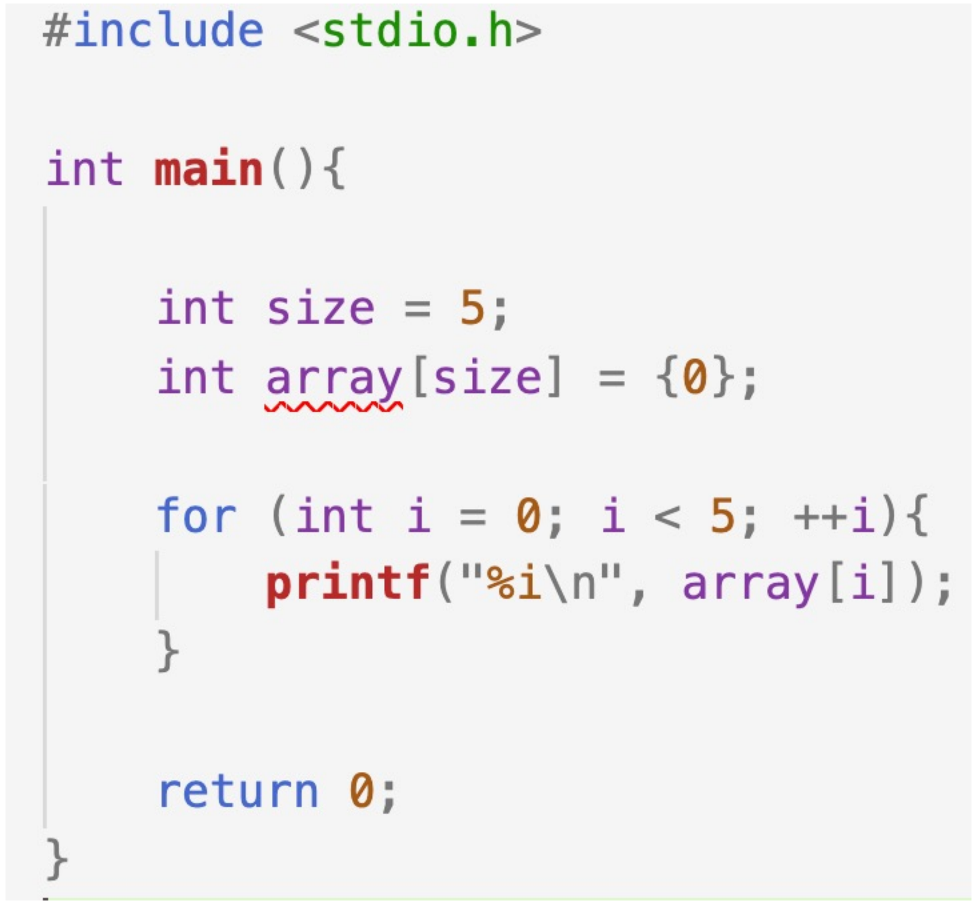
Predict the output



```c
#include <stdio.h>

int main(){

    int size;
    int array[size];

    for (int i = 0; i < 5; ++i){
        printf("%i\n", array[i]);
    }

    return 0;
}
```

- Segmentation fault: 11

What is the error?



```
#include <stdio.h>

int main(){

    int size = 5;
    int array[size] = {0};

    for (int i = 0; i < 5; ++i){
        printf("%i\n", array[i]);
    }

    return 0;
}
```

Functions

Working with Functions

- What are the functions that you’ve used so far (in this class)?
  - printf()
  - scanf()
  - main()

What is a function?

- Function is a self-contained block of code that performs a coherent task of some kind (Let Us C- Yashwant Kanetkar)
- Syntax
  - <return type> <name> (<parameter list>) { <body> }
  - int main(){<body>}

What is a Function?

```c
#include <stdio.h>

int main (void)
{
    printf ("Programming is fun.\n");

    return 0;
}
```

```c
#include <stdio.h>

void printMessage (void)
{
    printf ("Programming is fun.\n");
}

int main (void)
{
    printMessage ();

    return 0;
}
```

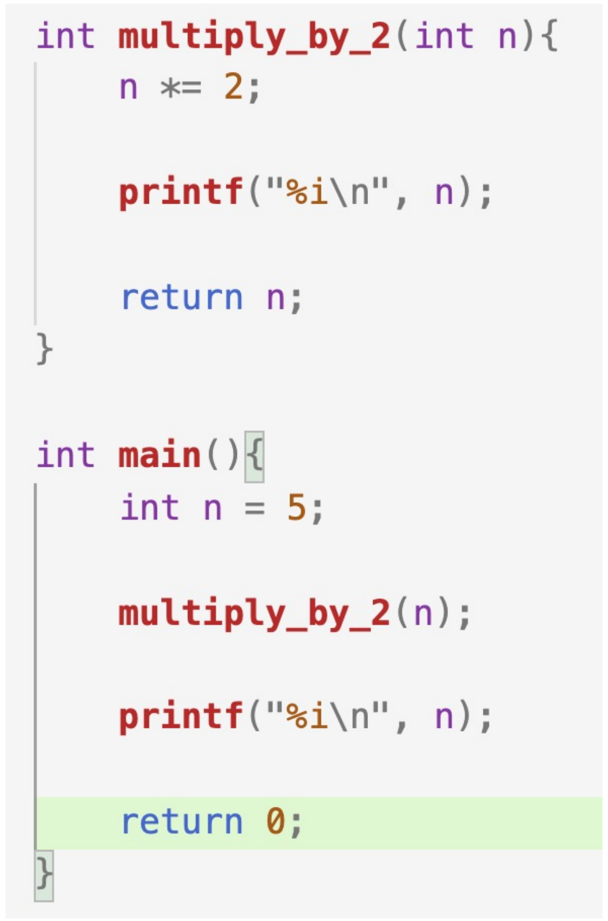
What is a Function?

- Both serve the same purpose
- The difference lies in the first and last line
- The first line of a function definition tells the compiler
  - The type of value it returns
  - Its name
  - The arguments it takes

Function Parameters

- Function Parameters
  - Variables which are local to the function
  - Could have the same names as variables in other functions, but they are distinct
- The function caller supplies arguments which are copied to the parameters

Predict Output

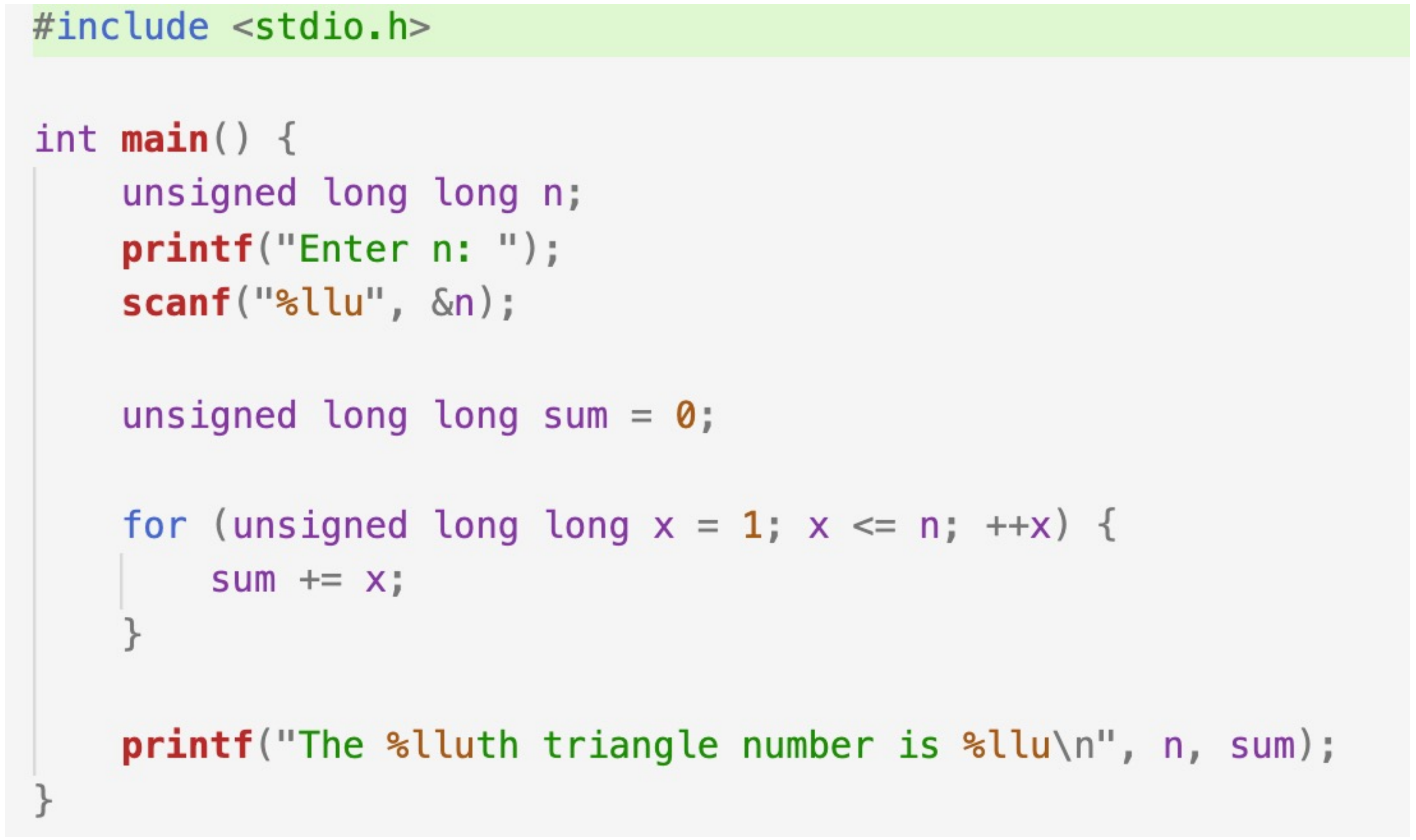


```
int multiply_by_2(int n){
    n *= 2;

    printf("%i\n", n);

    return n;
}

int main(){
    int n = 5;

    multiply_by_2(n);

    printf("%i\n", n);

    return 0;
}
```

Why use function?

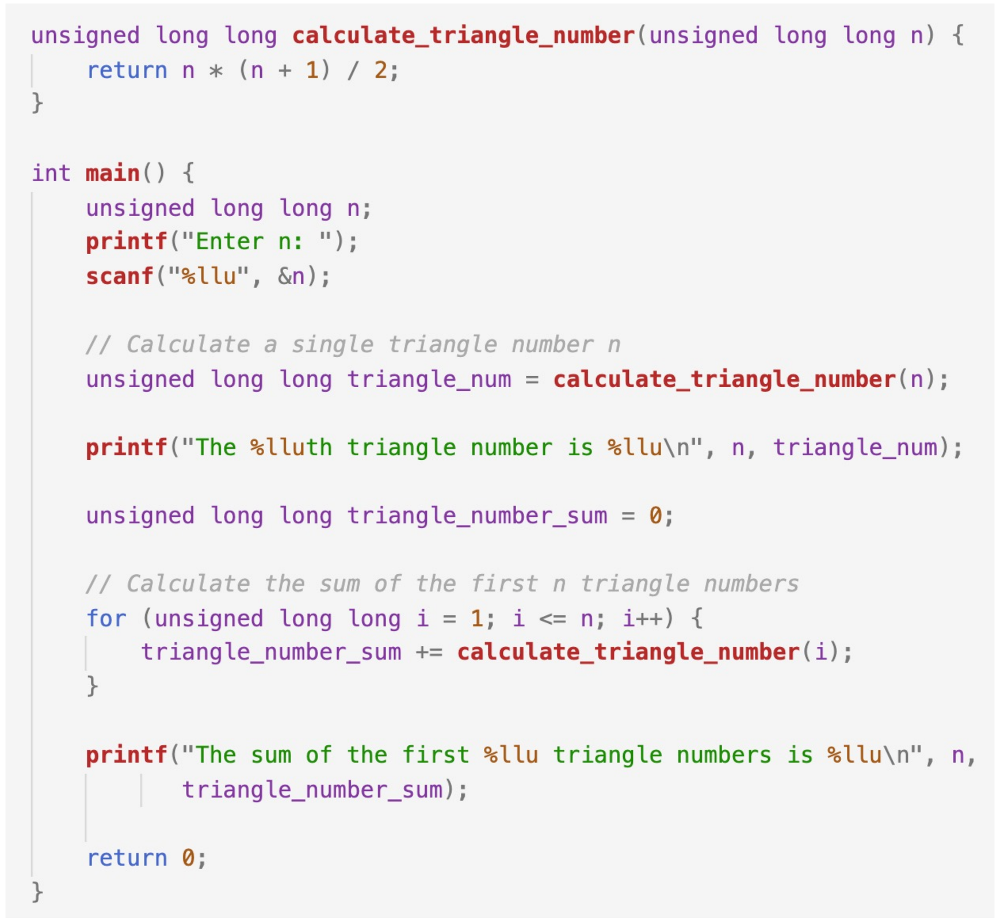
- Organize code
- Allow code reuse / avoid code duplication
- Abstraction
  - Distinction between external properties and internal details
  - You can use a function if you know what it does, even if you do not know how it does it

Why use function?



```c
#include <stdio.h>

int main() {
    unsigned long long n;
    printf("Enter n: ");
    scanf("%llu", &n);

    unsigned long long sum = 0;

    for (unsigned long long x = 1; x <= n; ++x) {
        sum += x;
    }

    printf("The %lluth triangle number is %llu\n", n, sum);
}
```

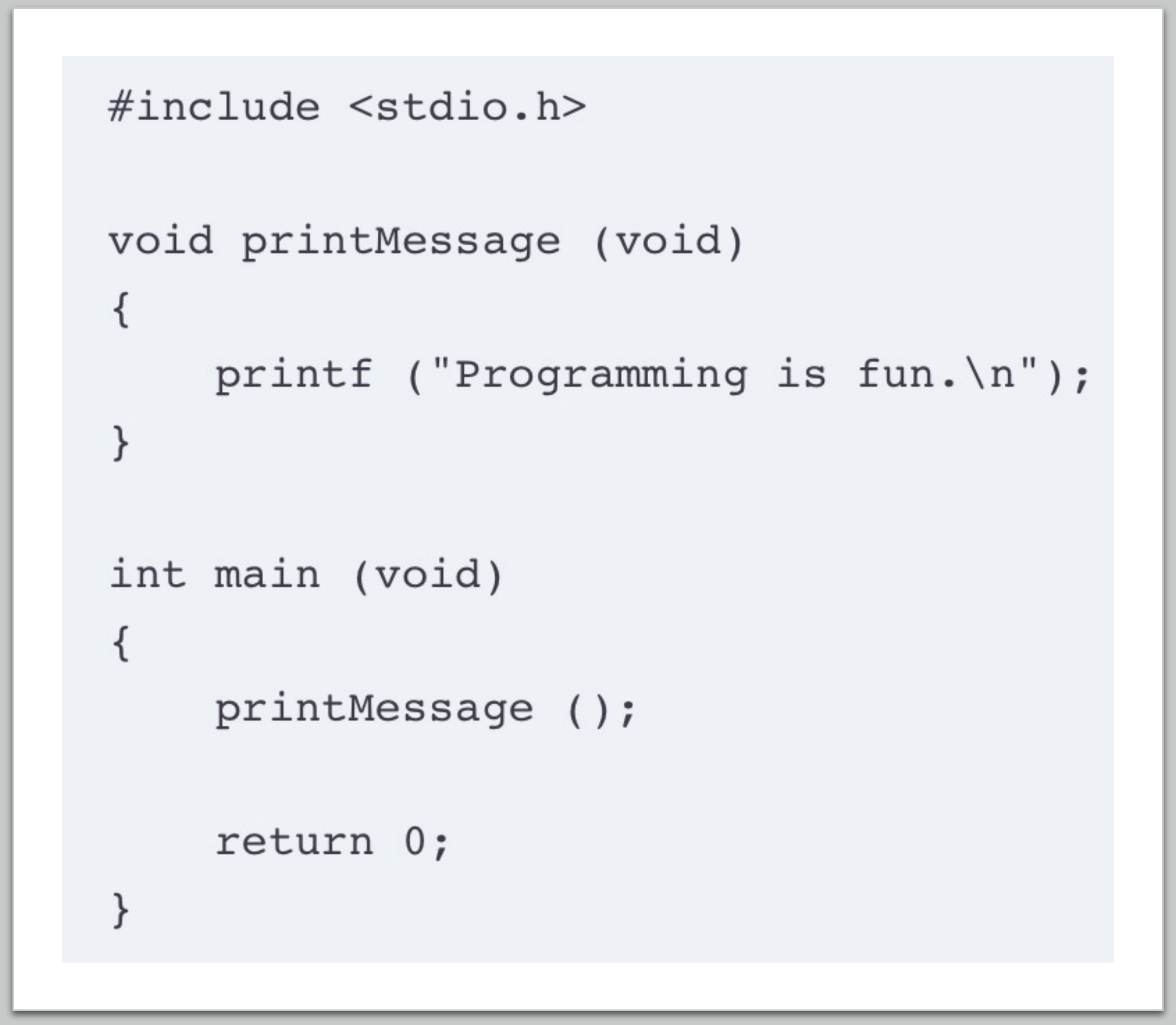
Why use function?



```c
unsigned long long calculate_triangle_number(unsigned long long n) {
    return n * (n + 1) / 2;
}

int main() {
    unsigned long long n;
    printf("Enter n: ");
    scanf("%llu", &n);

    // Calculate a single triangle number n
    unsigned long long triangle_num = calculate_triangle_number(n);

    printf("The %lluth triangle number is %llu\n", n, triangle_num);

    unsigned long long triangle_number_sum = 0;

    // Calculate the sum of the first n triangle numbers
    for (unsigned long long i = 1; i <= n; i++) {
        triangle_number_sum += calculate_triangle_number(i);
    }

    printf("The sum of the first %llu triangle numbers is %llu\n", n,
           triangle_number_sum);

    return 0;
}
```

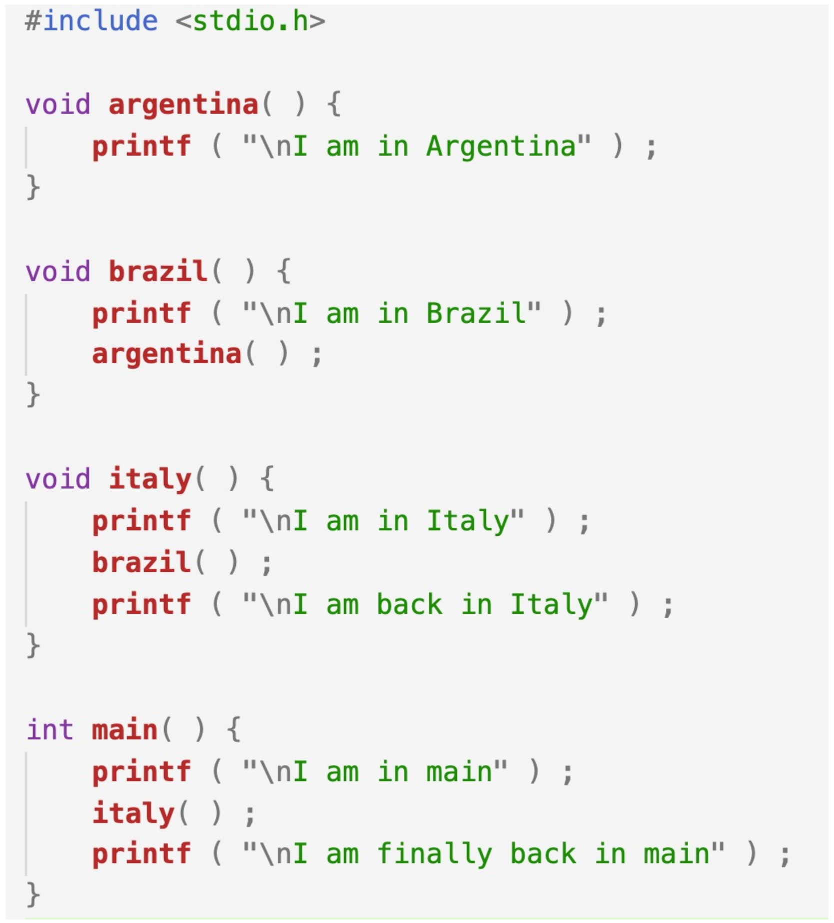
Function Call

- When a function call is executed, program execution is transferred directly to the indicated function
- After the routine is finished, program returns to main()
- Execution is returned to the program statement that immediately follows the call to the the function



```c
#include <stdio.h>

void printMessage (void)
{
    printf ("Programming is fun.\n");
}

int main (void)
{
    printMessage ();

    return 0;
}
```

Function Call



```c
#include <stdio.h>

void argentina( ) {
    printf ( "\nI am in Argentina" ) ;
}

void brazil( ) {
    printf ( "\nI am in Brazil" ) ;
    argentina( ) ;
}

void italy( ) {
    printf ( "\nI am in Italy" ) ;
    brazil( ) ;
    printf ( "\nI am back in Italy" ) ;
}

int main( ) {
    printf ( "\nI am in main" ) ;
    italy( ) ;
    printf ( "\nI am finally back in main" ) ;
}
```

- I am in main
- I am in Italy
- I am in Brazil
- I am in Argentina
- I am in back in Italy
- I am finally back in main

Predict the output

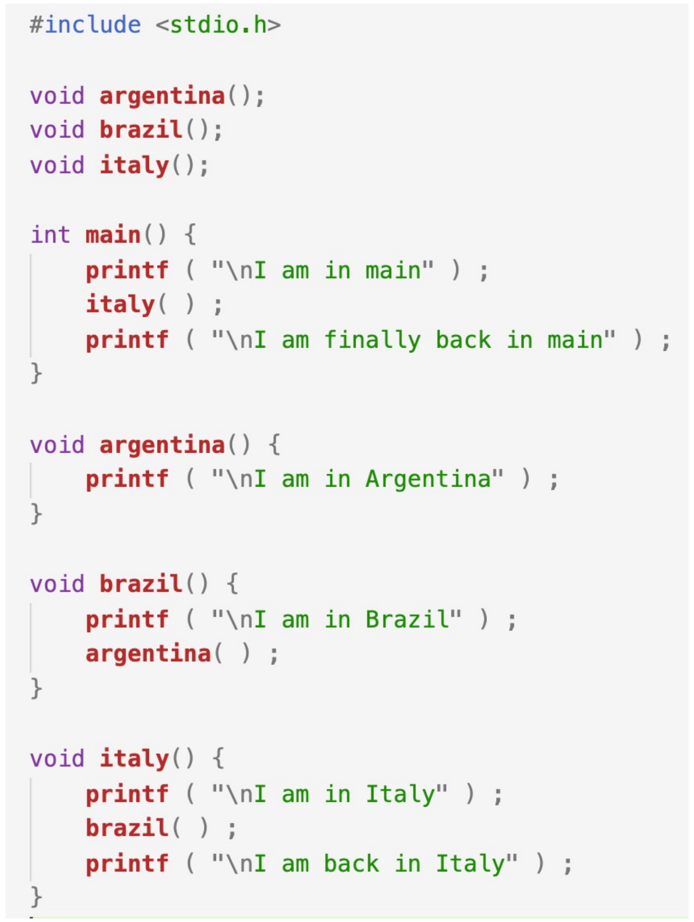
```c
#include <stdio.h>

int main( ) {
    printf ( "\nI am in main" ) ;
    italy( ) ;
    printf ( "\nI am finally back in main" ) ;
}

void argentina( ) {
    printf ( "\nI am in Argentina" ) ;
}

void brazil( ) {
    printf ( "\nI am in Brazil" ) ;
    argentina( ) ;
}

void italy( ) {
    printf ( "\nI am in Italy" ) ;
    brazil( ) ;
    printf ( "\nI am back in Italy" ) ;
}
```

```
example.c:5:5: error: implicit declaration of function 'italy' is invalid in C99
implicit-function-declaration]
    italy( ) ;
    ^
example.c:18:6: error: conflicting types for 'italy'
void italy( ) {
     ^
example.c:5:5: note: previous implicit declaration is here
    italy( ) ;
    ^
```

Function Prototype

- A function prototype defines the
  - return type
  - name
  - parameter list
  - **but not the body**
- This is all the information that the compiler needs to know about the function in order to determine valid calls to this function.
- The book declares prototypes within main()
  - More common practice is to define them above main()

Function Prototype



```c
#include <stdio.h>

void argentina();
void brazil();
void italy();

int main() {
    printf ( "\nI am in main" ) ;
    italy( ) ;
    printf ( "\nI am finally back in main" ) ;
}

void argentina() {
    printf ( "\nI am in Argentina" ) ;
}

void brazil() {
    printf ( "\nI am in Brazil" ) ;
    argentina( ) ;
}

void italy() {
    printf ( "\nI am in Italy" ) ;
    brazil( ) ;
    printf ( "\nI am back in Italy" ) ;
}
```

Paycheck Function

- Sample output
  - Enter hours worked: 45.5
  - Enter the rate of pay: 10
  - You earned $482.50
- Overtime
  - If an employee works more than 40 hours in a week they will receive overtime pay which is 1.5 times the regular pay

Paycheck Function

Your grade for this assignment will be out out of 15 points:

- 4 points - calculate_pay() function implementation
- 1 points - a function **prototype was used** for calculate_pay()
- 1 points - calculate_pay() has correct function name and parameters
- 1 points - main() accepts user input
- 1 point - output uses a dollar sign and two decimal digits of precision
- 7 points - passes the tests