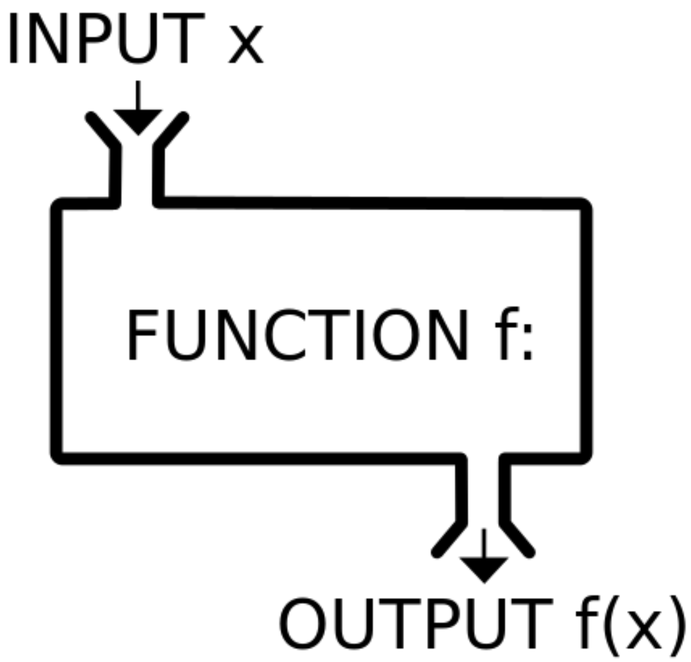


INPUT x

FUNCTION f:

OUTPUT f(x)

Week 5 - Lectures 1, 2

Functions

Edited by: Heshan Du

Autumn 2023

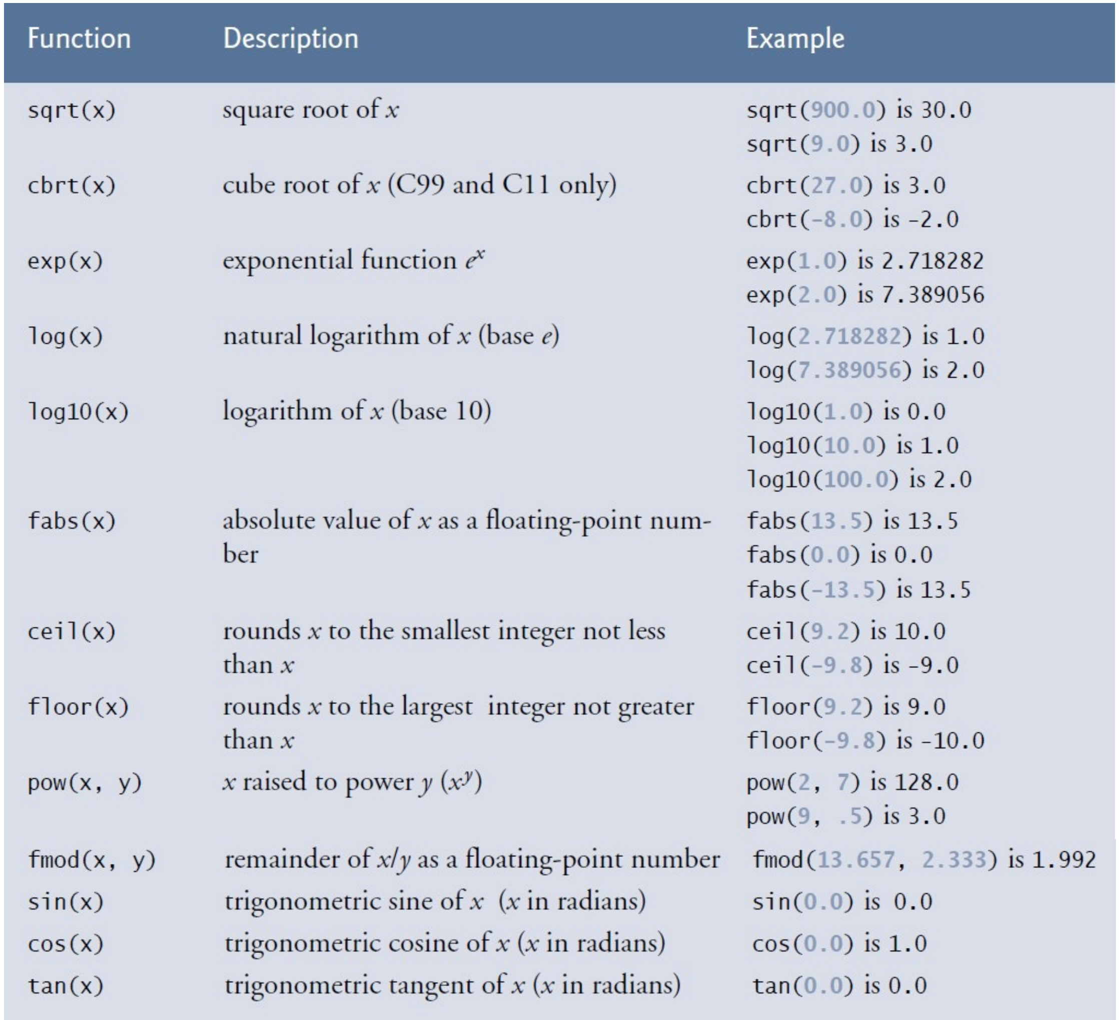
University of Nottingham
UK | CHINA | MALAYSIA

Overview

- **Function**
- Function Call Stack
- Passing data by values and references

Prepacked Functions

C standard libraries e.g., printf, scanf, pow

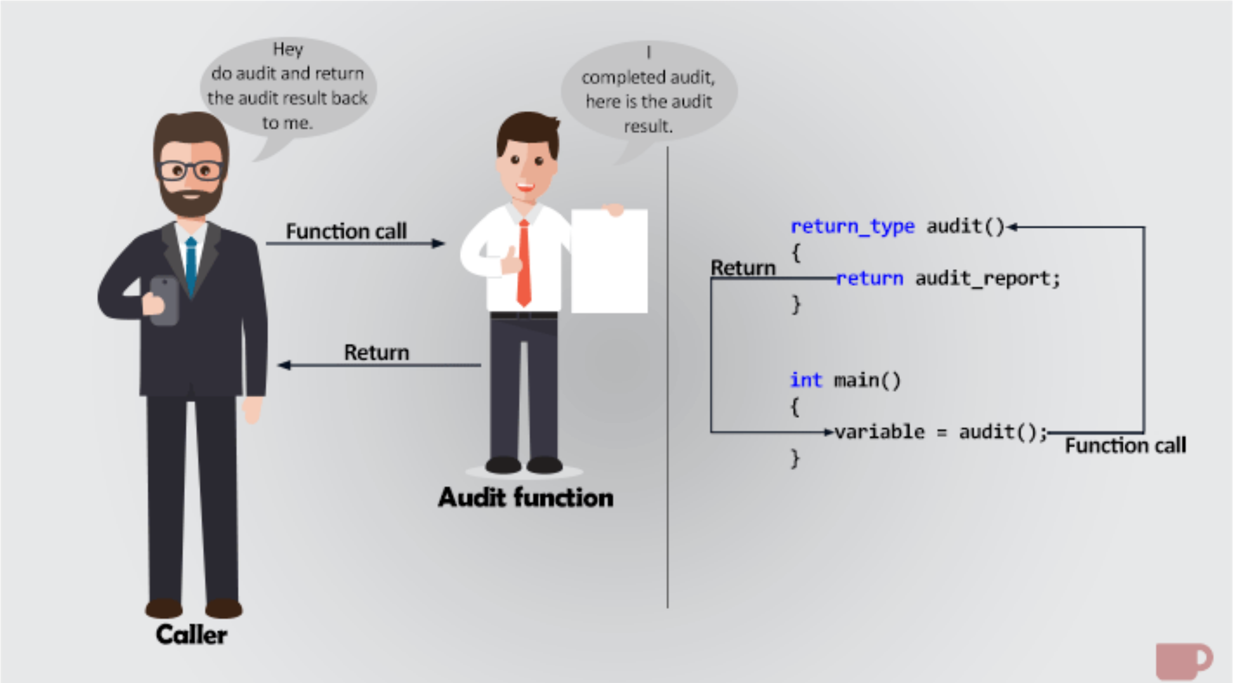


Function	Description	Example
`sqrt(x)`	square root of x	`sqrt(900.0)` is 30.0 `sqrt(9.0)` is 3.0
`cbrt(x)`	cube root of x (C99 and C11 only)	`cbrt(27.0)` is 3.0 `cbrt(-8.0)` is -2.0
`exp(x)`	exponential function e^x	`exp(1.0)` is 2.718282 `exp(2.0)` is 7.389056
`log(x)`	natural logarithm of x (base e)	`log(2.718282)` is 1.0 `log(7.389056)` is 2.0
`log10(x)`	logarithm of x (base 10)	`log10(1.0)` is 0.0 `log10(10.0)` is 1.0 `log10(100.0)` is 2.0
`fabs(x)`	absolute value of x as a floating-point number	`fabs(13.5)` is 13.5 `fabs(0.0)` is 0.0 `fabs(-13.5)` is 13.5
`ceil(x)`	rounds x to the smallest integer not less than x	`ceil(9.2)` is 10.0 `ceil(-9.8)` is -9.0
`floor(x)`	rounds x to the largest integer not greater than x	`floor(9.2)` is 9.0 `floor(-9.8)` is -10.0
`pow(x, y)`	x raised to power y (x^y)	`pow(2, 7)` is 128.0 `pow(9, .5)` is 3.0
`fmod(x, y)`	remainder of x/y as a floating-point number	`fmod(13.657, 2.333)` is 1.992
`sin(x)`	trigonometric sine of x (x in radians)	`sin(0.0)` is 0.0
`cos(x)`	trigonometric cosine of x (x in radians)	`cos(0.0)` is 1.0
`tan(x)`	trigonometric tangent of x (x in radians)	`tan(0.0)` is 0.0

Source: Deitel and Deiltel (2016). C How to Program with an Introduction to C++ (8th Ed.). Pearson.

Function

- Same way you have been using printf or scanf.
- Hide (encapsulate) information from user.



Hey do audit and return the audit result back to me.
I completed audit, here is the audit result.
Function call
Return
Audit function
Caller
Return
Function call

```
return_type audit()
{
    return audit_report;
}

int main()
{
    variable = audit();
}
```

Source: https://codeforwin.org/2017/12/return-multiple-value-from-function-c-programming.html

Function (2)

- An *independent* block of code that performs a specific task when called, and it may **return a value** to the calling program.
  - e.g., pow(), fmod().

- If you want, you can ignore return values from functions by just not using or storing them.
  - e.g., printf(), scanf().

Function (3)

- Each function is essentially a small program, with its own **variables** and **statements**.
- Functions must be defined or declared before they are used.
  - It has a name
  - zero or one return value
  - A function body, which includes the code
  - zero or more parameters i.e., argument.
- e.g., int main (void)

  int function(int x, char c)

Benefits of using Functions

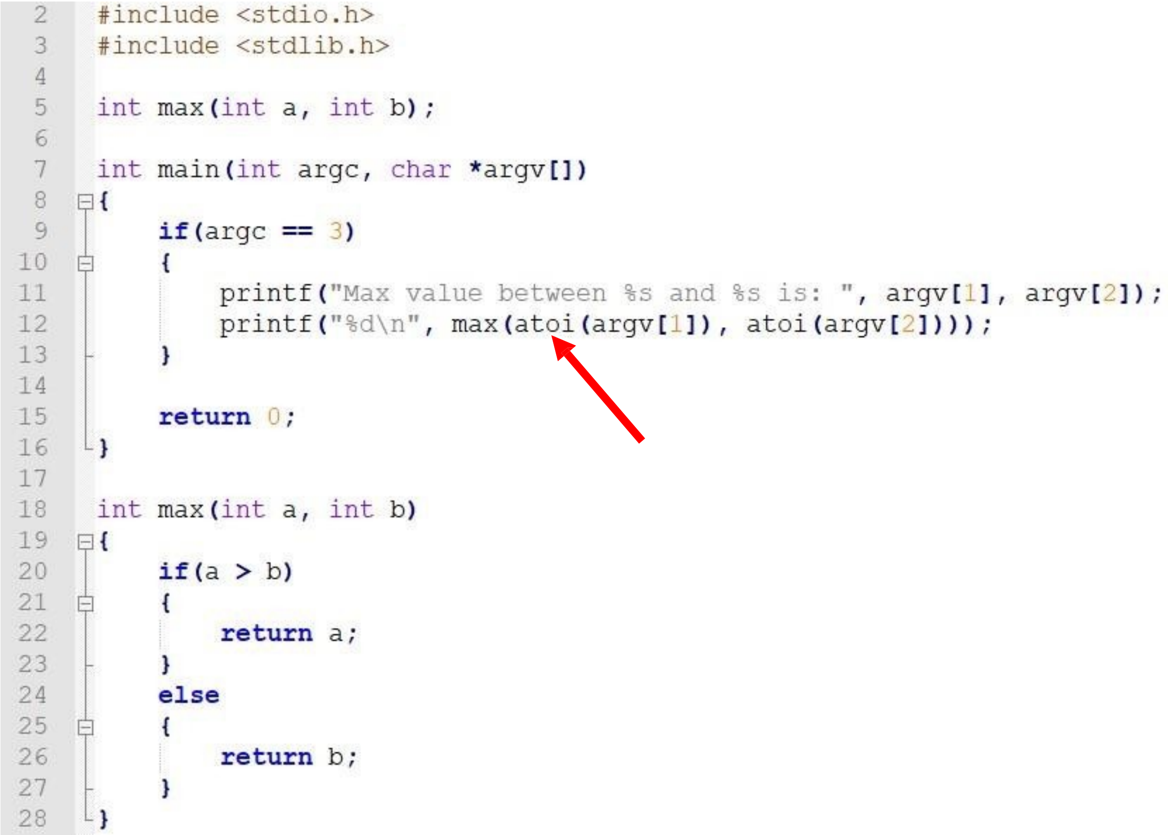
- The *divide-and-conquer* approach makes program development more manageable.

- *software reusability*—using existing functions as *building blocks* to create new programs.

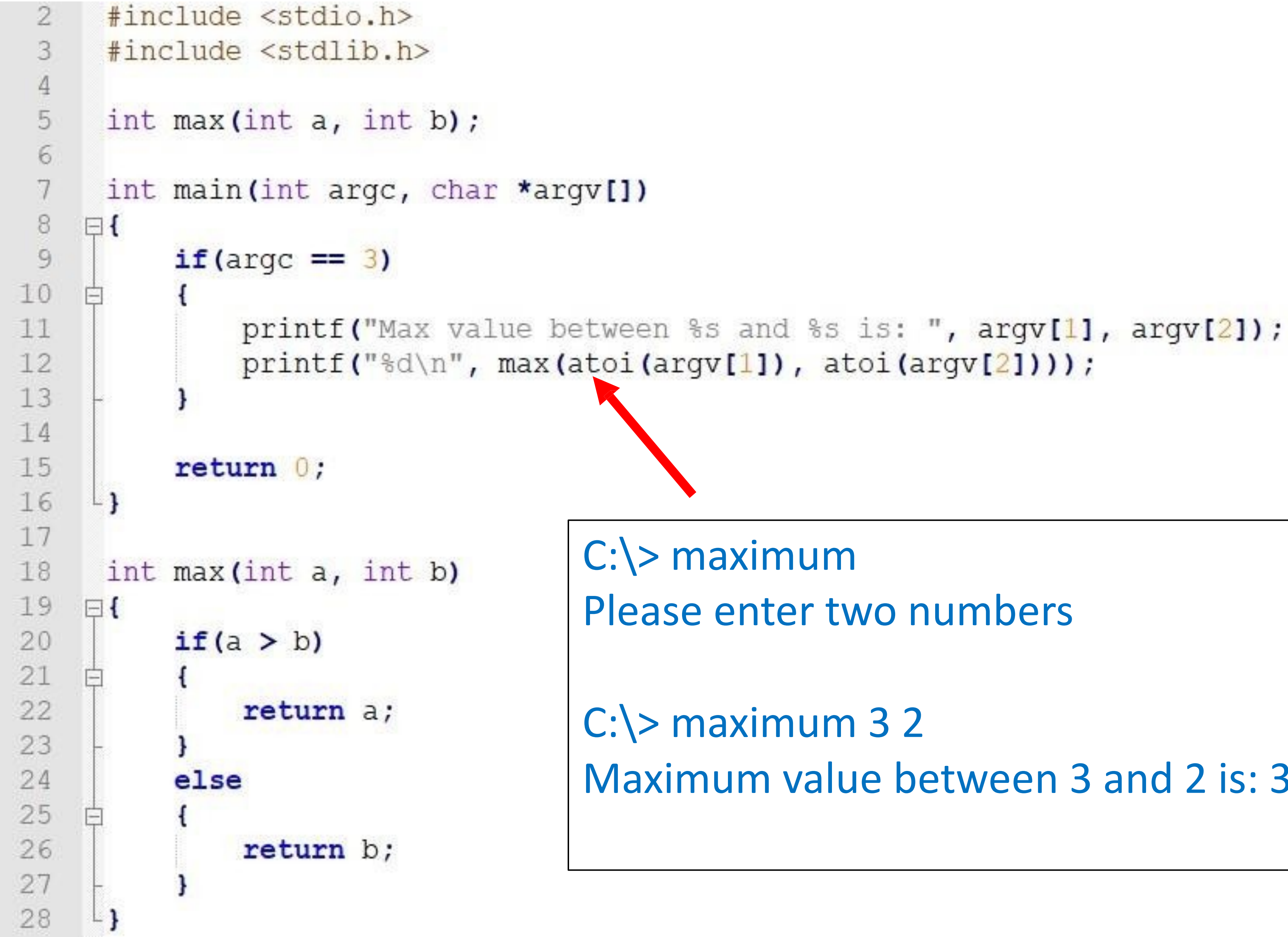
- Reduce repeated code in a program.

An Example Function

What does this program do?



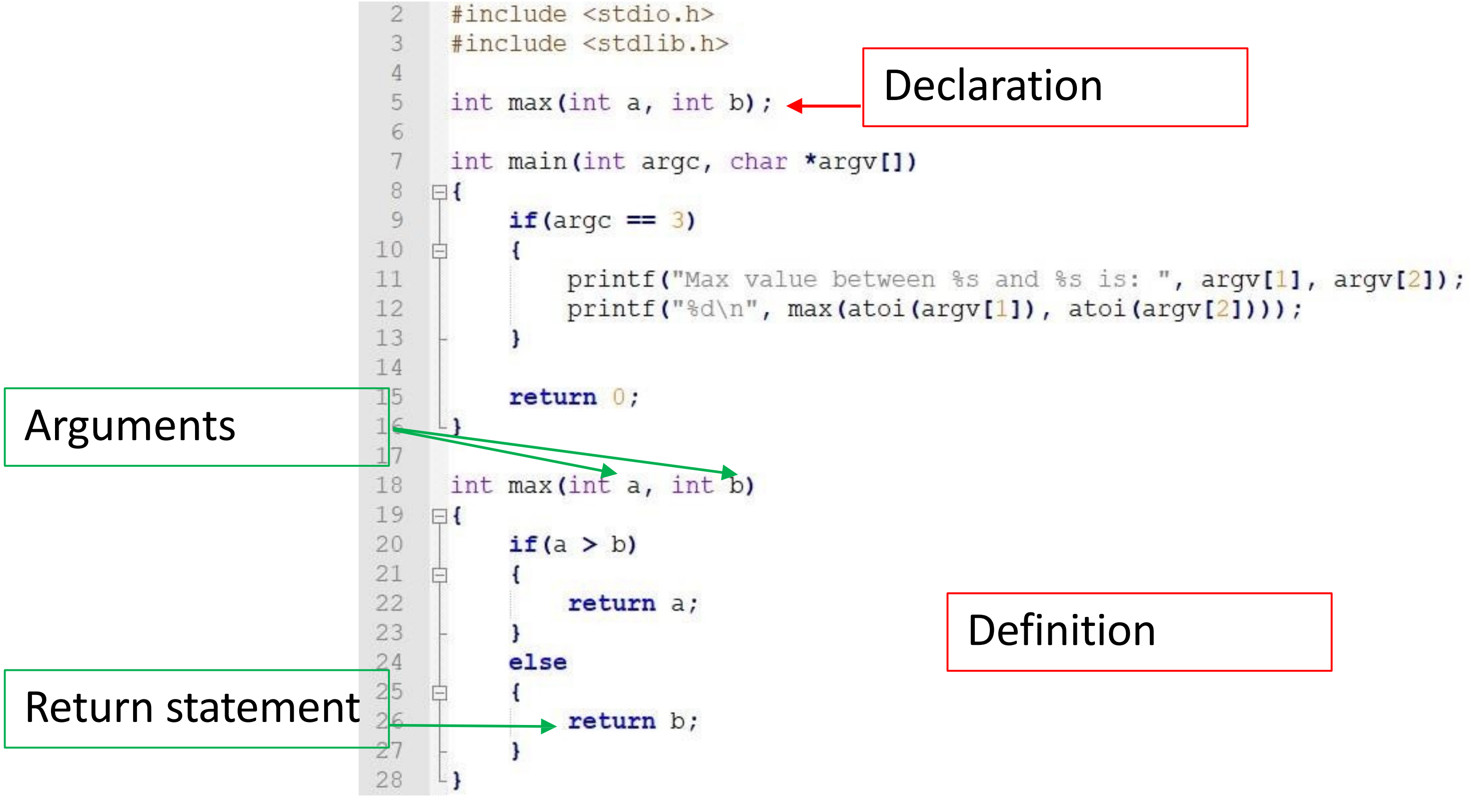
```c
#include <stdio.h>
#include <stdlib.h>

int max(int a, int b);

int main(int argc, char *argv[])
{
    if(argc == 3)
    {
        printf("Max value between %s and %s is: ", argv[1], argv[2]);
        printf("%d\n", max(atoi(argv[1]), atoi(argv[2])));
    }

    return 0;
}

int max(int a, int b)
{
    if(a > b)
    {
        return a;
    }
    else
    {
        return b;
    }
}
```



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

int max(int a, int b);

int main(int argc, char *argv[])
{
    if(argc == 3)
    {
        printf("Max value between %s and %s is: ", argv[1], argv[2]);
        printf("%d\n", max(atoi(argv[1]), atoi(argv[2])));
    }

    return 0;
}

int max(int a, int b)
{
    if(a > b)
    {
        return a;
    }
    else
    {
        return b;
    }
}
```

```
C:\> maximum
Please enter two numbers

C:\> maximum 3 2
Maximum value between 3 and 2 is: 3
```

Function (6)



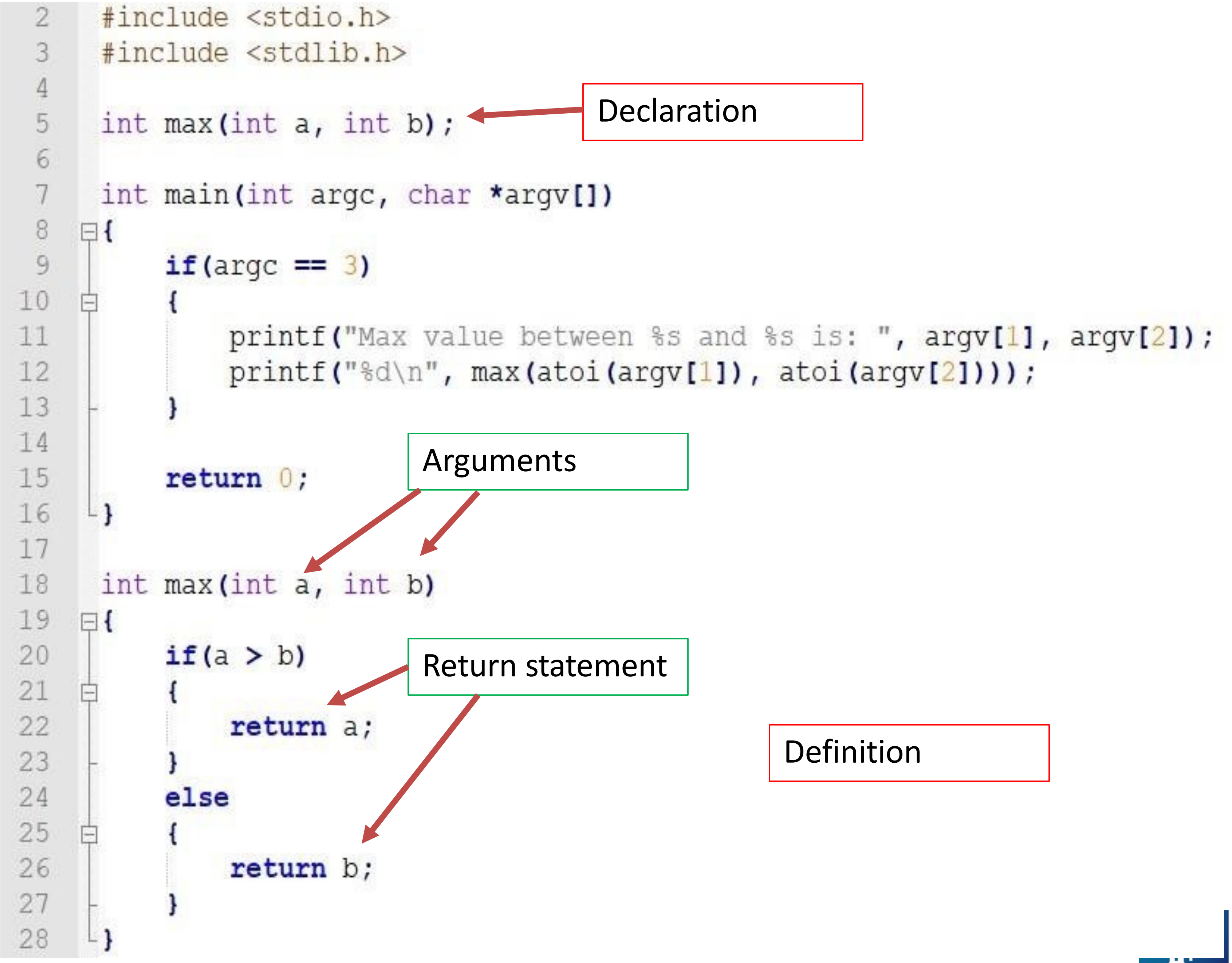
```c
#include <stdio.h>
#include <stdlib.h>

int max(int a, int b);

int main(int argc, char *argv[])
{
    if(argc == 3)
    {
        printf("Max value between %s and %s is: ", argv[1], argv[2]);
        printf("%d\n", max(atoi(argv[1]), atoi(argv[2])));
    }

    return 0;
}

int max(int a, int b)
{
    if(a > b)
    {
        return a;
    }
    else
    {
        return b;
    }
}
```

Declaration

Arguments

Definition

Return statement



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

int max(int a, int b);

int main(int argc, char *argv[])
{
    if(argc == 3)
    {
        printf("Max value between %s and %s is: ", argv[1], argv[2]);
        printf("%d\n", max(atoi(argv[1]), atoi(argv[2])));
    }

    return 0;
}

int max(int a, int b)
{
    if(a > b)
    {
        return a;
    }
    else
    {
        return b;
    }
}
```

Declaration

Arguments

Return statement

Definition

Function Declaration

Parameters are separated by commas.

Use “void” if no parameter or use an empty bracket.

- **return_type** function_name(parameter_list)**;**

Return at most one value, if return type is missing, the function is presumed to return type int.

If returns nothing, use “void”.

e.g., **void** show(**char** ch);
double show(**int** a, **float** b);

- Declare in header files
  - If you use multiple ".c" source files, write a header file with declarations of functions to use in the other files.
- For library functions, use #include …

e.g., printf(), scanf() use #include <stdio.h>

Function Definition

```
return_type function_name(parameter_list)
{
    /* Function body */
}
```

No semi-colon at the end!

```
void test();
int main(){
    test();
    return 0;
}
void test(){
    /* Function body. */
    printf("In\n");
}
```

The function’s body is executed only if the function is called somewhere in the program.

The function terminates if either an exit statement (i.e., return) is called or its last statement is executed.

return Statement

To terminate immediately the execution of a function and continue from the point where the function was called.

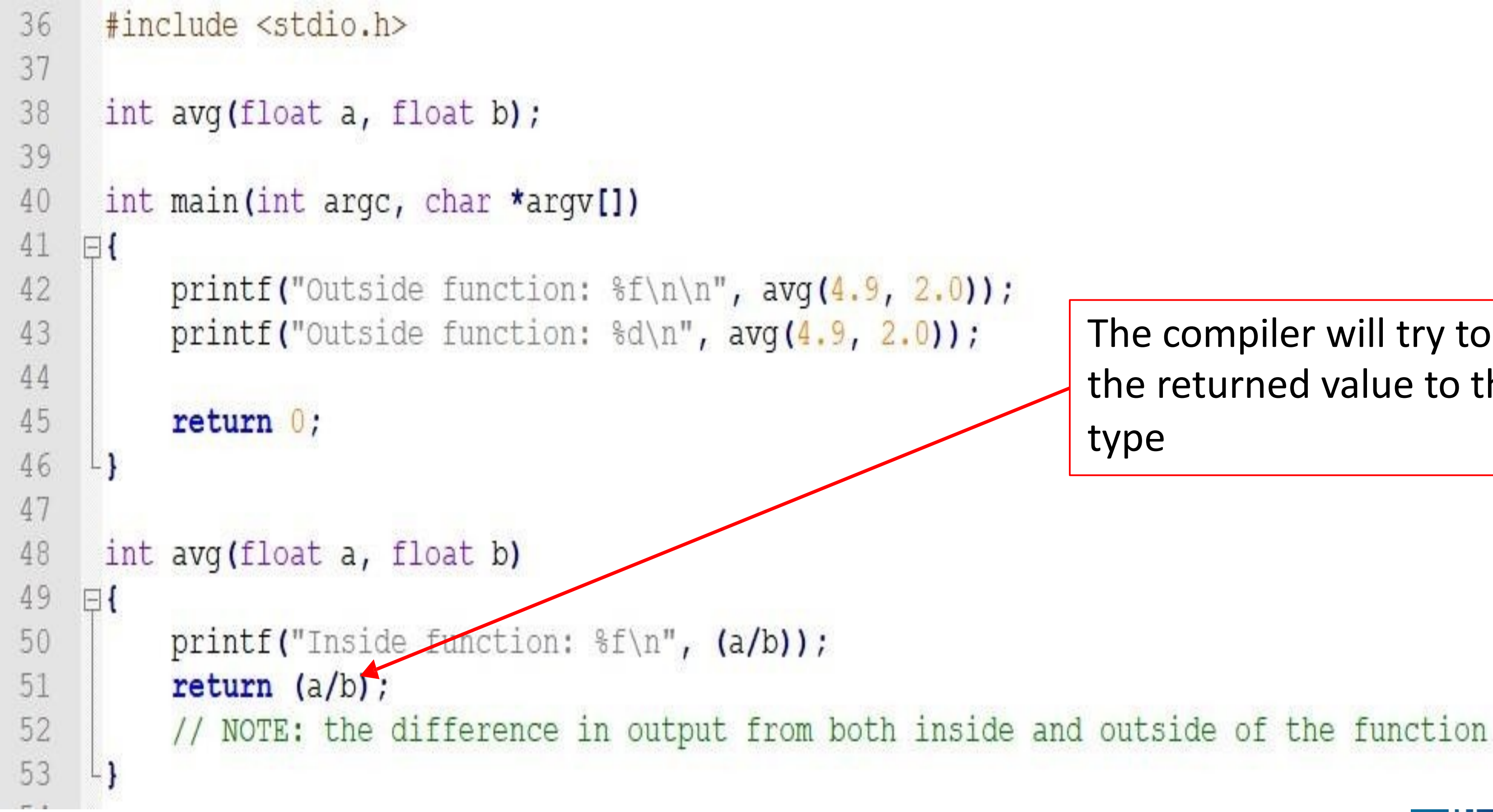
```
int main(void)
{
       while(1){
           printf("Enter number: ");
           scanf("%d", &num);
           if(num == 2)
               return 0; /* Program termination. */
           else
               printf("Num = %d\n", num);
        }
}
```

Do not do this!!

Indicates normal program termination

return Statement (2)

Make sure the type of the returned value matches the function’s return type.

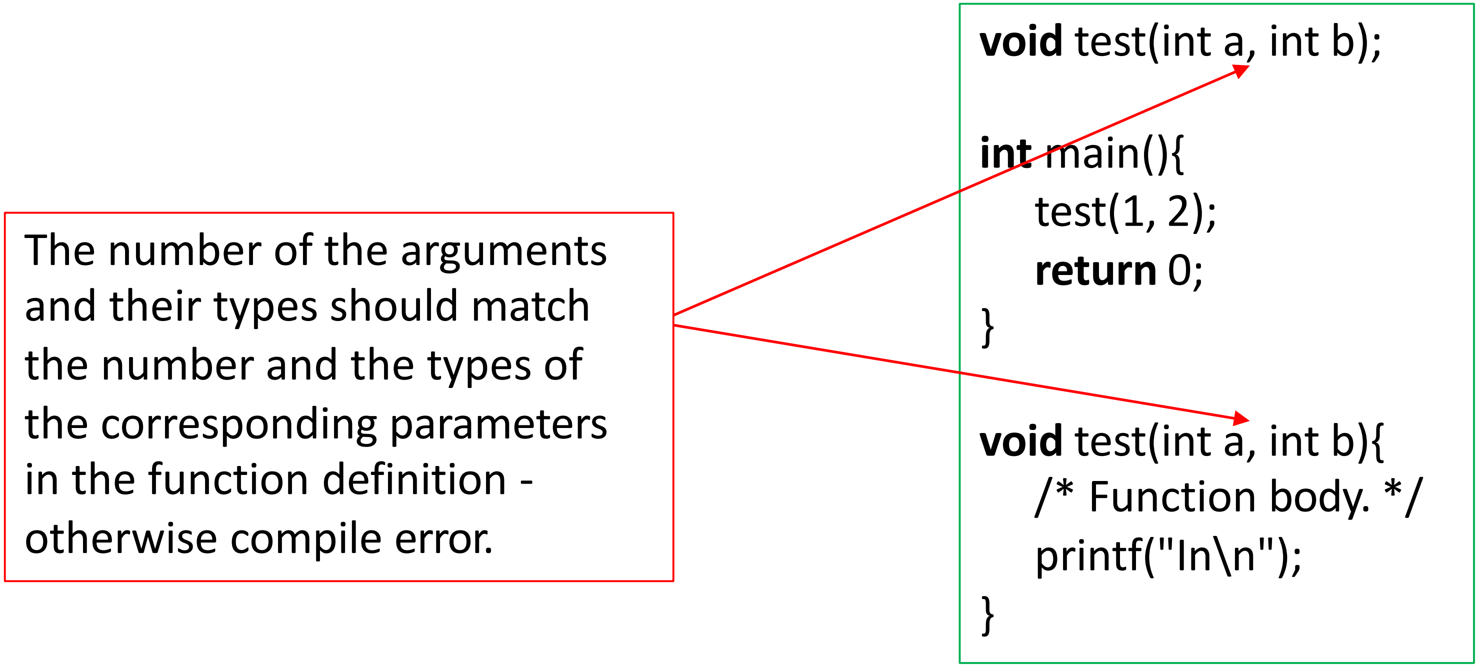


```
#include <stdio.h>

int avg(float a, float b);

int main(int argc, char *argv[])
{
    printf("Outside function: %f\n\n", avg(4.9, 2.0));
    printf("Outside function: %d\n", avg(4.9, 2.0));

    return 0;
}

int avg(float a, float b)
{
    printf("Inside function: %f\n", (a/b));
    return (a/b);
    // NOTE: the difference in output from both inside and outside of the function
}
```

The compiler will try to convert the returned value to the return type

Argument

The argument can be any valid expression, such as constant, variable, math, or logical expression, even another function with a return value.



The number of the arguments and their types should match the number and the types of the corresponding parameters in the function definition - otherwise compile error.

```
void test(int a, int b);

int main(){
    test(1, 2);
    return 0;
}

void test(int a, int b){
    /* Function body. */
    printf("In\n");
}
```

Example 1

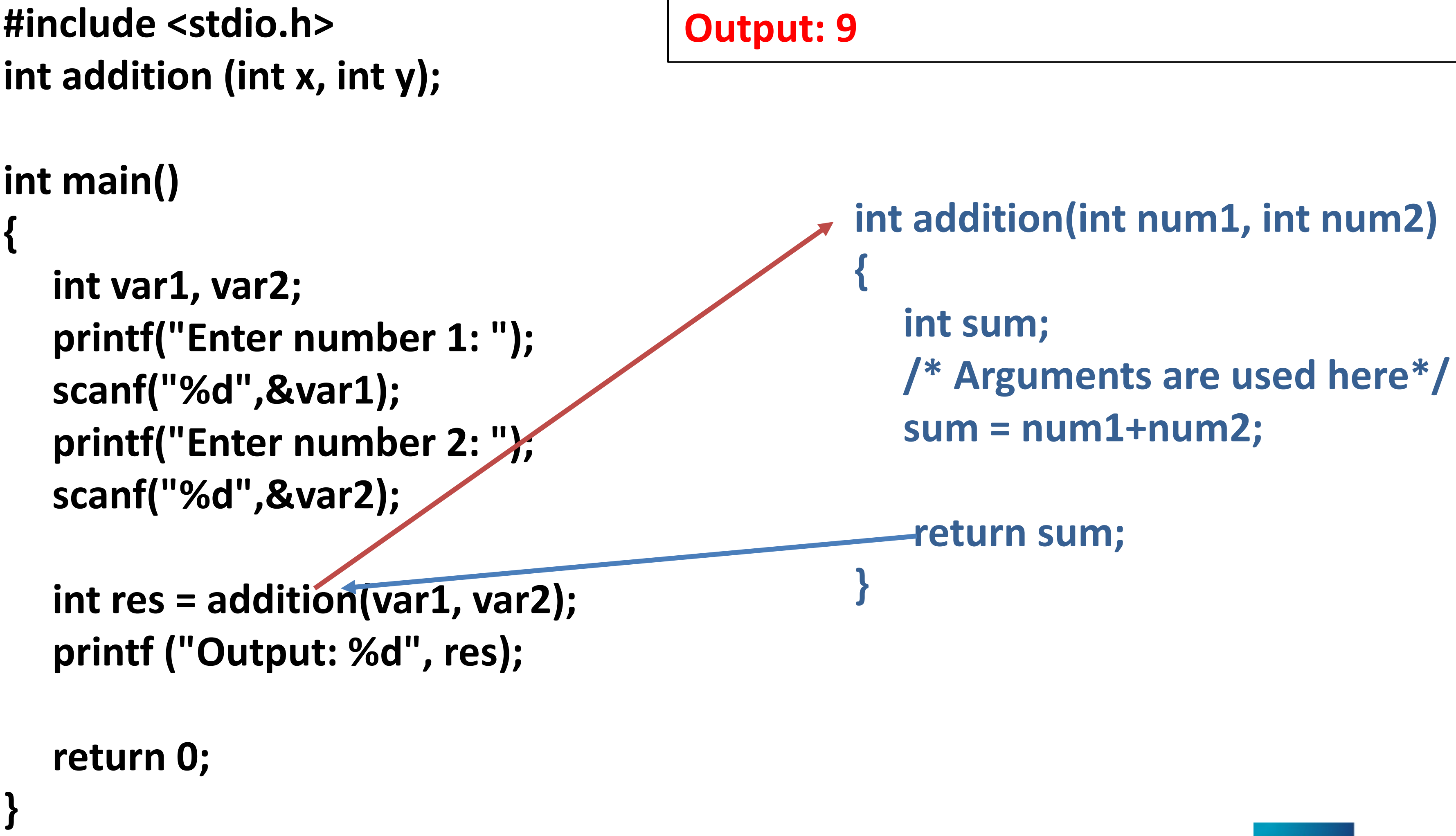
Output:

```
Hi
My name is John
How are you guys?
```

```c
#include <stdio.h>
void introduction();

int main()
{
    /*calling function*/
    introduction();
    return 0;
}


void introduction()
{
   printf("Hi\n");
   printf("My name is John\n");
   printf("How are you guys?");

   /* There is no return statement inside this function,
       since its return type is void*/
}
```

Example 2

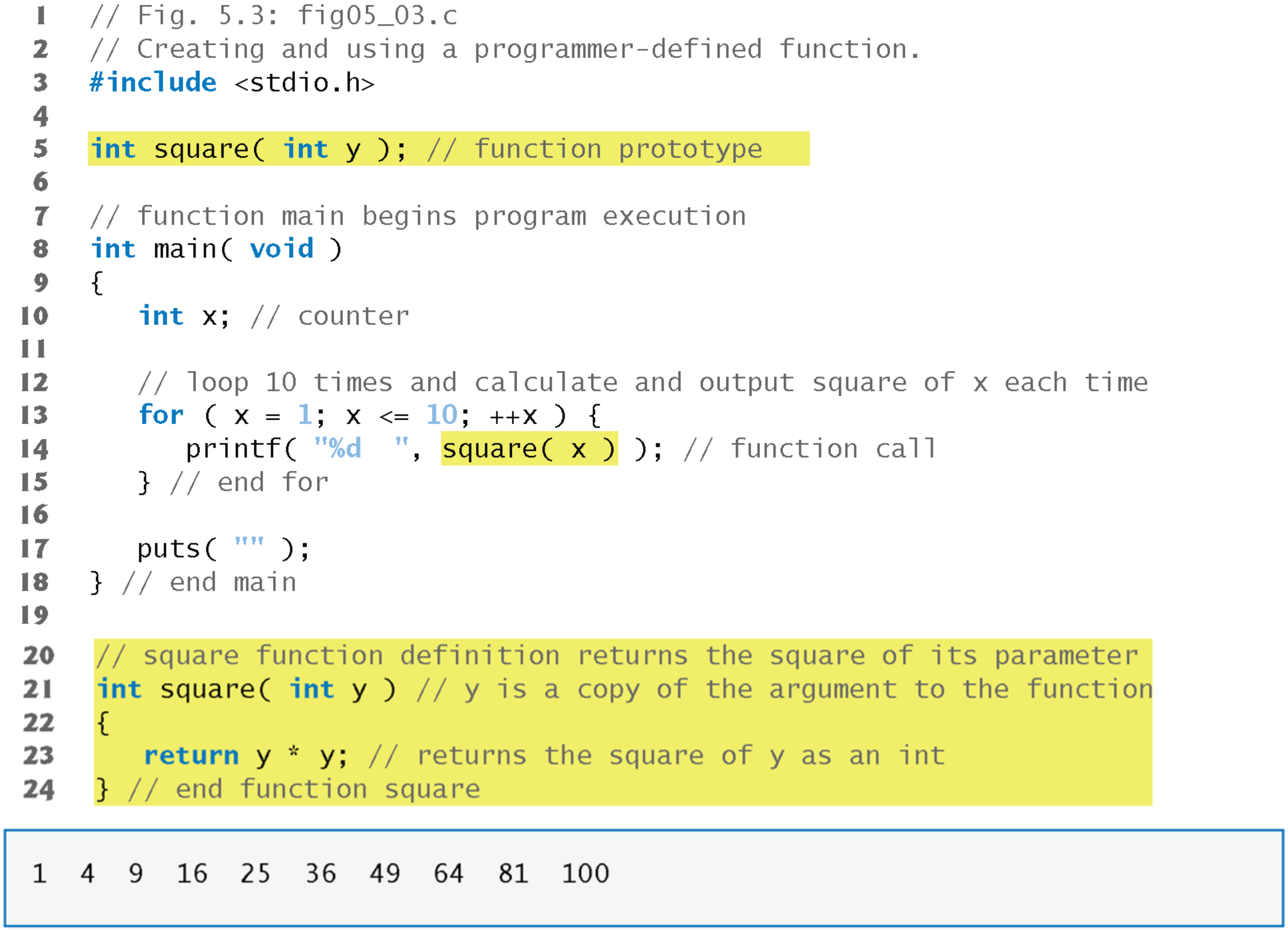
```
Enter number 1: 5
Enter number 2: 4
Output: 9
```



```
#include <stdio.h>
int addition (int x, int y);

int main()
{
    int var1, var2;
    printf("Enter number 1: ");
    scanf("%d",&var1);
    printf("Enter number 2: ");
    scanf("%d",&var2);

    int res = addition(var1, var2);
    printf ("Output: %d", res);

    return 0;
}
```

```
int addition(int num1, int num2)
{
    int sum;
    /* Arguments are used here*/
    sum = num1+num2;

    return sum;
}
```

Example 3



```c
// Fig. 5.3: fig05_03.c
// Creating and using a programmer-defined function.
#include <stdio.h>

int square( int y ); // function prototype

// function main begins program execution
int main( void )
{
   int x; // counter

   // loop 10 times and calculate and output square of x each time
   for ( x = 1; x <= 10; ++x ) {
      printf( "%d  ", square( x ) ); // function call
   } // end for

   puts( "" );
} // end main

// square function definition returns the square of its parameter
int square( int y ) // y is a copy of the argument to the function
{
   return y * y; // returns the square of y as an int
} // end function square
```

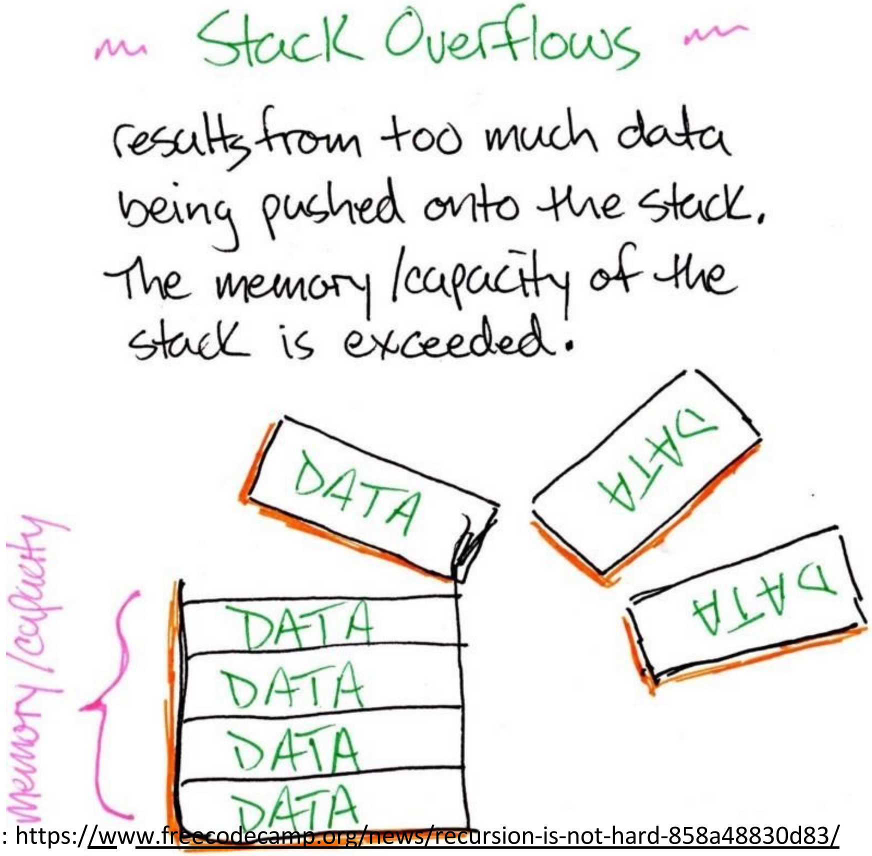
```
1   4   9   16   25   36   49   64   81   100
```

Overview

- Function
- **Function Call Stack**
- Passing data by values and references

Function Call Stack

- The compiler allocates memory (i.e., stack) to store the function's parameters and the variables when the function is called.
- Once it is terminated, the memory is automatically deallocated.



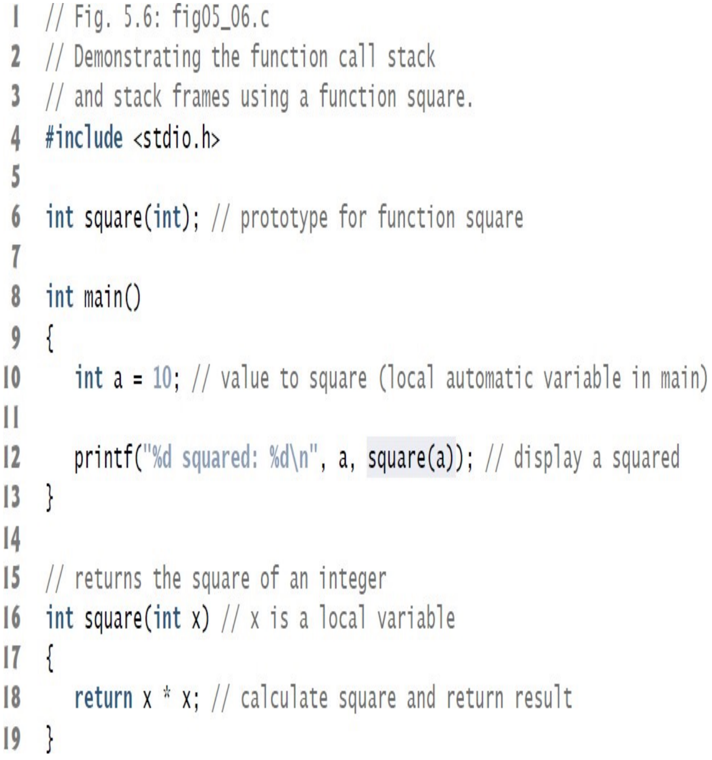
Stack Overflows
results from too much data being pushed onto the stack. The memory /capacity of the stack is exceeded.
DATA
DATA
DATA
DATA
DATA
DATA
DATA
DATA
memory /capacity

Source: https://www.freecodecamp.org/news/recursion-is-not-hard-858a48830d83/

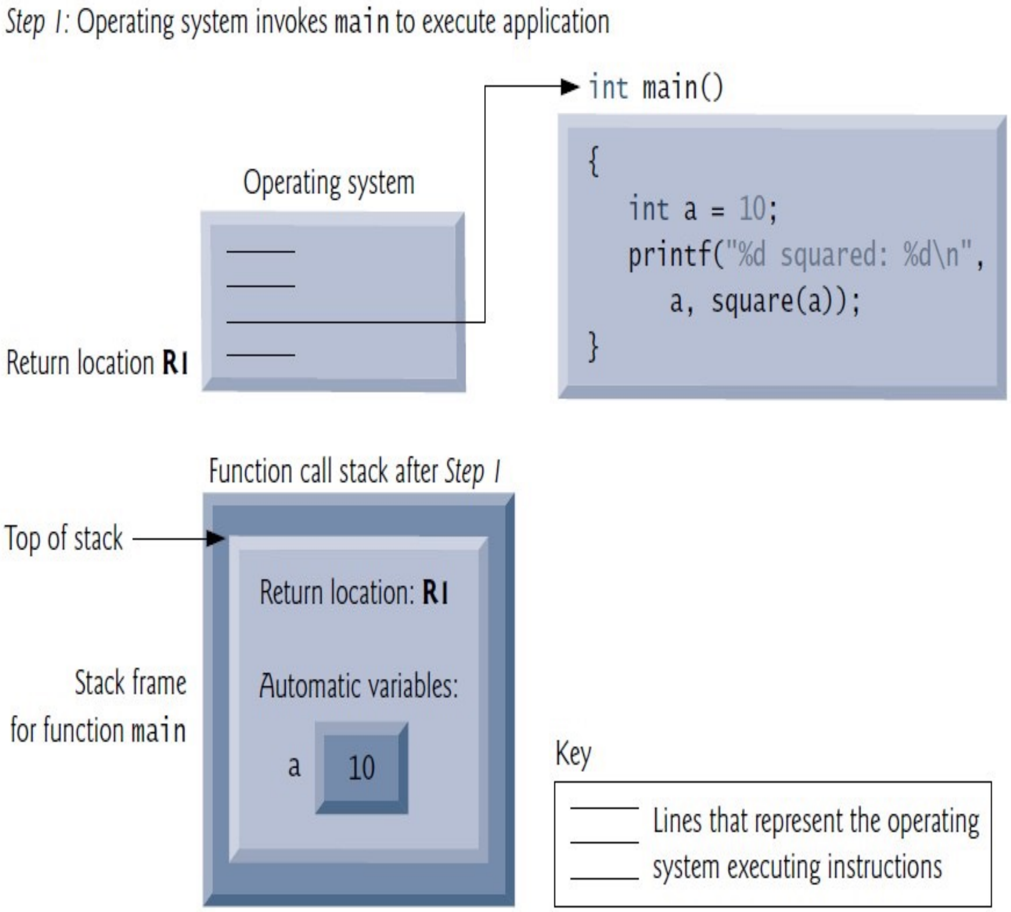
Function Call Stack (2)

- Stacks are known as *last-in, first-out* (LIFO) data structures—the last item pushed (inserted) on the stack is the first item popped (removed) from the stack.
- The function call stack supports the creation, maintenance and destruction of automatic variables of each called function.

Function Call Stack (3)



```c
// Fig. 5.6: fig05_06.c
// Demonstrating the function call stack
// and stack frames using a function square.
#include <stdio.h>

int square(int); // prototype for function square

int main()
{
   int a = 10; // value to square (local automatic variable in main)

   printf("%d squared: %d\n", a, square(a)); // display a squared
}

// returns the square of an integer
int square(int x) // x is a local variable
{
   return x * x; // calculate square and return result
}
```



Step 1: Operating system invokes main to execute application

Operating system

Return location R1

```c
int main()
{
   int a = 10;
   printf("%d squared: %d\n",
      a, square(a));
}
```

Function call stack after *Step 1*

Top of stack

Stack frame for function main

Return location: R1

Automatic variables:

a	10

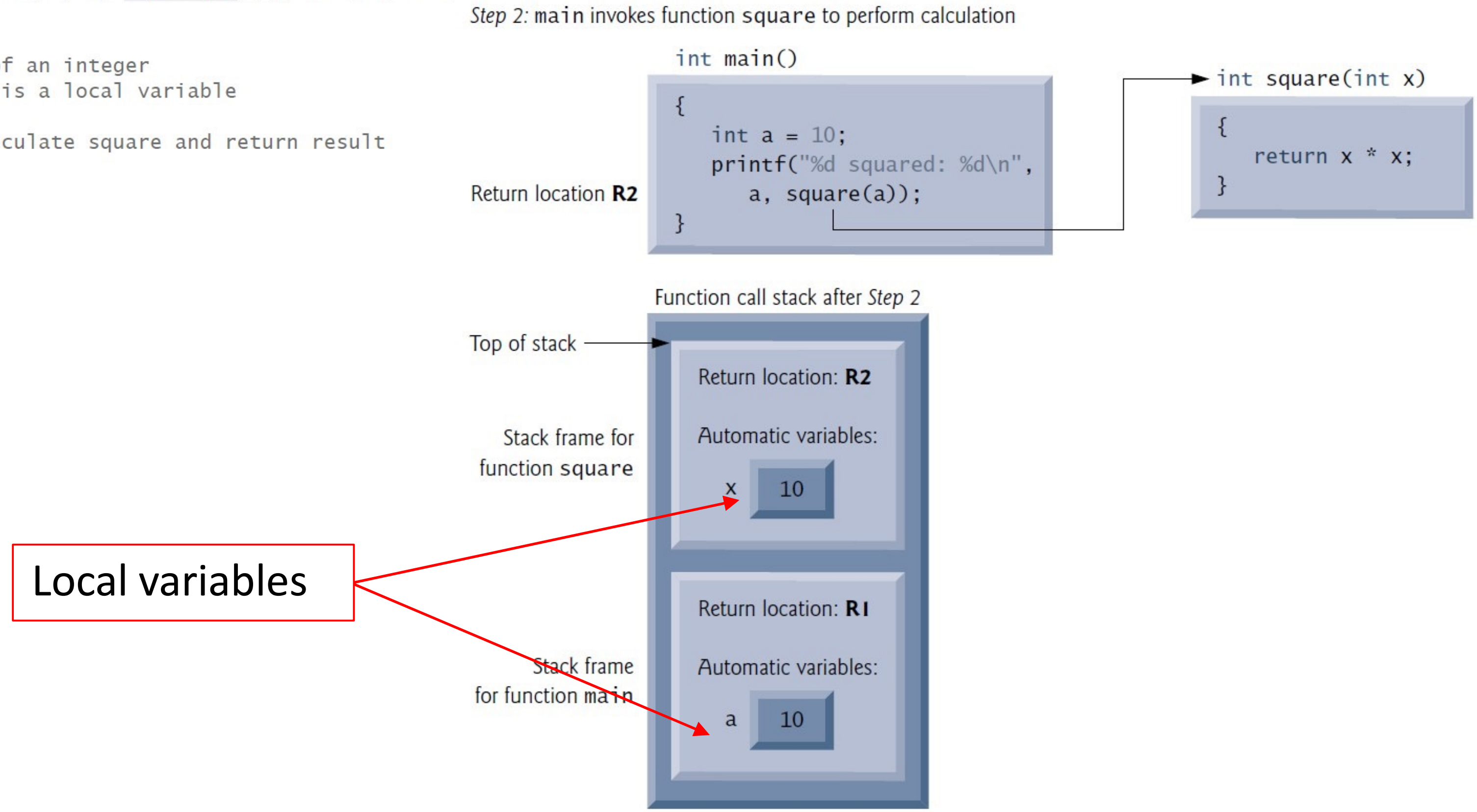
Key

Lines that represent the operating system executing instructions

Source: Deitel and Deiltel (2016). C How to Program with an Introduction to C++ (8th Ed.). Pearson.

Function Call Stack (4)

```c
// Fig. 5.6: fig05_06.c
// Demonstrating the function call stack
// and stack frames using a function square.
#include <stdio.h>

int square(int); // prototype for function square

int main()
{
   int a = 10; // value to square (local automatic variable in main)

   printf("%d squared: %d\n", a, square(a)); // display a squared
}

// returns the square of an integer
int square(int x) // x is a local variable
{
   return x * x; // calculate square and return result
}
```



Step 2: main invokes function square to perform calculation

```c
int main()
{
   int a = 10;
   printf("%d squared: %d\n",
      a, square(a));
}
```

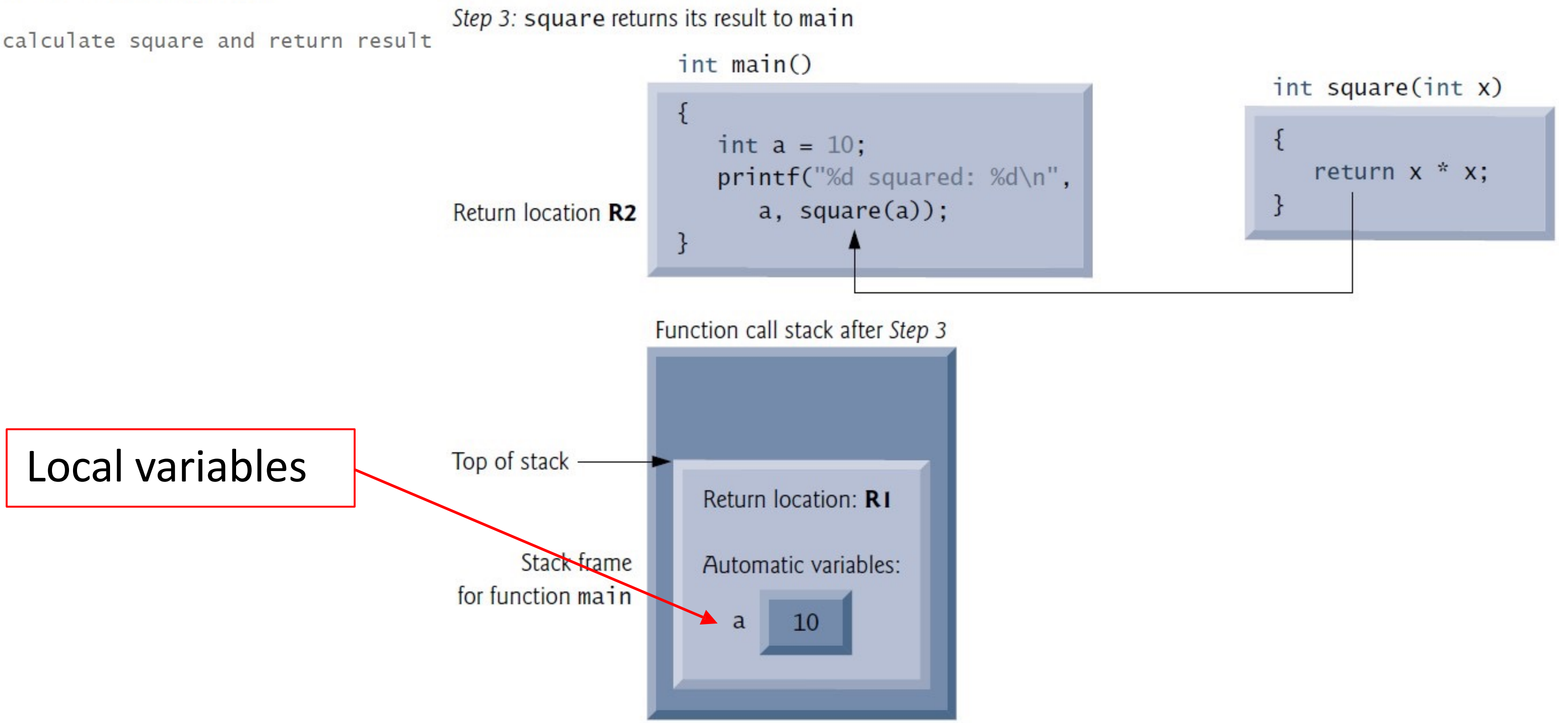
```c
int square(int x)
{
   return x * x;
}
```

Local variables

Source: Deitel and Deiltel (2016). C How to Program with an Introduction to C++ (8th Ed.). Pearson.

Function Call Stack (5)

```c
// Fig. 5.6: fig05_06.c
// Demonstrating the function call stack
// and stack frames using a function square.
#include <stdio.h>

int square(int); // prototype for function square

int main()
{
   int a = 10; // value to square (local automatic variable in main)

   printf("%d squared: %d\n", a, square(a)); // display a squared
}

// returns the square of an integer
int square(int x) // x is a local variable
{
   return x * x; // calculate square and return result
}
```



Step 3: `square` returns its result to `main`

```c
int main()
{
   int a = 10;
   printf("%d squared: %d\n",
      a, square(a));
}
```

```c
int square(int x)
{
   return x * x;
}
```

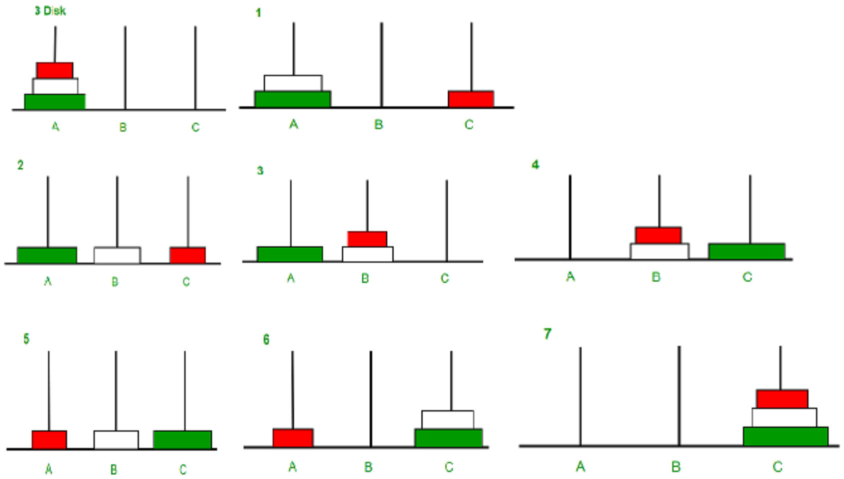
Source: Deitel and Deiltel (2016). C How to Program with an Introduction to C++ (8th Ed.). Pearson.

Stack Overflow

- A recursive function is a function which calls itself.
- E.g., factorial, tower of hanoi

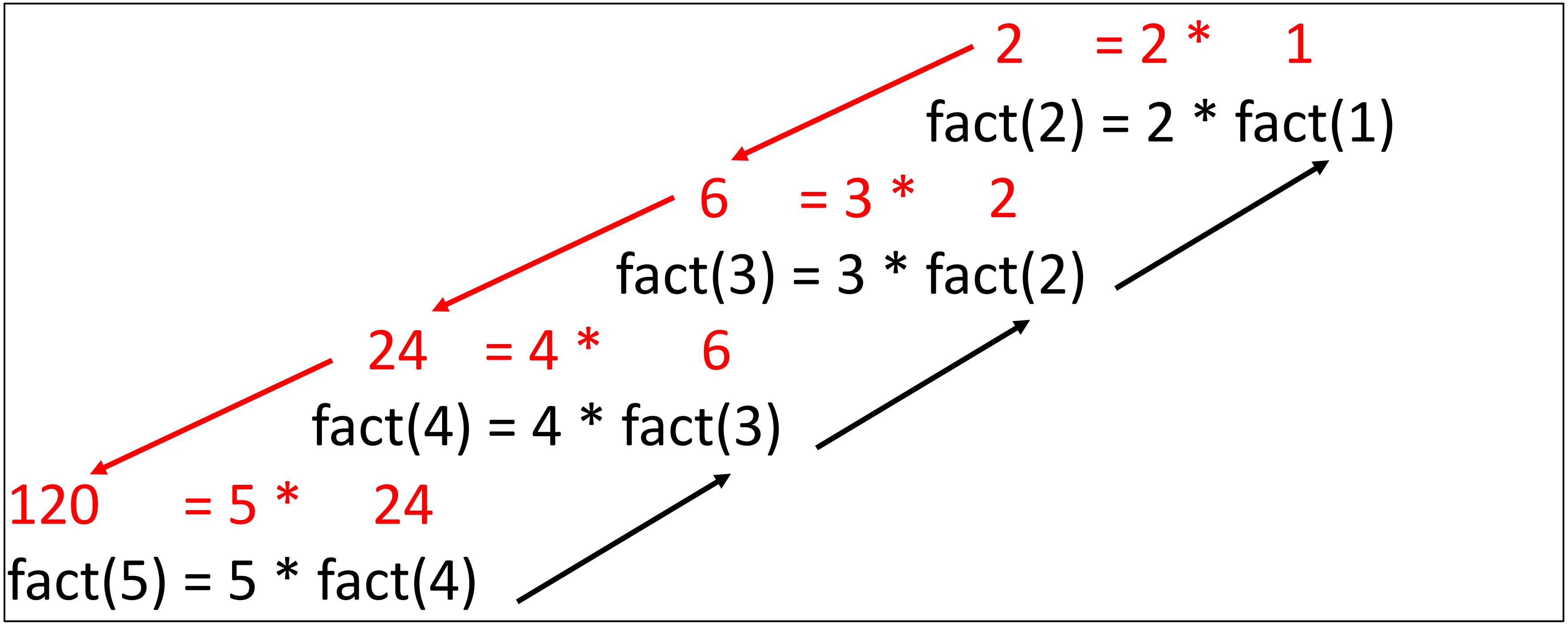
Factorial(5) = 5 x 4 x 3 x 2 x 1

```
int fact(int n)
{
        if (n < = 1) // base case
                return 1;
        else
                return n*fact(n-1);
}
```



Source: https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/

```
int fact(int n)
 {
        if (n < = 1) // base case
                return 1;
        else
                return n*fact(n-1);
 }
```



2 = 2 * 1

fact(2) = 2 * fact(1)

6 = 3 * 2

fact(3) = 3 * fact(2)

24 = 4 * 6

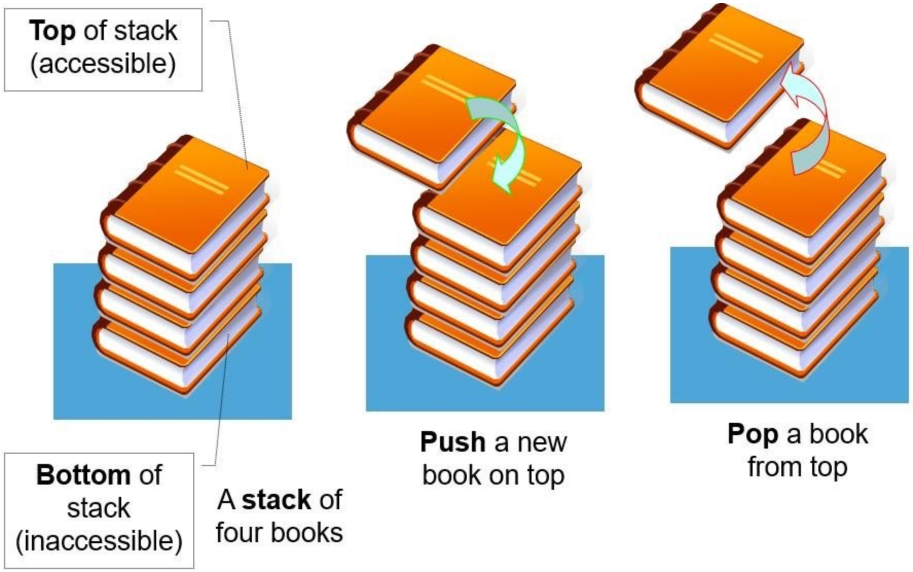
fact(4) = 4 * fact(3)

120 = 5 * 24

fact(5) = 5 * fact(4)

Stack Overflow (2)

- A finite amount of memory in a computer
- Only a certain amount of memory can be used to store stack frames.
- If function calls require more than the amount of memory for stack, then a fatal error occurs.


Top of stack (accessible)
Bottom of stack (inaccessible)
A **stack** of four books
Push a new book on top
Pop a book from top

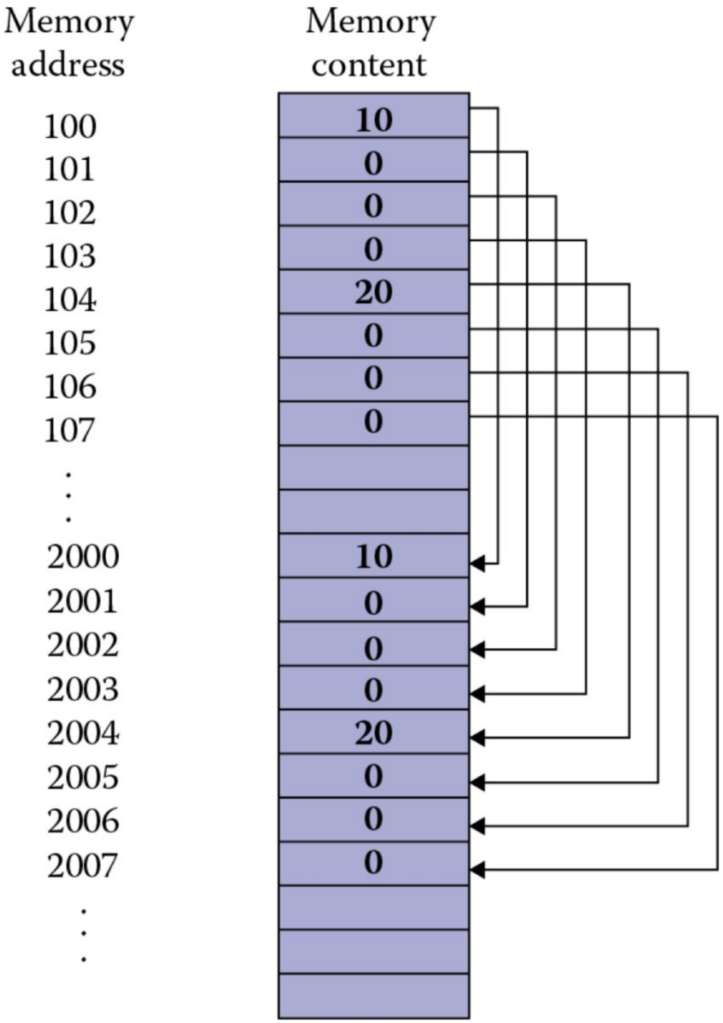
Source: https://visualgo.net/en/list?slide=4.

Overview

- Function
- Function Call Stack
- **Passing data by values and references**

Passing Values: by Value (or Copy)

```
void test(int a, int b);

int main(){
    test(10, 20);
    return 0;
}

void test(int a, int b){
    /* Function body. */
}
```



Memory address	Memory content
100	10
101	0
102	0
103	0
104	20
105	0
106	0
107	0
⋮	
2000	10
2001	0
2002	0
2003	0
2004	20
2005	0
2006	0
2007	0
⋮	

Pass By Value

- Each parameter copies the value given to the function when it is called.

- Changes to the copy do *not* affect an original variable’s value in the caller.

- Pass-by-value should be used whenever the called function does *not* need to modify the value of the caller’s original variable.

Pass By Value (2)

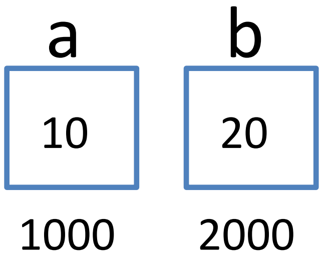
```c
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n", a, b);
   swap(a, b);
   printf("After swapping values in main a = %d, b = %d\n", a, b);
}
void swap (int a, int b)
{
   int temp;
   temp = a;
   a=b;
   b=temp;
   printf("After swapping values in function a = %d, b = %d\n", a, b);
}
```

Output:

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 10, b = 20
```

https://www.javatpoint.com/call-by-value-and-call-by-reference-in-c

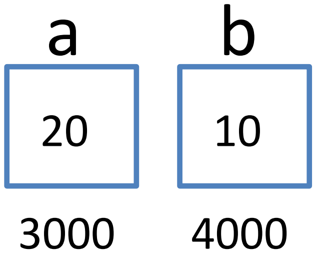
```c
#include <stdio.h>
void swap(int , int);

int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n", a, b);
    swap(a, b);
    printf("After swapping values in main a = %d, b = %d\n", a, b);
}
```



Output:

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 10, b = 20
```

```c
void swap (int a, int b)
{
    int temp;
    temp = a;
    a=b;
    b=temp;
    printf("After swapping values in function a = %d, b = %d\n", a, b);
}
```



https://www.javatpoint.com/call-by-value-and-call-by-reference-in-c

Pass By Reference

- Pass-by-reference should be used only with trusted called functions that need to modify the original variable, or when a huge data-structure needs to be passed around.
- The *memory address* is passed by copying into a variable.
- This allows a function to simulate returning multiple values!!

Pass By Reference (2)

```c
#include <stdio.h>
void swap(int *, int *); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n", a, b);
   swap(&a, &b);
   printf("After swapping values in main a = %d, b = %d\n", a, b);
}
void swap (int *a, int *b)
{
   int temp;
   temp = *a;
   *a=*b;
   *b=temp;
   printf("After swapping values in function a = %d, b = %d\n", *a, *b);
}
```

Output:

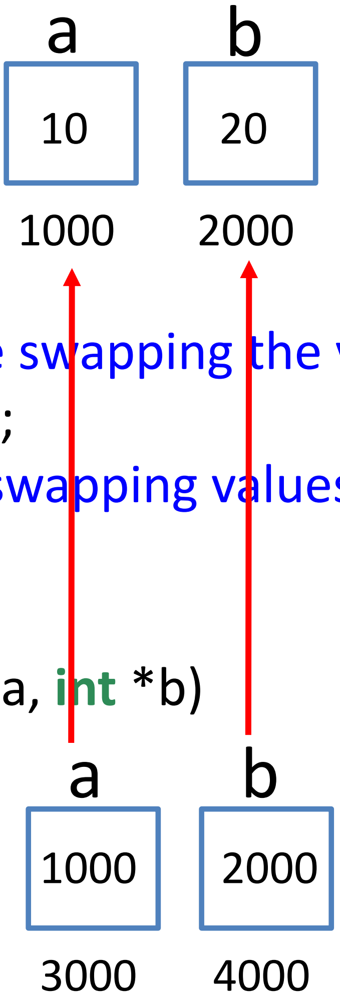
```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 20, b = 10
```

https://www.javatpoint.com/call-by-value-and-call-by-reference-in-c

```
#include <stdio.h>
void swap(int *, int *);

int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swapping values in main a = %d, b = %d\n", a, b);
}

void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a=*b;
    *b=temp;
    printf("After swapping values in function a = %d, b = %d\n", *a, *b);
}
```

Output:

```
Before swapping the values in main a = 10, b = 20
```

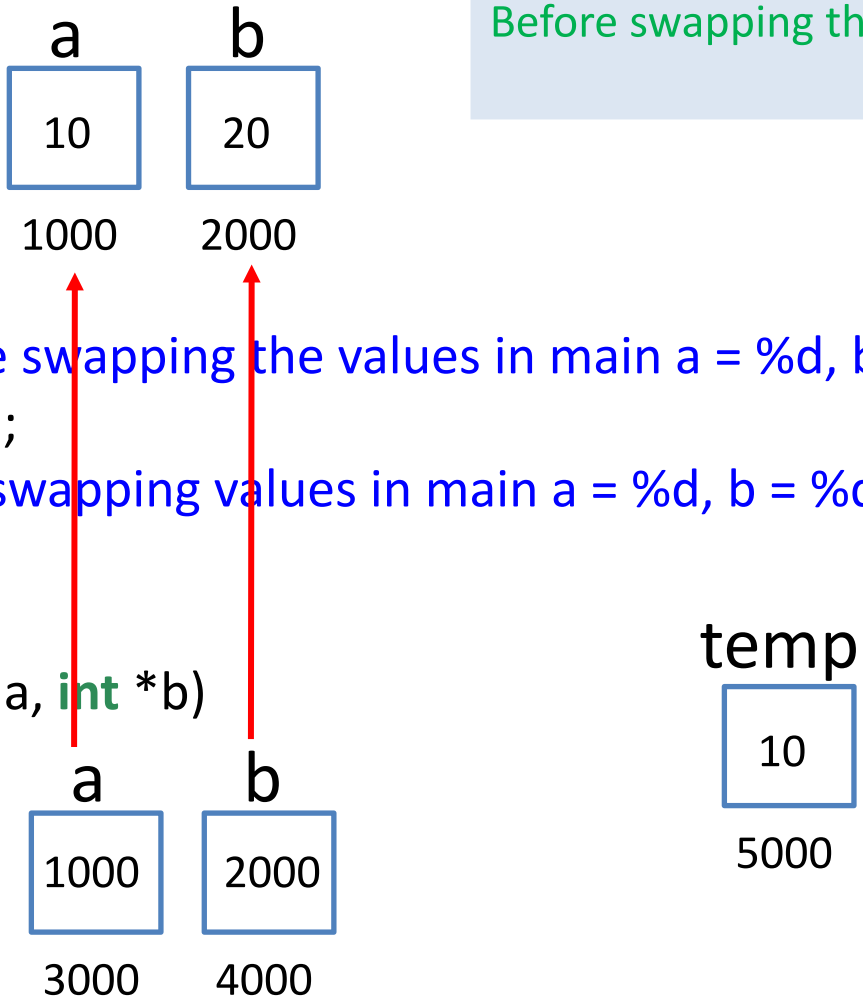


https://www.javatpoint.com/call-by-value-and-call-by-reference-in-c

```
#include <stdio.h>
void swap(int *, int *);

int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swapping values in main a = %d, b = %d\n", a, b);
}

void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a=*b;
    *b=temp;
    printf("After swapping values in function a = %d, b = %d\n", *a, *b);
}
```

Output:

```
Before swapping the values in main a = 10, b = 20
```

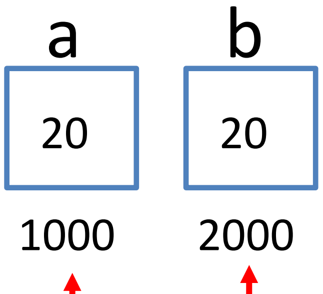


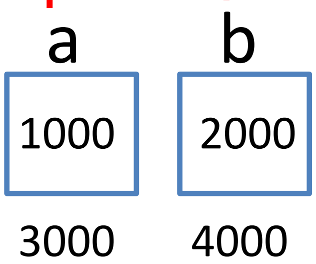
https://www.javatpoint.com/call-by-value-and-call-by-reference-in-c

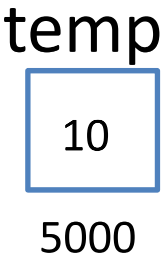
```c
#include <stdio.h>
void swap(int *, int *);

int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swapping values in main a = %d, b = %d\n", a, b);
}

void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a=*b;
    *b=temp;
    printf("After swapping values in function a = %d, b = %d\n", *a, *b);
}
```

Output:

```
Before swapping the values in main a = 10, b = 20
```





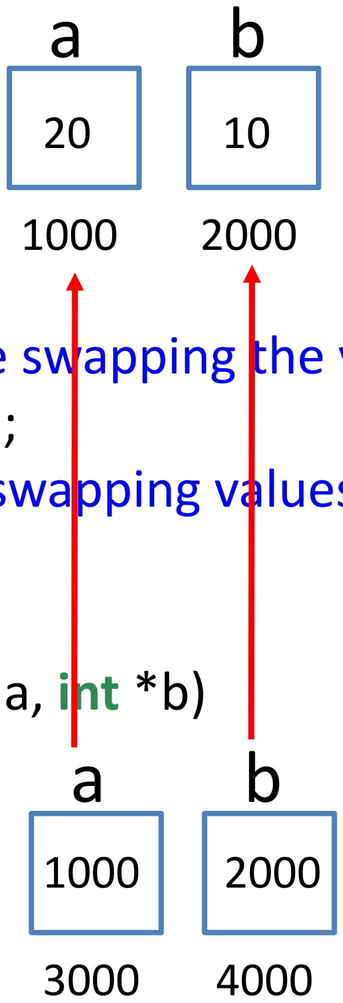


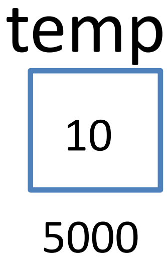
https://www.javatpoint.com/call-by-value-and-call-by-reference-in-c

```
#include <stdio.h>
void swap(int *, int *);

int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swapping values in main a = %d, b = %d\n", a, b);
}

void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a=*b;
    *b=temp;
    printf("After swapping values in function a = %d, b = %d\n", *a, *b);
}
```

Output:

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 20, b = 10
```





https://www.javatpoint.com/call-by-value-and-call-by-reference-in-c

Passing Values: by Reference

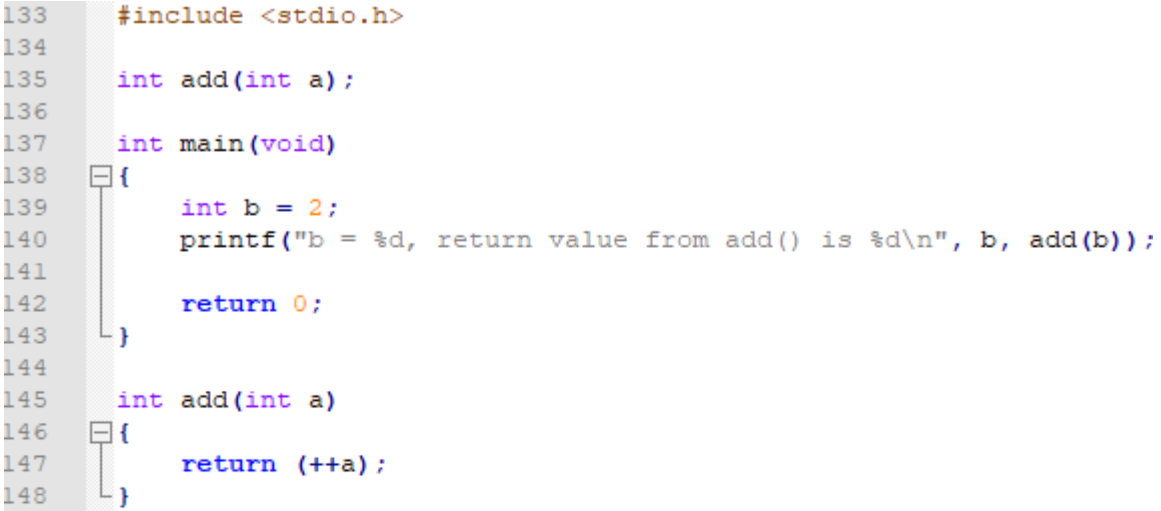
```
void test(int a, int b);

int main(){
    test(10, 20);
    return 0;
}



void test(int a, int b){
    /* Function body. */
}
```

```
void test(int *a, int b);

int main(){
    int *ptr, i = 10;
    ptr = &i;
    test(ptr, 20);
    return 0;
}

void test(int *a, int b){
    /* Function body. */
}
```

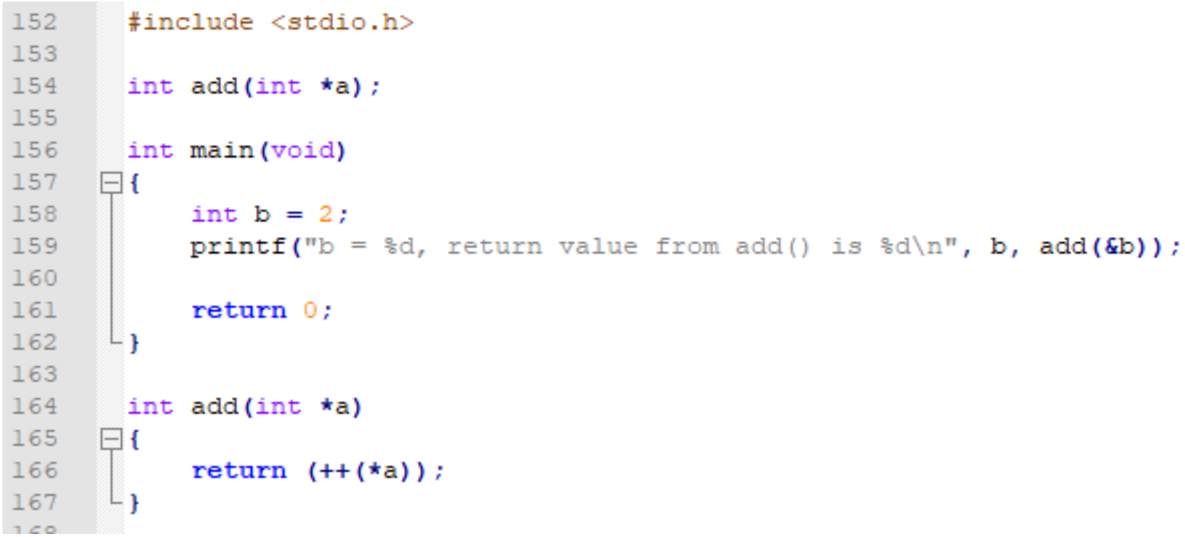
Or test(&i, 20);

Since a function cannot return more than one value, passing arguments by reference is the most flexible way to change the values of the arguments.

Pass By Value vs. Pass By Reference

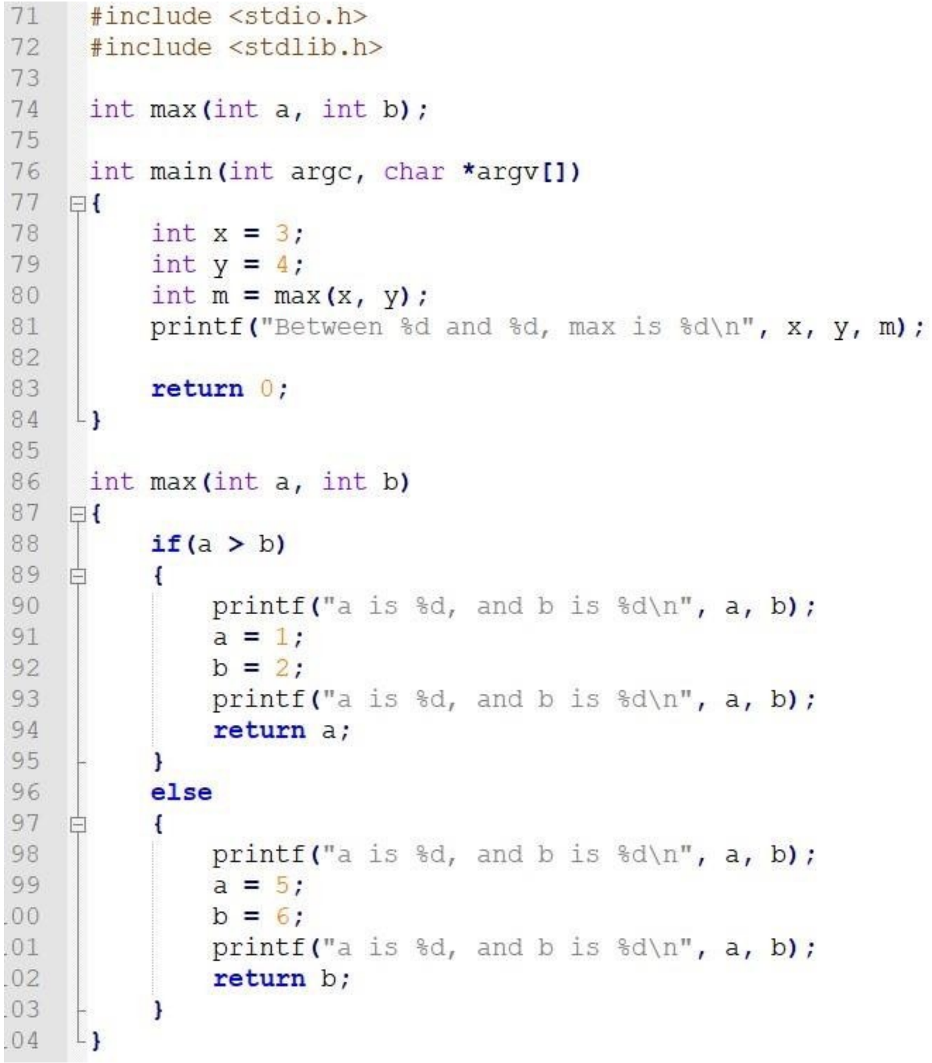


```c
#include <stdio.h>

int add(int a);

int main(void)
{
    int b = 2;
    printf("b = %d, return value from add() is %d\n", b, add(b));

    return 0;
}

int add(int a)
{
    return (++a);
}
```

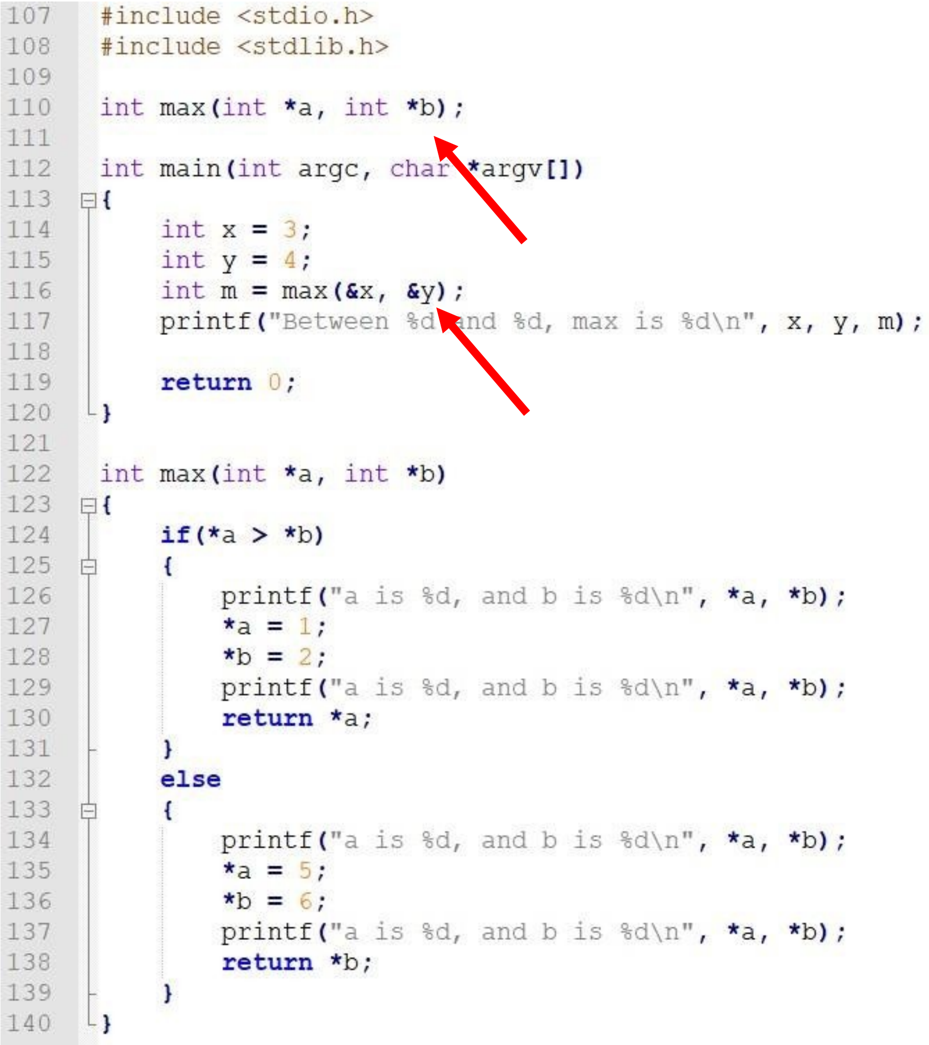


```c
#include <stdio.h>

int add(int *a);

int main(void)
{
    int b = 2;
    printf("b = %d, return value from add() is %d\n", b, add(&b));

    return 0;
}

int add(int *a)
{
    return (++(*a));
}
```

Pass By Value vs. Pass By Reference (2)



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

int max(int a, int b);

int main(int argc, char *argv[])
{
    int x = 3;
    int y = 4;
    int m = max(x, y);
    printf("Between %d and %d, max is %d\n", x, y, m);

    return 0;
}

int max(int a, int b)
{
    if(a > b)
    {
        printf("a is %d, and b is %d\n", a, b);
        a = 1;
        b = 2;
        printf("a is %d, and b is %d\n", a, b);
        return a;
    }
    else
    {
        printf("a is %d, and b is %d\n", a, b);
        a = 5;
        b = 6;
        printf("a is %d, and b is %d\n", a, b);
        return b;
    }
}
```



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

int max(int *a, int *b);

int main(int argc, char *argv[])
{
    int x = 3;
    int y = 4;
    int m = max(&x, &y);
    printf("Between %d and %d, max is %d\n", x, y, m);

    return 0;
}

int max(int *a, int *b)
{
    if(*a > *b)
    {
        printf("a is %d, and b is %d\n", *a, *b);
        *a = 1;
        *b = 2;
        printf("a is %d, and b is %d\n", *a, *b);
        return *a;
    }
    else
    {
        printf("a is %d, and b is %d\n", *a, *b);
        *a = 5;
        *b = 6;
        printf("a is %d, and b is %d\n", *a, *b);
        return *b;
    }
}
```

Pass By Value vs. Pass By Reference (3)

```c
#include<stdio.h>
void change(int num) {
    printf("Before adding value inside function num=%d \n", num);
    num=num+100;
    printf("After adding value inside function num=%d \n", num);
}
int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(x);
    printf("After function call x=%d \n", x);
return 0;
}
```

```c
#include<stdio.h>
void change(int *num) {
    printf("Before adding value inside function num=%d \n",*num);
    (*num) += 100;
    printf("After adding value inside function num=%d \n", *num);
}
int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(&x);
    printf("After function call x=%d \n", x);
return 0;
}
```

University of
Nottingham
UK | CHINA | MALAYSIA

Summary

- Function
- Function Call Stack
- Passing data by values and references