

# Week 5 - Lectures 1, 2

## Functions

Edited by: Heshan Du  
Autumn 2024



# Overview

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



# Prepacked Functions

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

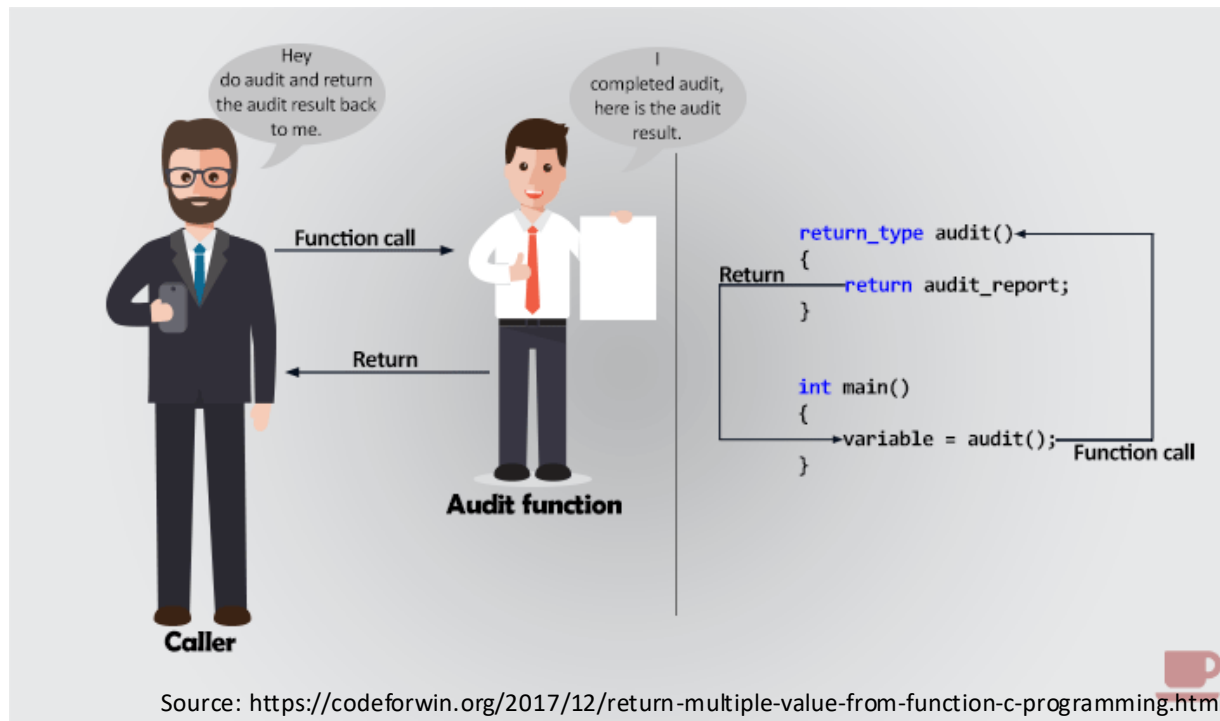
Function	Description	Example
sqrt(x)	square root of $x$	sqrt(900.0) is 30.0 sqrt(9.0) is 3.0
cbrt(x)	cube root of $x$ (C99 and C11 only)	cbrt(27.0) is 3.0 cbrt(-8.0) is -2.0
exp(x)	exponential function $e^x$	exp(1.0) is 2.718282 exp(2.0) is 7.389056
log(x)	natural logarithm of $x$ (base $e$ )	log(2.718282) 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 Deitel (2016). C How to Program with an Introduction to C++ (8<sup>th</sup> Ed.). Pearson.



# Function

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



# 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?

```
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int max(int a, int b);
6
7  int main(int argc, char *argv[])
8  {
9      if(argc == 3)
10     {
11         printf("Max value between %s and %s is: ", argv[1], argv[2]);
12         printf("%d\n", max(atoi(argv[1]), atoi(argv[2])));
13     }
14
15     return 0;
16 }
17
18 int max(int a, int b)
19 {
20     if(a > b)
21     {
22         return a;
23     }
24     else
25     {
26         return b;
27     }
28 }
```





```

2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int max(int a, int b);
6
7  int main(int argc, char *argv[])
8  {
9      if(argc == 3)
10     {
11         printf("Max value between %s and %s is: ", argv[1], argv[2]);
12         printf("%d\n", max(atoi(argv[1]), atoi(argv[2])));
13     }
14
15     return 0;
16 }
17
18 int max(int a, int b)
19 {
20     if(a > b)
21     {
22         return a;
23     }
24     else
25     {
26         return b;
27     }
28 }

```

C:\> maximum

Please enter two numbers

C:\> maximum 3 2

Maximum value between 3 and 2 is: 3



# Function (6)

```
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int max(int a, int b);
6
7  int main(int argc, char *argv[])
8  {
9      if(argc == 3)
10     {
11         printf("Max value between %s and %s is: ", argv[1], argv[2]);
12         printf("%d\n", max(atoi(argv[1]), atoi(argv[2])));
13     }
14
15     return 0;
16 }
17
18 int max(int a, int b)
19 {
20     if(a > b)
21     {
22         return a;
23     }
24     else
25     {
26         return b;
27     }
28 }
```

Declaration

Arguments

Return statement

Definition



```

2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int max(int a, int b);
6
7  int main(int argc, char *argv[])
8  {
9      if(argc == 3)
10     {
11         printf("Max value between %s and %s is: ", argv[1], argv[2]);
12         printf("%d\n", max(atoi(argv[1]), atoi(argv[2])));
13     }
14
15     return 0;
16 }
17
18 int max(int a, int b)
19 {
20     if(a > b)
21     {
22         return a;
23     }
24     else
25     {
26         return b;
27     }
28 }

```

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;
}
```

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

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

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.

```
36 #include <stdio.h>
37
38 int avg(float a, float b);
39
40 int main(int argc, char *argv[])
41 {
42     printf("Outside function: %f\n\n", avg(4.9, 2.0));
43     printf("Outside function: %d\n", avg(4.9, 2.0));
44
45     return 0;
46 }
47
48 int avg(float a, float b)
49 {
50     printf("Inside function: %f\n", (a/b));
51     return (a/b);
52     // NOTE: the difference in output from both inside and outside of the function
53 }
```

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?

```
#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

```
#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;
}
```

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

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

    return sum;
}
```



# Example 3

```
1 // Fig. 5.3: fig05_03.c
2 // Creating and using a programmer-defined function.
3 #include <stdio.h>
4
5 int square( int y ); // function prototype
6
7 // function main begins program execution
8 int main( void )
9 {
10     int x; // counter
11
12     // loop 10 times and calculate and output square of x each time
13     for ( x = 1; x <= 10; ++x ) {
14         printf( "%d ", square( x ) ); // function call
15     } // end for
16
17     puts( "" );
18 } // end main
19
20 // square function definition returns the square of its parameter
21 int square( int y ) // y is a copy of the argument to the function
22 {
23     return y * y; // returns the square of y as an int
24 } // 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 (1)

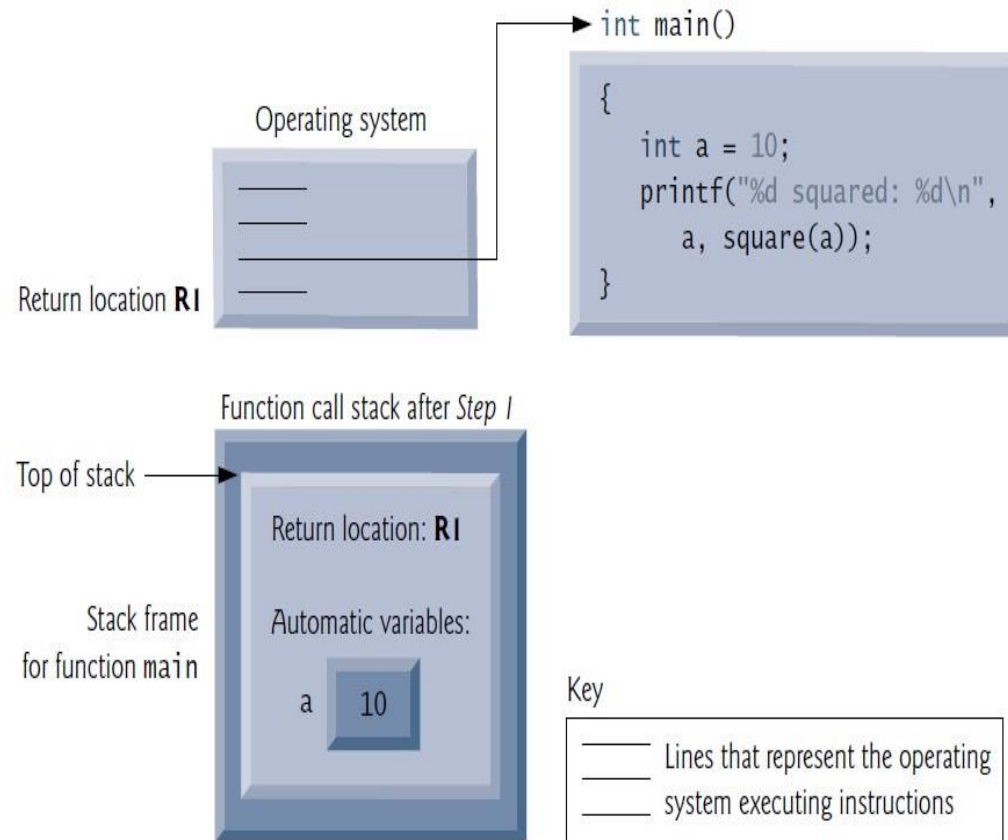
- Stacks are known as *last-in, first-out* (LIFO) data structures—the last item pushed (inserted) on the stack is the first item popped (removed) from the stack.
- The function call stack *data structure* supports the function call/return mechanism.
- The function call stack also supports the creation,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



# Function Call Stack (2)

```
1 // Fig. 5.6: fig05_06.c
2 // Demonstrating the function call stack
3 // and stack frames using a function square.
4 #include <stdio.h>
5
6 int square(int); // prototype for function square
7
8 int main()
9 {
10     int a = 10; // value to square (local automatic variable in main)
11
12     printf("%d squared: %d\n", a, square(a)); // display a squared
13 }
14
15 // returns the square of an integer
16 int square(int x) // x is a local variable
17 {
18     return x * x; // calculate square and return result
19 }
```

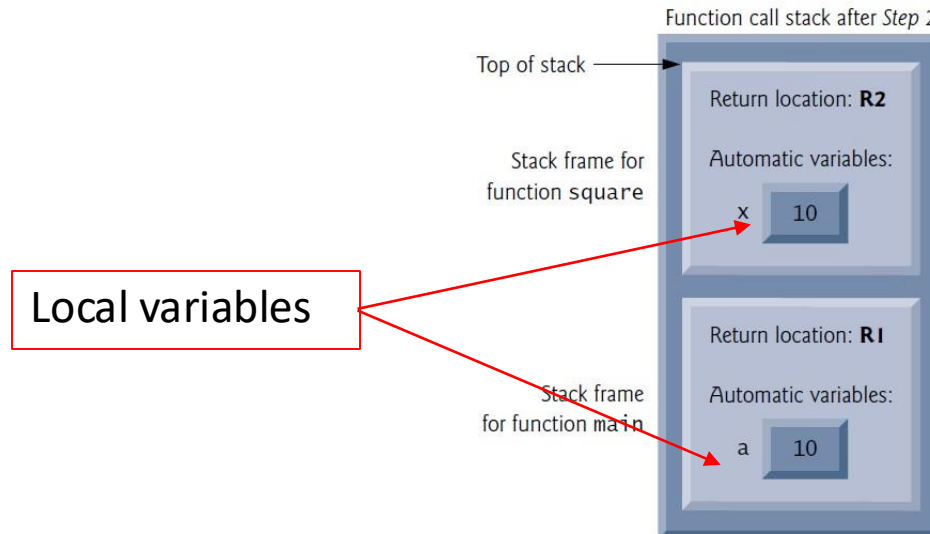
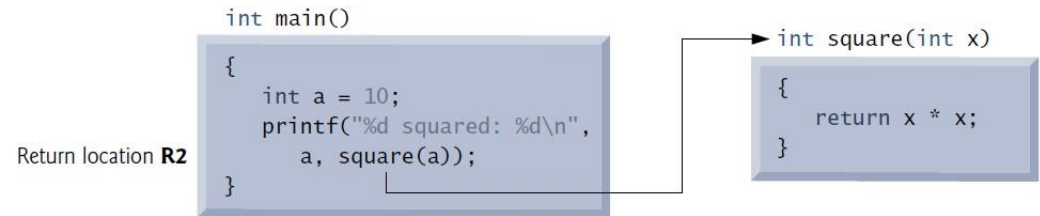
Step 1: Operating system invokes main to execute application



# Function Call Stack (3)

```
1 // Fig. 5.6: fig05_06.c
2 // Demonstrating the function call stack
3 // and stack frames using a function square.
4 #include <stdio.h>
5
6 int square(int); // prototype for function square
7
8 int main()
9 {
10     int a = 10; // value to square (local automatic variable in main)
11
12     printf("%d squared: %d\n", a, square(a)); // display a squared
13 }
14
15 // returns the square of an integer
16 int square(int x) // x is a local variable
17 {
18     return x * x; // calculate square and return result
19 }
```

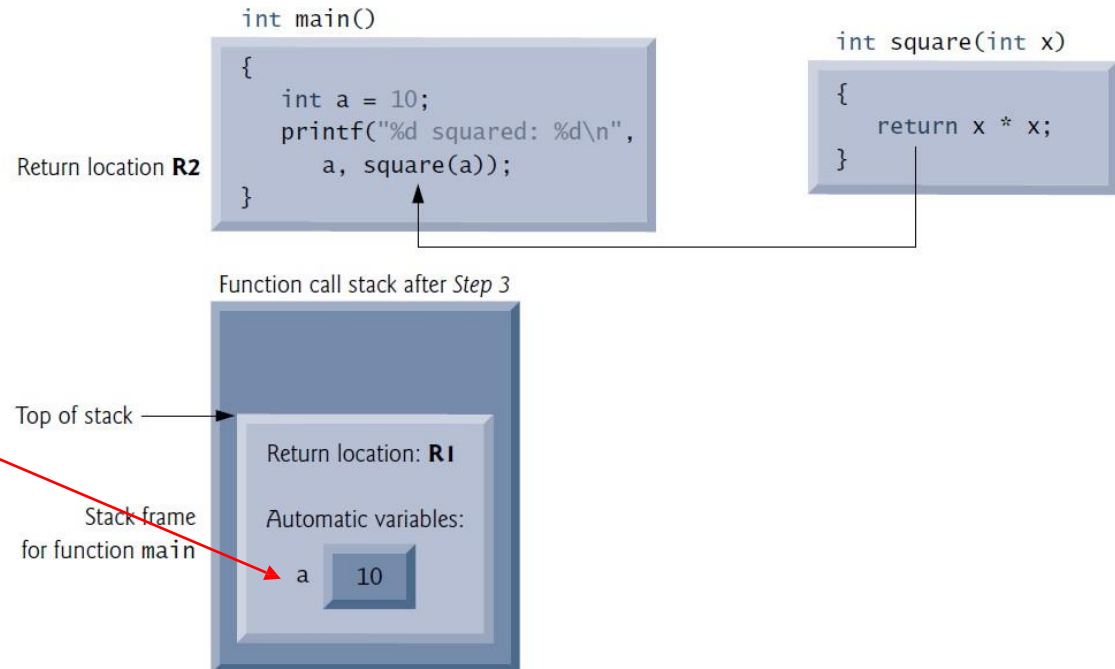
Step 2: main invokes function square to perform calculation



# Function Call Stack (4)

```
1 // Fig. 5.6: fig05_06.c
2 // Demonstrating the function call stack
3 // and stack frames using a function square.
4 #include <stdio.h>
5
6 int square(int); // prototype for function square
7
8 int main()
9 {
10     int a = 10; // value to square (local automatic variable in main)
11
12     printf("%d squared: %d\n", a, square(a)); // display a squared
13 }
14
15 // returns the square of an integer
16 int square(int x) // x is a local variable
17 {
18     return x * x; // calculate square and return result
19 }
```

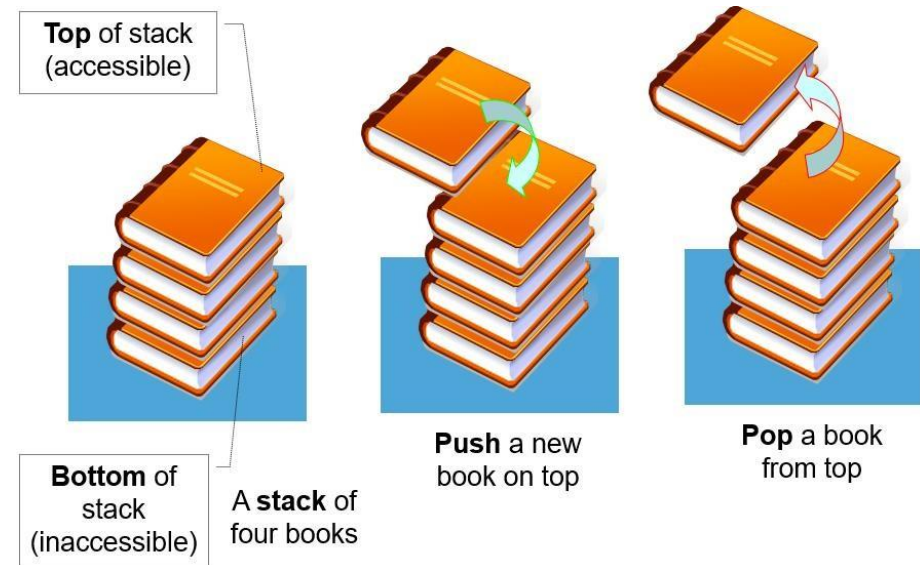
Step 3: square returns its result to main





# Stack Overflow

- 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.



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



# Overview

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

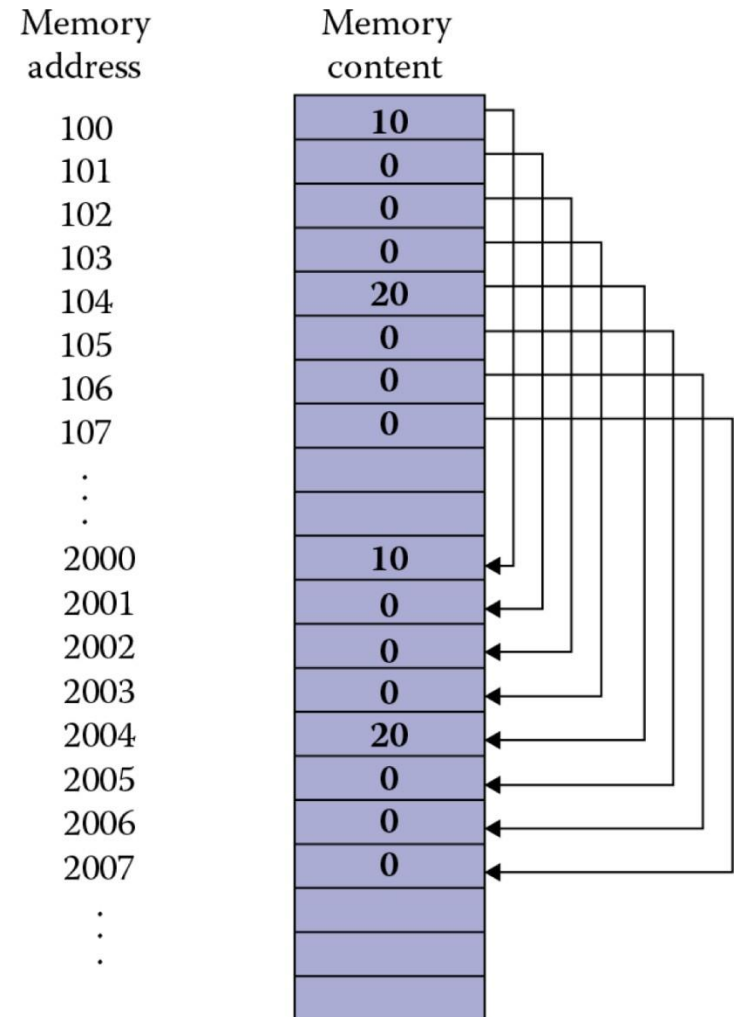


# Passing Values: by Value (or Copy)

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

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

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



# 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)

```
#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>



```
#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);  
}
```

a	b
10	20
1000	2000

### 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

```
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);  
}
```

a	b
20	10
3000	4000

# 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.

# Pass By Reference (2)

```
#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
```



```
#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;
```

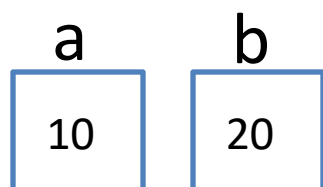
```
    *b=temp;
```

```
    printf("After swapping values in function a = %d, b = %d\n", *a, *b);
```

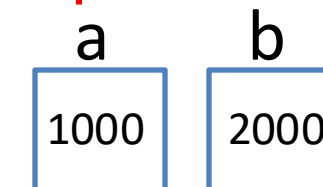
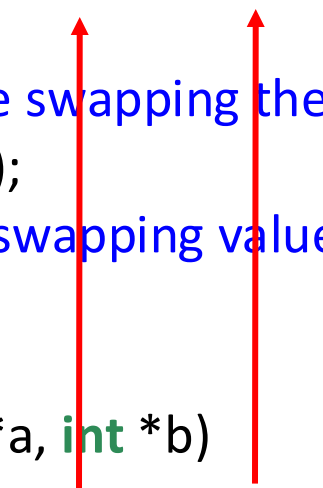
```
}
```

Output:

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



1000      2000



3000      4000

```
#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;
```

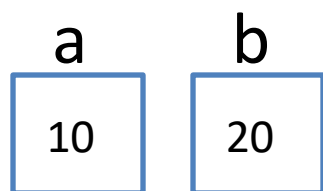
```
    *b=temp;
```

```
    printf("After swapping values in function a = %d, b = %d\n", *a, *b);
```

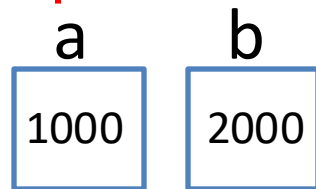
```
}
```

Output:

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

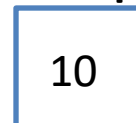


1000      2000



3000      4000

temp



5000

```
#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;
```

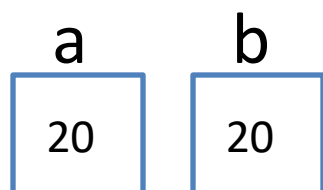
```
    *b=temp;
```

```
    printf("After swapping values in function a = %d, b = %d\n", *a, *b);
```

```
}
```

Output:

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

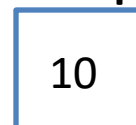


1000

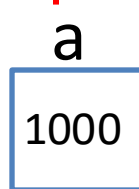
2000



temp



5000



3000

4000

```
#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);
```

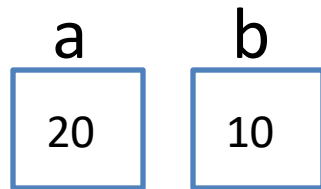
```
}
```

Output:

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

After swapping values in function a = 20, b = 10

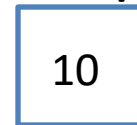
After swapping values in main a = 20, b = 10



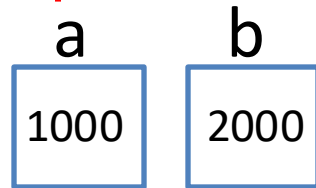
1000      2000



temp



5000



3000      4000

# 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

```
133 #include <stdio.h>
134
135 int add(int a);
136
137 int main(void)
138 {
139     int b = 2;
140     printf("b = %d, return value from add() is %d\n", b, add(b));
141
142     return 0;
143 }
144
145 int add(int a)
146 {
147     return (++a);
148 }
```

```
152 #include <stdio.h>
153
154 int add(int *a);
155
156 int main(void)
157 {
158     int b = 2;
159     printf("b = %d, return value from add() is %d\n", b, add(&b));
160
161     return 0;
162 }
163
164 int add(int *a)
165 {
166     return (++(*a));
167 }
```



# Pass By Value vs. Pass By Reference (2)

```
71 #include <stdio.h>
72 #include <stdlib.h>
73
74 int max(int a, int b);
75
76 int main(int argc, char *argv[])
77 {
78     int x = 3;
79     int y = 4;
80     int m = max(x, y);
81     printf("Between %d and %d, max is %d\n", x, y, m);
82
83     return 0;
84 }
85
86 int max(int a, int b)
87 {
88     if(a > b)
89     {
90         printf("a is %d, and b is %d\n", a, b);
91         a = 1;
92         b = 2;
93         printf("a is %d, and b is %d\n", a, b);
94         return a;
95     }
96     else
97     {
98         printf("a is %d, and b is %d\n", a, b);
99         a = 5;
100        b = 6;
101        printf("a is %d, and b is %d\n", a, b);
102        return b;
103    }
104 }
```

```
107 #include <stdio.h>
108 #include <stdlib.h>
109
110 int max(int *a, int *b);
111
112 int main(int argc, char *argv[])
113 {
114     int x = 3;
115     int y = 4;
116     int m = max(&x, &y);
117     printf("Between %d and %d, max is %d\n", x, y, m);
118
119     return 0;
120 }
121
122 int max(int *a, int *b)
123 {
124     if(*a > *b)
125     {
126         printf("a is %d, and b is %d\n", *a, *b);
127         *a = 1;
128         *b = 2;
129         printf("a is %d, and b is %d\n", *a, *b);
130         return *a;
131     }
132     else
133     {
134         printf("a is %d, and b is %d\n", *a, *b);
135         *a = 5;
136         *b = 6;
137         printf("a is %d, and b is %d\n", *a, *b);
138         return *b;
139     }
140 }
```



# Pass By Value vs. Pass By Reference (3)

```
#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;
}
```

```
#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;
}
```





# Summary

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

