Functions

Advanced C

Functions – What?

An activity that is natural to or the purpose of a person or thing.

"bridges perform the function of providing access across water"

A relation or expression involving one or more variables.

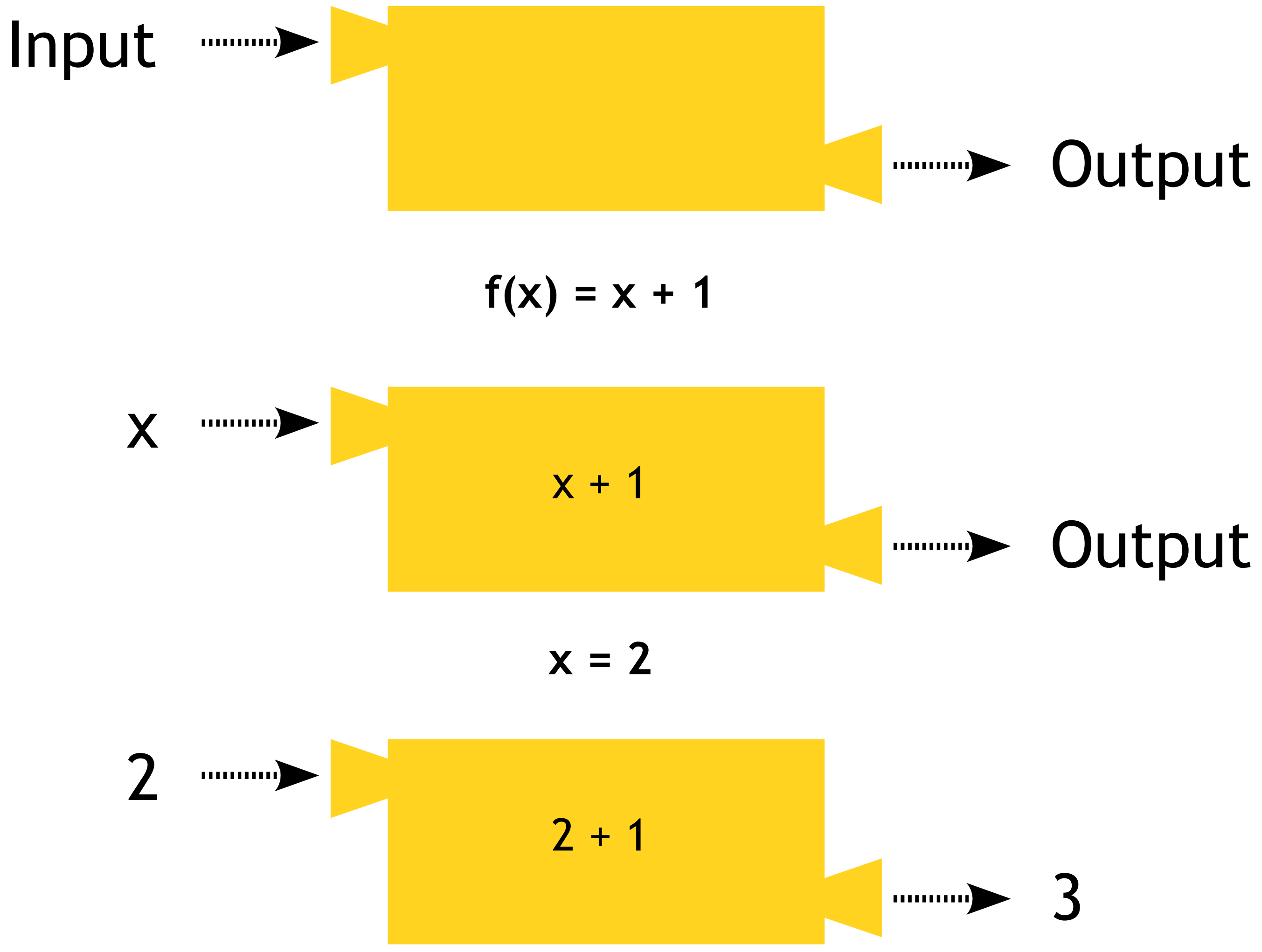
"the function (bx + c)"

Source: Google

- In programing languages it can be something which performs a specific service
- Generally a function has 3 properties
  - Takes Input
  - Perform Operation
  - Generate Output

Advanced C

Functions – What?



Advanced C

Functions – How to write

Syntax

```
return_data_type function_name(arg_1, arg_2, ..., arg_n)
{
    /* Function Body */
}
```

List of function parameters

Example

Return data type as **int**

First parameter with **int** type

```
int foo(int arg_1, int arg_2)
{

}
```

Second parameter with **int** type

Advanced C

Functions – How to write

y = x + 1

Example

```
int foo(int x)
{
    int ret;

    ret = x + 1;

    return ret;
}
```

Return from function

Advanced C

Functions – How to call

Example

```c
#include <stdio.h>

int main()
{
    int x, y;

    x = 2;

    y = foo(x);
    printf(“y is %d\n”, y);

    return 0;
}
```

```c
int foo(int x)
{
    int ret = 0;

    ret = x + 1;

    return ret;
}
```

The function call

Advanced C

Functions – Why?

- Re usability
  - Functions can be stored in library & re-used
  - When some specific code is to be used more than once, at different places, functions avoids repetition of the code.
- Divide & Conquer
  - A big & difficult problem can be divided into smaller sub-problems and solved using divide & conquer technique
- Modularity can be achieved.
- Code can be easily understandable & modifiable.
- Functions are easy to debug & test.
- One can suppress, how the task is done inside the function, which is called Abstraction

Advanced C

Functions – A complete look

Example

```
#include <stdio.h>

int main()
{
    int num1 = 10, num2 = 20;
    int sum = 0;

    sum = add_numbers(num1, num2);
    printf("Sum is %d\n", sum);

    return 0;
}
int add_numbers(int num1, int num2)
{
    int sum = 0;

    sum = num1 + num2;

    return sum;
}
```

The main function

The function call

Actual arguments

Return type

Formal arguments

operation

Return result from function and exit

Advanced C

Functions – Ignoring return value

Example

```
#include <stdio.h>

int main()
{
    int num1 = 10, num2 = 20;
    int sum = 0;

    add_numbers(num1, num2);
    printf("Sum is %d\n", sum);

    return 0;
}
```

```
int add_numbers(int num1, int num2)
{
    int sum = 0;

    sum = num1 + num2;

    return sum;
}
```

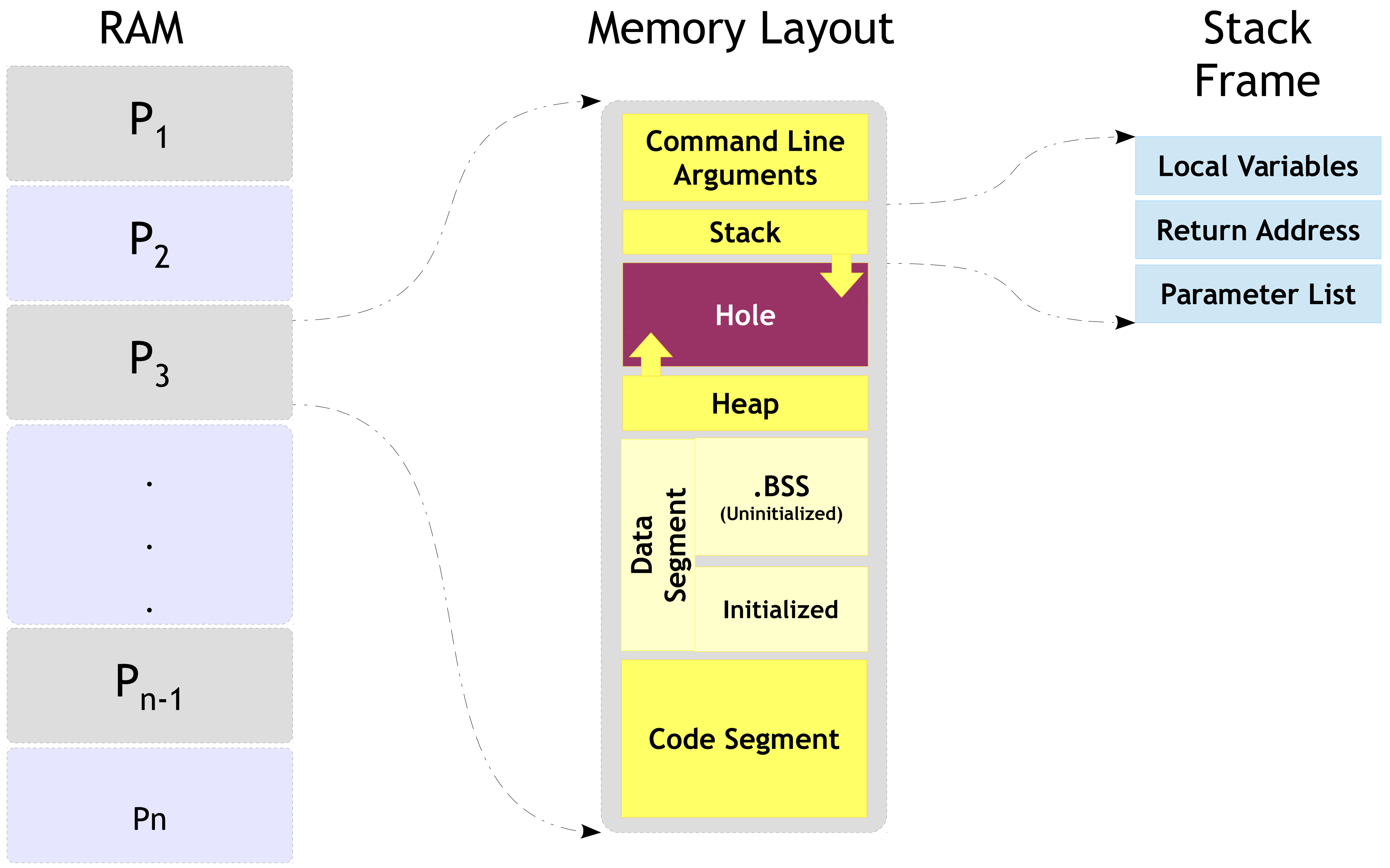
Ignored the return from function
In C, it is up to the programmer to capture or ignore the return value

Advanced C

Functions – DIY

- Write a function to calculate square a number
  - y = x * x
- Write a function to convert temperature given in degree Fahrenheit to degree Celsius
  - C = 5/9 * (F – 32)
- Write a program to check if a given number is even or odd. Function should return TRUE or FALSE

Advanced C
Function and the Stack



Advanced C

Functions – Parameter Passing Types

Pass by Value	Pass by reference
• This method copies the actual value of an argument into the formal parameter of the function.	• This method copies the address of an argument into the formal parameter.
• In this case, changes made to the parameter inside the function have no effect on the actual argument.	• Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

Advanced C

Functions – Pass by Value

Example

```c
#include <stdio.h>

int add_numbers(int num1, int num2);

int main()
{
    int num1 = 10, num2 = 20, sum;

    sum = add_numbers(num1, num2);
    printf("Sum is %d\n", sum);

    return 0;
}
```

```c
int add_numbers(int num1, int num2)
{
    int sum = 0;

    sum = num1 + num2;

    return sum;
}
```

Advanced C

Functions – Pass by Value

Example

```
#include <stdio.h>

void modify(int num1)
{
    num1 = num1 + 1;
}

int main()
{
    int num1 = 10;

    printf(“Before Modification\n”);
    printf(“num1 is %d\n”, num1);

    modify(num1);

    printf(“After Modification\n”);
    printf(“num1 is %d\n”, num1);

    return 0;
}
```

Advanced C

Functions – Pass by Value

Are you sure you understood the previous problem?

Are you sure you are ready to proceed further?

Do you know the prerequisite to proceed further?

If no let's get it cleared

Advanced C

Functions – Pass by Reference

Example

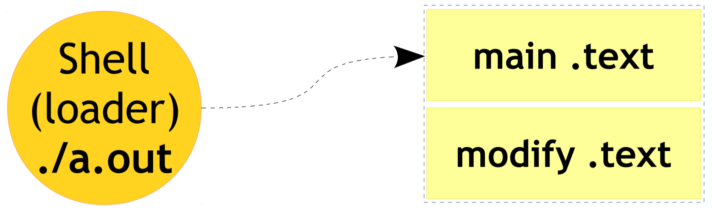
```c
#include <stdio.h>

void modify(int *iptr)
{
    *iptr = *iptr + 1;
}

int main()
{
    int num = 10;

    printf(“Before Modification\n”);
    printf(“num1 is %d\n”, num);

    modify(&num);

    printf(“After Modification\n”);
    printf(“num1 is %d\n”, num);

    return 0;
}
```

Advanced C

Functions – Pass by Reference

Example

```c
#include <stdio.h>

void modify(int *iptr)
{
    *iptr = *iptr + 1;
}

int main()
{
    int num = 10;

    printf(“Before Modification\n”);
    printf(“num1 is %d\n”, num);

    modify(&num);

    printf(“After Modification\n”);
    printf(“num1 is %d\n”, num);

    return 0;
}
```

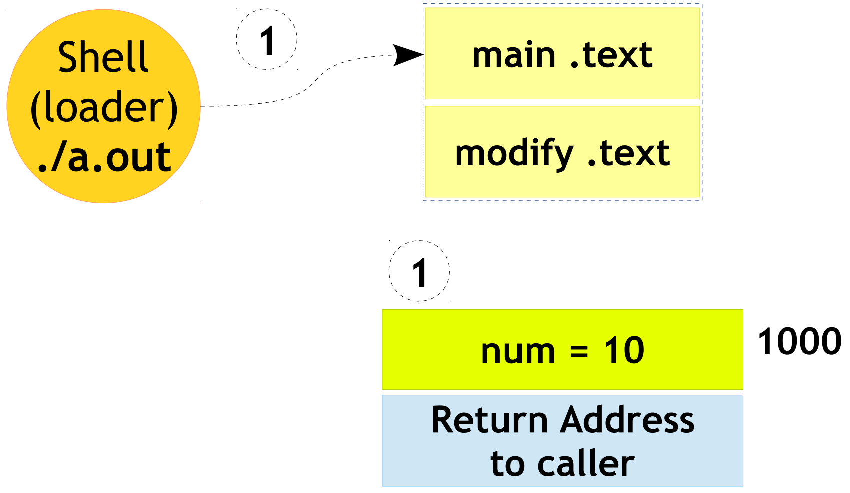


Advanced C

Functions – Pass by Reference

Example

```
#include <stdio.h>

void modify(int *iptr)
{
    *iptr = *iptr + 1;
}

int main()
{
    int num = 10;

    printf(“Before Modification\n”);
    printf(“num1 is %d\n”, num);

    modify(&num);

    printf(“After Modification\n”);
    printf(“num1 is %d\n”, num);

    return 0;
}
```

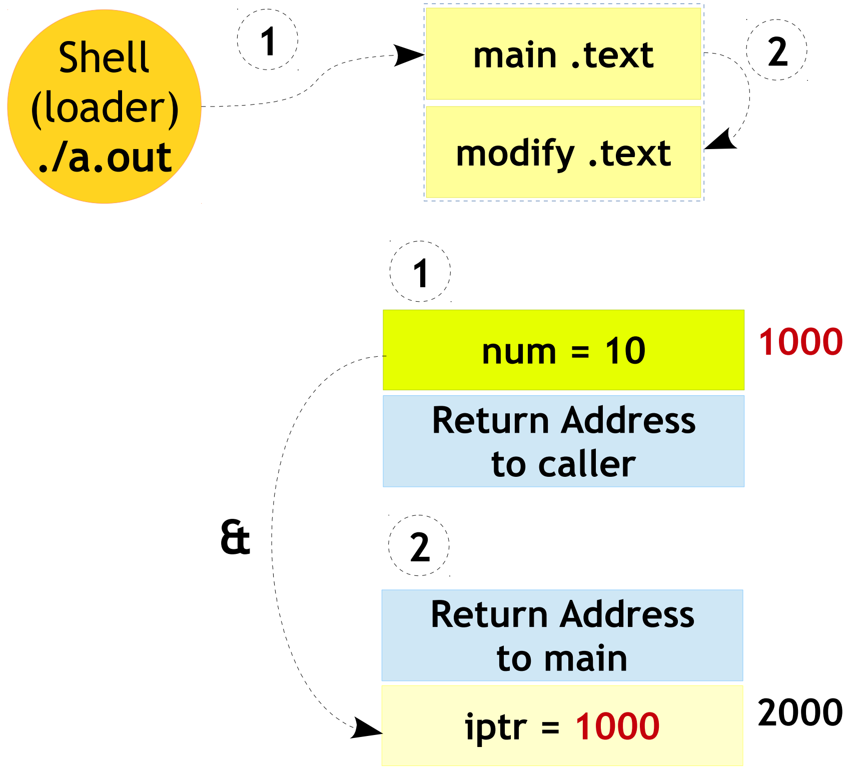


Advanced C

Functions – Pass by Reference

Example

```c
#include <stdio.h>

void modify(int *iptr)
{
    *iptr = *iptr + 1;
}

int main()
{
    int num = 10;

    printf("Before Modification\n");
    printf("num1 is %d\n", num);

    modify(&num);

    printf("After Modification\n");
    printf("num1 is %d\n", num);

    return 0;
}
```

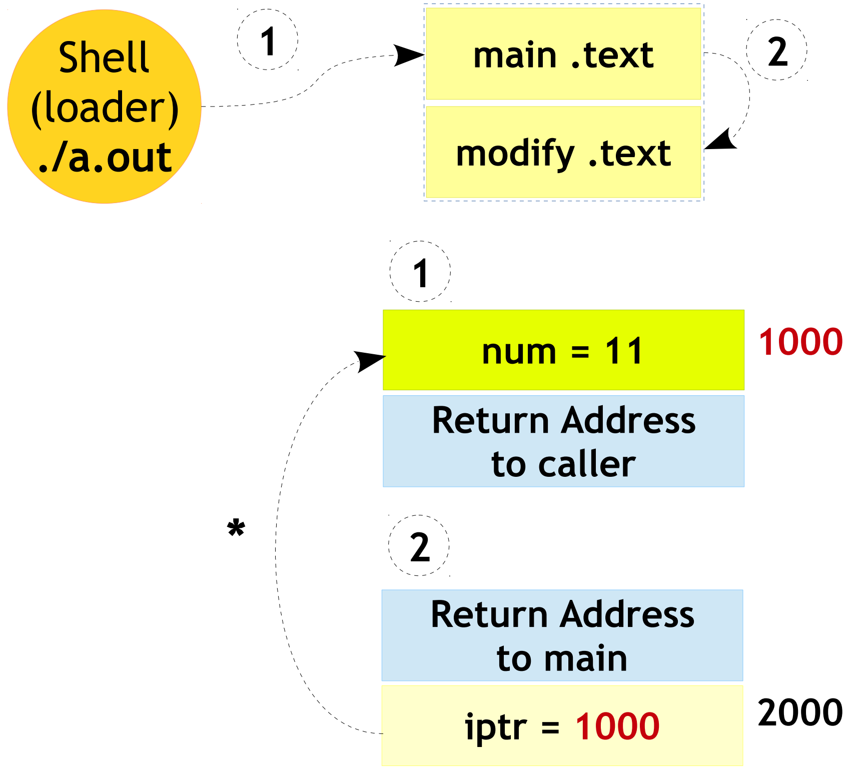


Advanced C

Functions – Pass by Reference

Example

```
#include <stdio.h>

void modify(int *iptr)
{
    *iptr = *iptr + 1;
}

int main()
{
    int num = 10;

    printf("Before Modification\n");
    printf("num1 is %d\n", num);

    modify(&num);

    printf("After Modification\n");
    printf("num1 is %d\n", num);

    return 0;
}
```



Advanced C

Functions – Pass by Reference - Advantages

- Return more than one value from a function
- Copy of the argument is not made, making it fast, even when used with large variables like arrays etc.
- Saving stack space if argument variables are larger (example – user defined data types)

Advanced C

Functions – DIY (pass-by-reference)

- Write a program to find the square and cube of a number
- Write a program to swap two numbers
- Write a program to find the sum and product of 2 numbers
- Write a program to find the square of a number

Advanced C

Functions – Prototype

- Need of function prototype
- Implicit int rule

Advanced C

Functions – Passing Array

- As mentioned in previous slide passing an array to function can be faster
- But before you proceed further it is expected you are familiar with some pointer rules
- If you are OK with your concepts proceed further, else please know the rules first

Advanced C

Functions – Passing Array

Example

```
#include <stdio.h>

void print_array(int array[]);

int main()
{
    int array[5] = {10, 20, 30, 40, 50};

    print_array(array);

    return 0;
}
```

```
void print_array(int array[])
{
    int iter;

    for (iter = 0; iter < 5; iter++)
    {
        printf("Index %d has Element %d\n", iter, array[iter]);
    }
}
```

Advanced C

Functions – Passing Array

Example

```
#include <stdio.h>

void print_array(int *array);

int main()
{
    int array[5] = {10, 20, 30, 40, 50};

    print_array(array);

    return 0;
}
```

```
void print_array(int *array)
{
    int iter;

    for (iter = 0; iter < 5; iter++)
    {
        printf("Index %d has Element %d\n", iter, *array);
        array++;
    }
}
```

Advanced C

Functions – Passing Array

Example

```
#include <stdio.h>

void print_array(int *array, int size);

int main()
{
    int array[5] = {10, 20, 30, 40, 50};

    print_array(array, 5);

    return 0;
}
```

```
void print_array(int *array, int size)
{
    int iter;

    for (iter = 0; iter < size; iter++)
    {
        printf("Index %d has Element %d\n", iter, *array++);
    }
}
```

ΣMERTXE

Advanced C

Functions – Returning Array

Example

```
#include <stdio.h>

int *modify_array(int *array, int size);
void print_array(int array[], int size);

int main()
{
     int array[5] = {10, 20, 30, 40, 50};
     int *new_array_val;

     new_array_val = modify_array(array, 5);
     print_array(new_array_val, 5);

     return 0;
}
```

```
int *modify_array(int *array, int size)
{
     int  iter;

     for (iter = 0; iter < size; iter++)
     {
          *(array + iter) += 10;
     }

     return array;
}
```

```
void print_array(int array[], int size)
{
     int  iter;

     for (iter = 0; iter < size; iter++)
     {
          printf("Index %d has Element %d\n", iter, array[iter]);
     }
}
```

Advanced C

Functions – Returning Array

Example

```c
#include <stdio.h>

int *return_array(void);
void print_array(int *array, int size);

int main()
{
	int *array_val;

	array_val = return_array();
	print_array(array_val, 5);

	return 0;
}
```

```c
int *return_array(void)
{
	static int array[5] = {10, 20, 30, 40, 50};

	return array;
}
```

```c
void print_array(int *array, int size)
{
	int  iter;

	for (iter = 0; iter < size; iter++)
	{
		printf("Index %d has Element %d\n", iter, array[iter]);
	}
}
```

Advanced C

Functions – DIY

- Write a program to find the average of 5 array elements using function
- Write a program to square each element of array which has 5 elements

Advanced C

Functions – Return Type

- Local return
- Void return

Recursive Function

Advanced C

Functions

Advanced C

Functions – Recursive

- Recursion is the process of repeating items in a self-similar way
- In programming a function calling itself is called as recursive function
- Two steps

  Step 1: Identification of base case

  Step 2: Writing a recursive case

Advanced C

Functions - Recursive - Example

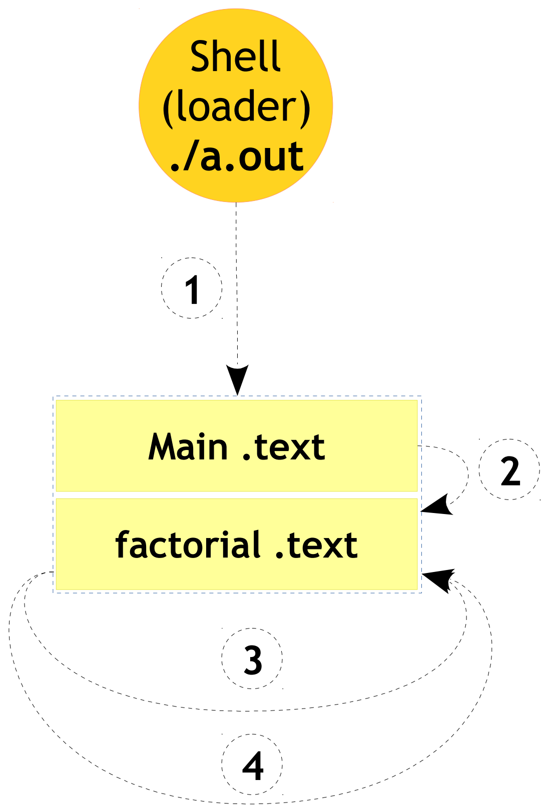
Example

```c
#include <stdio.h>

/* Factorial of 3 numbers */

int factorial(int number)
{
    if (number <= 1) /* Base Case */
    {
        return 1;
    }
    else /* Recursive Case */
    {
        return number * factorial(number - 1);
    }
}

int main()
{
    int ret;

    ret = factorial(3);
    printf("Factorial of 3 is %d\n", ret);

    return 0;
}
```

n	!n
0	1
1	1
2	2
3	6
4	24

Embedded C

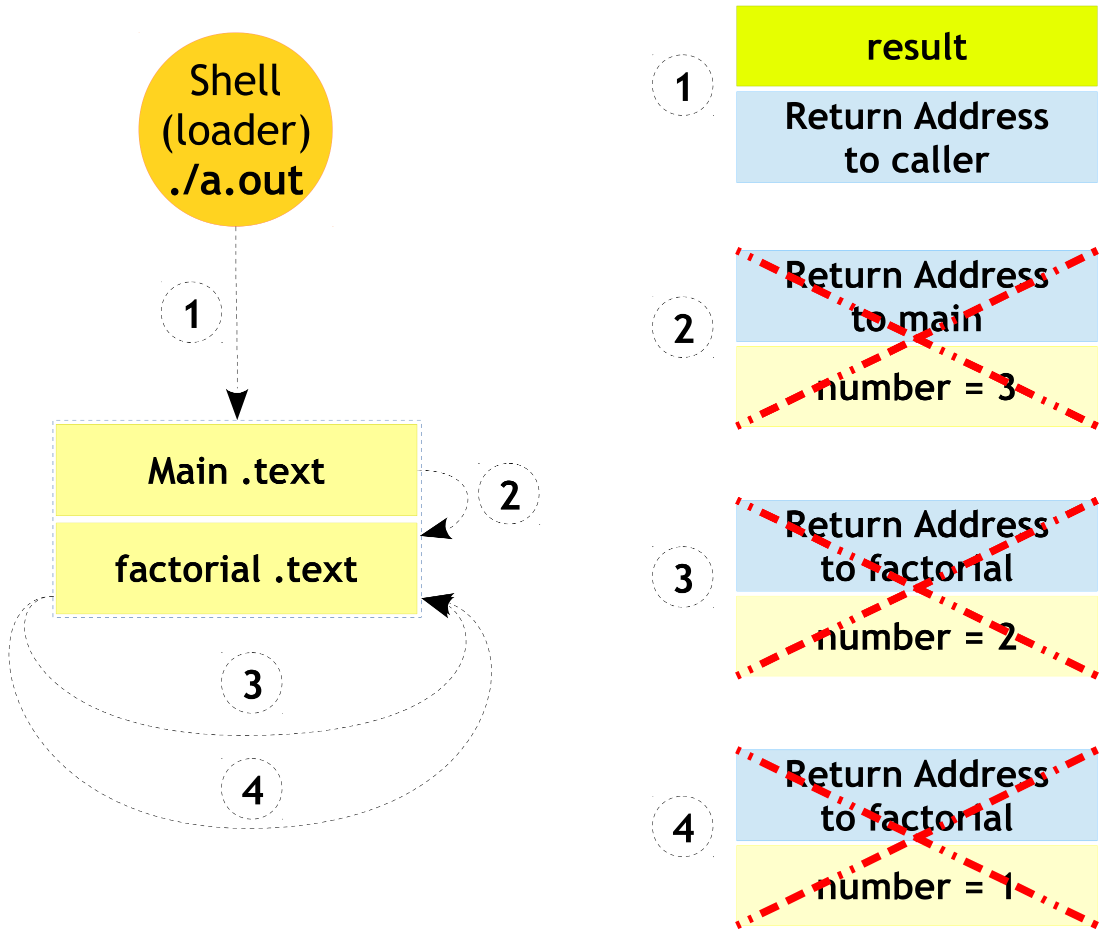
Functions – Recursive – Example Flow



	Stack Frames	Value with calls
1	result Return Address to caller	factorial(3)
2	Return Address to main number = 3	number != 1 number * factorial(number -1) 3 * factorial(3 -1)
3	Return Address to factorial number = 2	number != 1 number * factorial(number -1) 2 * factorial(2 -1)
4	Return Address to factorial number = 1	number == 1

Embedded C

Functions – Recursive – Example Flow



Results with return

Gets 6 a value

Returns 3 * 2 to the caller

Returns 2 * 1 to the caller

returns 1 to the caller

Advanced C

Functions – DIY

- Write a program to find the sum of sequence of N from starting from 1

Standard I/O Functions