# Short Course on Programming in C/C++

Organized by Onur Pekcan

Contributor Selim Temizer  Instructor Hasan Yılmaz

# Week 1 - Lecture

**Today**

We will cover;

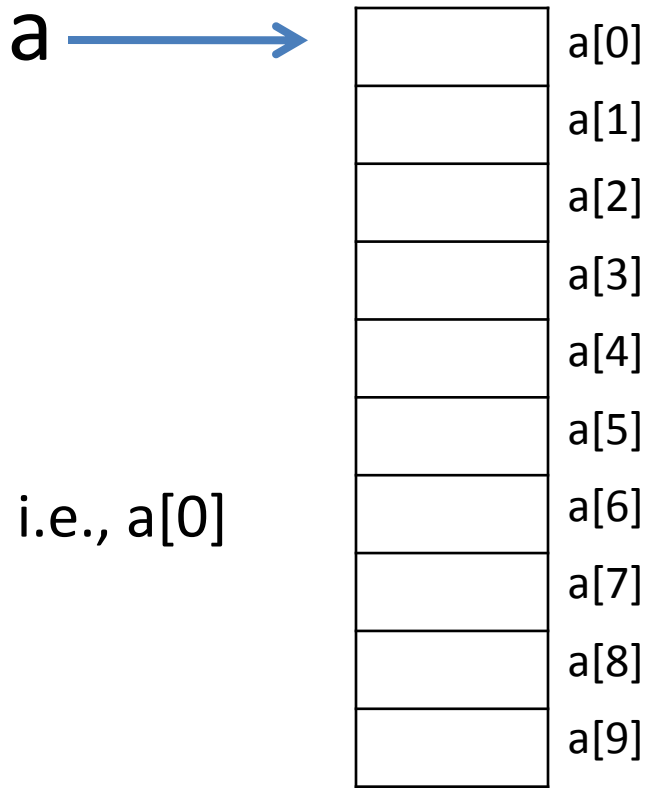
- Collection of data, Arrays
  - Arrays of Numerical Values
  - Accessing Array Elements
  - Initializing Arrays
  - Strings: Arrays of characters
  - Multi-dimensional Arrays

- Functions
  - Function Definition
  - Function Call(call-by-value)
  - Function Prototypes(header files)
  - Recursion

# Arrays

## Arrays of Numerical Values

- Array Declaration:
  - *type* name[SIZE];
- Ex: int a[10];
  - Length: 10
  - Size: 10 x sizeof(int)
  - Location of a: the first element of a; i.e., a[0]
- Ex: float b[20];
  - Length: 20
  - Size: 20 x sizeof(float)



# Arrays

## Accessing Array Elements

```
/* declaration */
int a[10];


/* I can use the elements of an array like a variable */
int b = a[8];
int c = 25 + a[2] - a[8] / a[0];


/* Like a variable, I can assign values to the elements */
a[2] = 25;
a[i] += 25 – a[2]++;
```

# Arrays

**Initializing Arrays**

/* The following two are equivalent */

int a[3] = {1, 2, 3};

int a[] = {1, 2, 3};

float c[] = {.1 2.2 0.3};

/* If the number of initializers is less than the size of the array, the remaining ones are set to zero */

int a[8] = {1, 2, 3}; ➔ int a[8] = {1, 2, 3, 0, 0, 0, 0, 0};

# Arrays

**Strings: Arrays of characters**

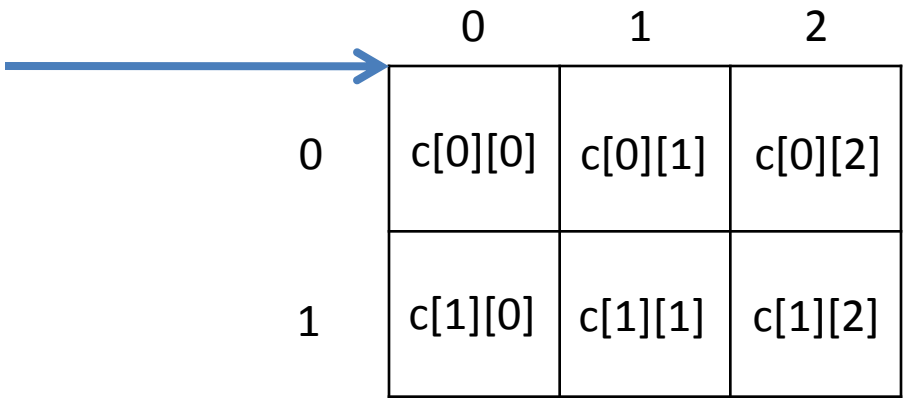
- char  a[3] = “AB”; ➔ char a[3] = {‘A’, ‘B’, ‘\0’};
- char a[] = “AB”; ➔ char a[3] = {‘A’, ‘B’, ‘\0’};
- char b[2] = “AB”; ➔  char b[2] = {‘A’, ‘B’};
  - You cannot use string functions on b since it does not have an ending mark, i.e., ‘\0’.

# Arrays

## Multi-dimensional Arrays

```
int a[] = {1, 2, 3};
int c[2][3] = {
                {1, 2, 3},
                {2, 3, 4},
                };

int d[2][2][3] = {
        { {1, 2, 3},    {2, 3, 4}},
        { {5, 6, 3},    {7, 8, 4}},
        };
```

C



# Notes

- Arrays cannot be copied
  - int a[10], b[10];
  - a = b; ➔ error!
  - Correct way: for(i=0; i < 10; i++) a[i] = b[i];
- Arrays cannot be automatically initialized to a value:
  - int a[10];
  - a = 0; ➔ error!
  - Correct way: for(i=0; i < 10; i++) a[i] = 0;
- If you try to access an array's element with negative index or with an index which is bigger than its length, you would get a run-time error.

# Examples

- Write a program that asks the user to type 10 integers of an array. The program must compute and write the number of integers greater or equal to 10.

# Solution

```
#include <stdio.h>

int N=10;
int main()
{
	int t[10], i, nb = 0;
	printf("Type 10 integers: \n“);

	for (i=0; i<N; i++)
	{
		scanf(“%d”, &t[i]);
		nb +=  ( t[i] >= 10 );	// note that true converts to 1, false to 0
	}
	printf("the number of integers greater or equal to 10 is: %d\n“, nb);

return 0;
}
```

# Example

- Write a C program that gets the number of students, then grades of students and calculate the average of the class, number of students which are under the average.

- Sample I/O:

N grade1 grade2 .. gradeN

İnput:

3 55 45 80

Output:

average: 60

2 students are under the average.

# Example

- Write a program that asks the user to type 10 integers of an array and an integer V. The program must search if V is in the array of 10 integers. The program writes "V is in the array" or "V is not in the array".

# Solution

```
#include <stdio.h>

int N = 10;
int main (){
	int t[N], i=0, V ;

	printf("Type 10 integers: \n“);
	for (i = 0; i < N; i++)
		scanf(“%d”, &t[i]);

	printf("Type the value of V: “);
	scanf(“%d”, &V);
	for (i = 0; i < N; i++){
		if (t[i] == V)
		{
		   printf("V is in the array\n“);		//If found, write and return
		   return 0;
		}
	}
		printf("V is not in the array\n“);		//If not found, write and return
return 0;
}
```

# Functions

**Function definition**

```
return_type function_name(parameter declarations)
{
    statement-1;
    statement-2;
...
}
```

- if is *return_type* not void, “return” statement has to be used:

```
return expression;
```

# Functions

**Function declaration**

- *return_type* function_name(list-of-params);
- The parameters have to have the same types as in the function definition although the names of the parameters may differ.
- Example:
  - int factorial(int N);
  - void print_matrix(int matrix[N][M]);
- If a function is used before it is defined, it has to be declared first.

# Functions

**Function call**

function_name(list of arguments)

- Example:
  - Function declaration:

    int greatest(int A, int B, int C);
  - Example function call:

    printf("%d\n", greatest(10, 20, -10));

# Functions

## Call by Value

- The arguments of the function are just copies of the passed data!

```
void f(int a)
{
    a = 10 * a;
}
void g(int b)
{
    b = 10;
    f(b);
    printf("%d",  b);
}
```

# Functions

## Call by Value

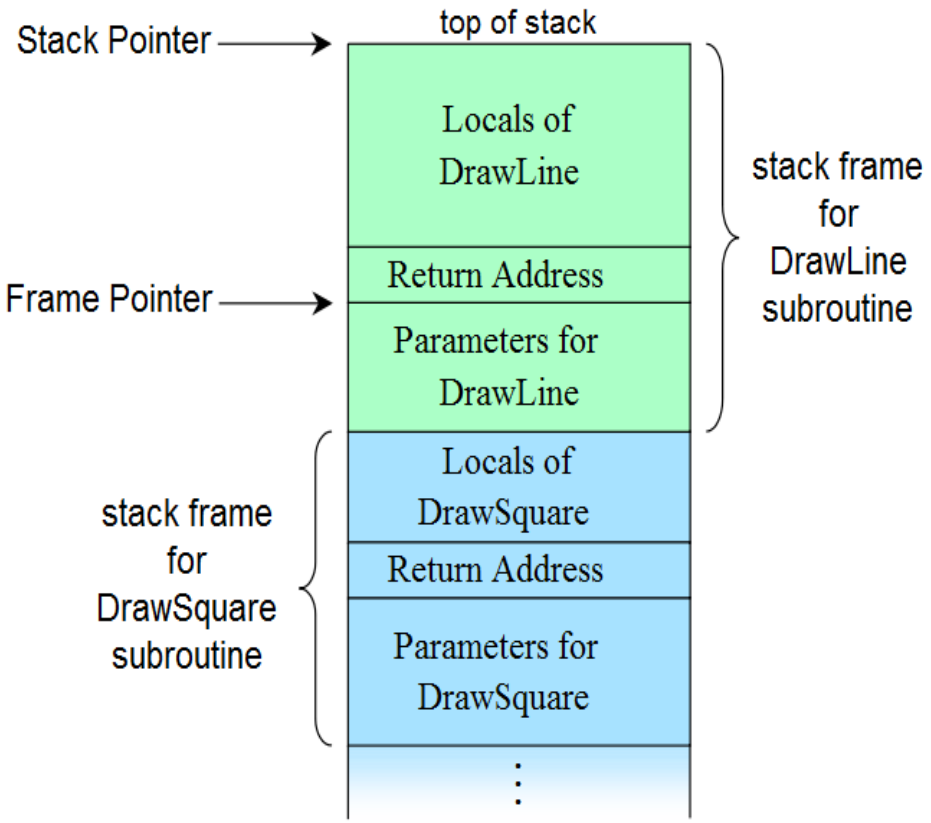
- So, what do we do? How can I get the changed value?
  - You can use the “return” statement for a variable.
  - If you have more than one variable, you can use global variables.
  - Or, you can use pointers!

# Functions

## Tracking Function Calls

- Function calls are

"traced" using call stack.



# Functions

**Namespaces**

- Determines where the definition of variables are valid!
- Global space.
- main() function space.
- Block structures.

# Namespace Example

```
1  #include<stdio.h>
2  int a;
3
4  void f(int a)
5  { printf("a in f() = %d\n", a); }
6
7  void g()
8  { int a = 30; printf("a in g() = %d\n", a); }
9
10 void h()
11 { printf("a in h() = %d\n", a); }
12
13 int main()
14 {
15 int a = 10;
16
17         { int a = 20; printf("a in block structure = %d\n", a); }
18
19         printf("a in main() = %d\n", a);
20
21         f(a);
22         g();
23         h();
24
25 return 0;
26 }
```

**Output:**

a in block structure = 20
a in main() = 10
a in f() = 10
a in g() = 30
a in h() = 0

# Block Structure

```
void f(int bP)
{
int aL;
        aL = 3;

        {
                int aL;
                aL = 4;
                printf("aL = %d\n", aL);
        }
        printf("aL = %d\n", aL);
}
```

- Block Structure: Statements enclosed within braces.

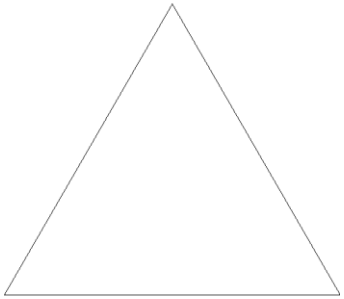
# Functions

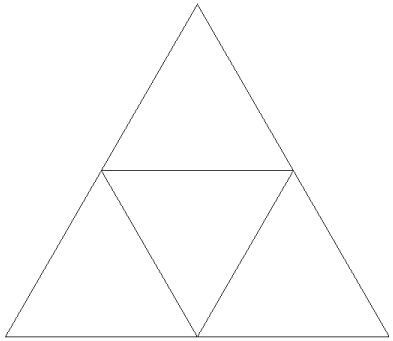
## Function Prototypes

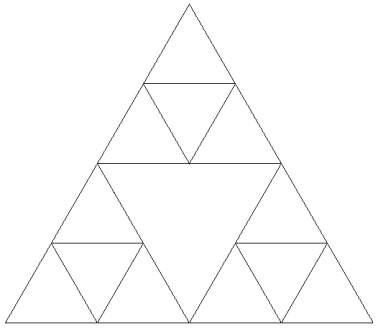
```
#include <stdio.h>
/* * If this prototype is provided, the compiler will catch the error * in main(). If it is
     omitted, then the error may go unnoticed. */
int fac(int n);                          /* Prototype */
int main(void){
     printf("%d\n", fac());              /* Calling function */
                                         /* Error: forgot argument to fac */
     return 0;}

 int fac(int n){                         /* Called function */
     if (n == 0)
          return 1;
     else
          return n * fac(n - 1); }
```

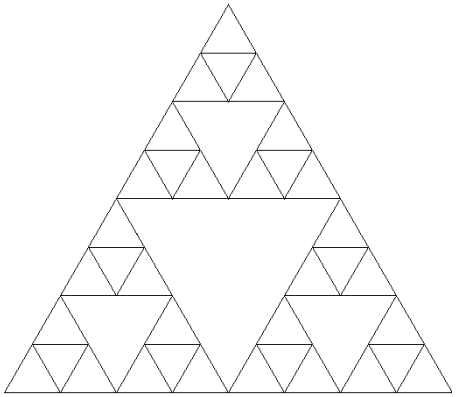
# Functions

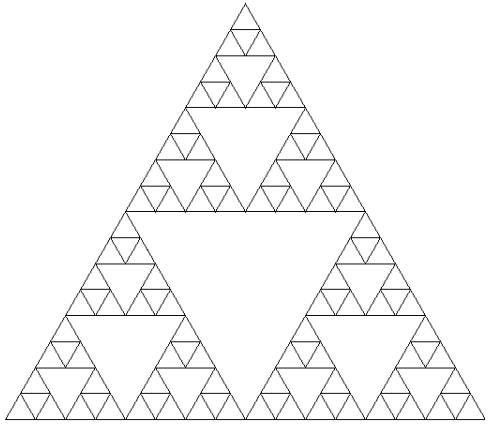
## Recursion





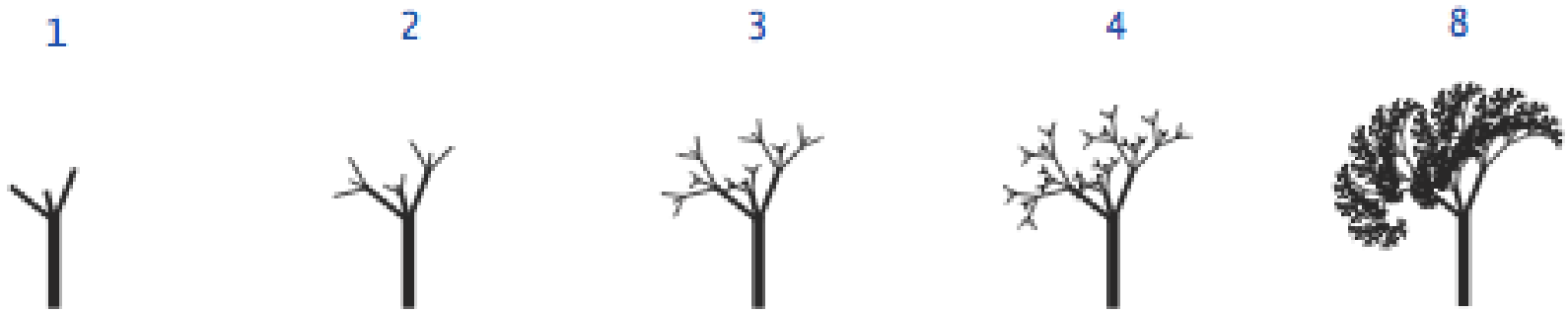






# Recursion

## Recursive Tree



# Recursion

## Towers of Hanoi

$$\text{hanoi}(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2 \cdot \text{hanoi}(n-1) + 1 & \text{if } n > 1 \end{cases}$$

Computing the recurrence relation for n = 4:

hanoi(4) = 2*hanoi(3) + 1
= 2*(2*hanoi(2) + 1) + 1
= 2*(2*(2*hanoi(1) + 1) + 1) + 1
= 2*(2*(2*1 + 1) + 1) + 1
= 2*(2*(3) + 1) + 1
= 2*(7) + 1 = 15

# Functions

- Recursion
  - Function calls itself.

- Limit:
  - The stack space!

```
int factorial(int nP)
{
        if(nP <= 1 )
                return 1;

return nP * f(nP);
}
```

# Recursion vs Iteration

- Recursion:
  - Limited number of calls
  - Easier to formulate/write
- Iteration:
  - Not limited
  - More difficult to formulate/write

# Examples

- Write two functions named reverse1 and reverse2 with the prototypes:

```
void reverse1(int number);
int reverse2(int number, int reversed);
```

- In order to test them write a main function

# Solutions

```
void reverse1(int number)
{
    int remain;
    if(number < 10)
    {       printf("%d", number);
            return;
    }

    remain = number%10;
    printf("%d", remain);

    reverse1(number/10);
    return;
}
```

# Solutions

```
int reverse2(int number, int reversedNumber)
{
//    printf("%d %d\n", number, reversedNumber);
                              /*if you want to see step by step delete comment of printf*/
    int remain;
    if(number < 10)
    {
        reversedNumber += number;
        return reversedNumber;
    }
    remain = number%10;
    reversedNumber = (reversedNumber + remain) * 10;
    number /= 10;
    return reverse2(number, reversedNumber);
}
```

# Solutions

```
#include<stdio.h>

void reverse1(int number);
int reverse2(int number, int reversedNumber);

int main()
{
	int number = 123;
	int reversedNumber = 0;
	//scanf("%d",&number);
	reverse1(number);
	reversedNumber = reverse2(number, 0);
	printf("\n%d\n", reverse2(number,0));

	  return 0;
}
```

For this main function output is:

321

321

# Examples

- Write a function that takes a positive integer as input and returns the leading digit in its decimal representation. For example, the leading digit of 234567 is 2.
- Write a boolean function that takes a positive integer n as an argument and returns true if n is prime, and false otherwise.

  Bool isPrime(int number);

# Another simple example for recursion: Fibonacci numbers

- Let us write the recursive and the iterative solutions to fibonacci numbers

$$\text{fib}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{if } n >= 2 \end{cases}$$

# Naming Conventions

- One option:
  - Append G to the end of the global variables.
    - Ex: bufferG, arrayG, namesG, wordsG
  - Append P to the end of parameters:
    - Ex: indexP, rangeP, numberP
  - Append L to the end of local variables:
    - Ex: tempL, indexL, resultL

# Naming Conventions

- Second option (should be combined with the first):
  - Use “i_” as a prefix for integers:
    - Ex: int i_number;
  - Use “f_” as a prefix for floats:
    - Ex: float f_division;
  - Use “c_” as a prefix for characters.
  - Use “str_” as a prefix for strings.
  - Use “a_” for arrays:
    - Ex: int i_a_numbers[10];

# Modular Programming

- **"a.c" file:**

```
#include "a.h"
int main()
{
        f();
return 0;
}
void f()
{
.....
}
```

- **"a.h" file:**

```
/* Include Directives */
#include<stdio.h>

/* Global Variables */
int flagG;
char wordG[10];

/* Function Declarations */
void f();
```