

C Language Tutorial for Beginners (with Notes & Practice Questions)

Timestamps:

Introduction: 00:00:00

Installation (VS Code) : 00:01:27

Compiler + Setup: 00:04:31

Chapter 1 - Variables, Data types + Input/Output: 00:10:12

Chapter 2 - Instructions & Operators: 00:51:54

Chapter 3 - Conditional Statements: 02:01:15

Chapter 4 - Loop Control Statements: 02:45:36

Chapter 5 - Functions & Recursion: 03:51:34

Chapter 6 - Pointers: 05:21:30

Chapter 7 - Arrays: 06:18:50

Chapter 8 - Strings: 07:22:29

Chapter 9 - Structures: 08:26:20

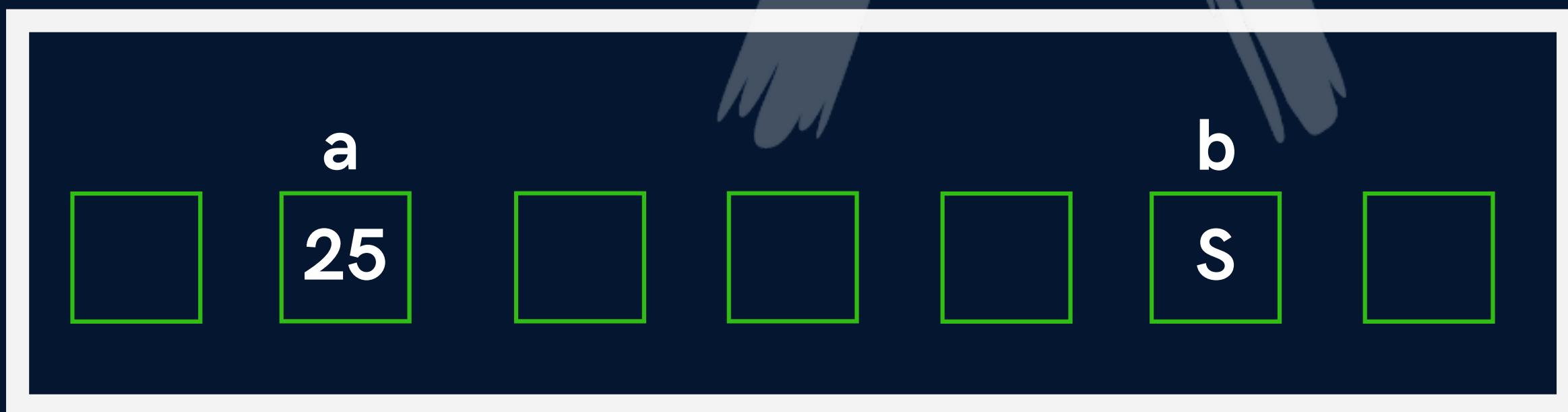
Chapter 10 - File I/O: 09:22:15

Chapter 11 - Dynamic Memory Allocation: 10:05:00

Variables

Variable is the name of a memory location which stores some data.

Memory



Variables

Rules

- a. Variables are case sensitive
- b. 1st character is alphabet or '_'
- c. no comma/blank space
- d. No other symbol other than '_'

Variables

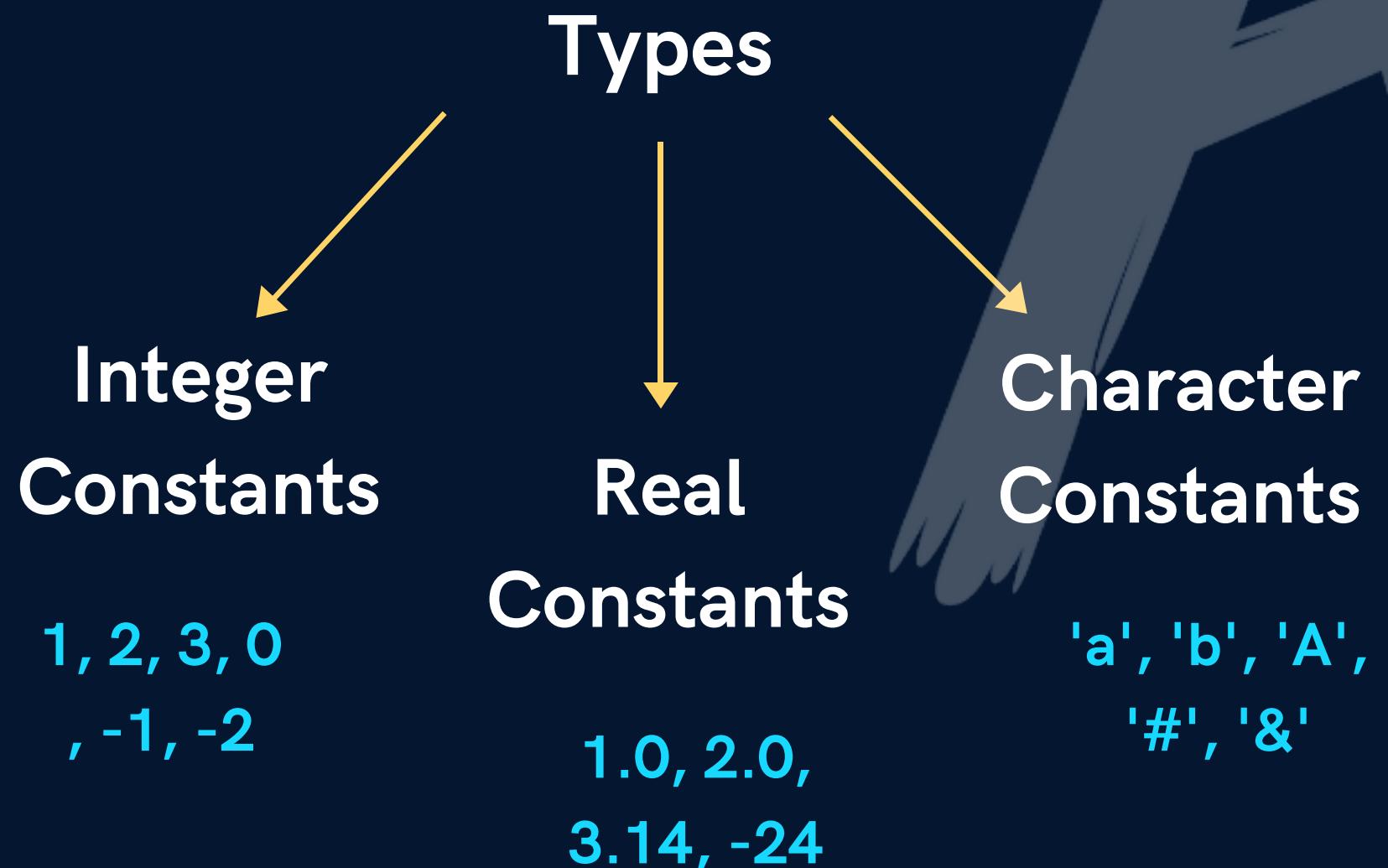
Data Types

Data type	Size in bytes
Char or signed char	1
Unsigned char	1
int or signed int	2
Unsigned int	2
Short int or Unsigned short int	2
Signed short int	2
Long int or Signed long int	4
Unsigned long int	4
float	4
double	8
Long double	10



Constants

Values that don't change(fixed)



Keywords

Reserved words that have **special** meaning to the compiler



32 Keywords in C

Keywords

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

Program Structure

```
#include<stdio.h>

int main() {
    printf("Hello World");
    return 0;
}
```

Comments

Lines that are not part of program

Single Line

//

Multiple
Line

/*
*/

Output

```
printf(" Hello World ");
```

new line

```
printf(" kuch bhi \n");
```



Output

CASES

1. integers

```
printf(" age is %d ", age);
```

2. real numbers

```
printf(" value of pi is %f ", pi);
```

3. characters

```
printf(" star looks like this %c ", star);
```

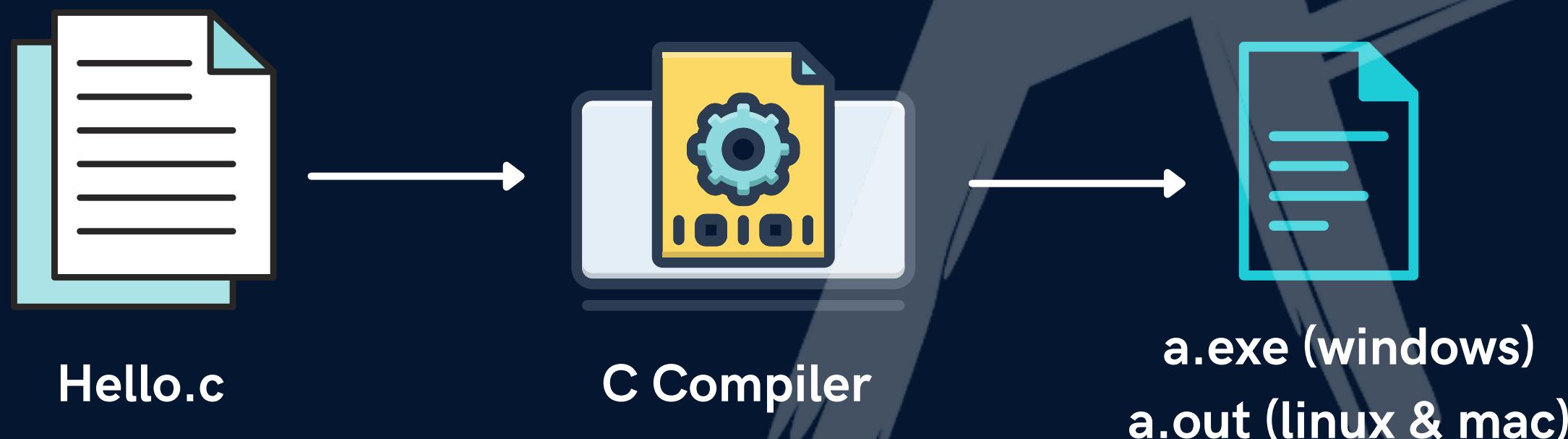


Input

```
scanf(" %d ", &age);
```

Compilation

A computer program that translates C code
into machine code



C Language Tutorial

(Basic to Advanced)

Topics to be covered :

Installation + Setup
Chapter 1 - Variables, Data types + Input/Output
Chapter 2 - Instructions & Operators
Chapter 3 - Conditional Statements
Chapter 4 - Loop Control Statements
Chapter 5 - Functions & Recursion
Chapter 6 - Pointers
Chapter 7 - Arrays
Chapter 8 - Strings
Chapter 9 - Structures
Chapter 10 - File I/O
Chapter 11 - Dynamic Memory Allocation

Variables, Data Types + Input/Output

(Chapter 1)

1. First Program

```
#include<stdio.h>

int main() {
    printf("Hello World");
    return 0;
}
```

2. Variables & Data Types + Constants & Keywords

```
#include<stdio.h>

int main() {
    int number;
    int age;
    int price;
    return 0;
}
```

```
#include<stdio.h>

int main() {
    int age = 22;
    float pi = 3.14;
    char percentage = '%';
    return 0;
}
```

3. Comments

```
#include<stdio.h>
//This program prints Hello World
int main() {
    printf("Hello World");
    return 0;
}
```

4. Output

```
#include<stdio.h>

int main() {
    int age = 22;
    float pi = 3.14;
    char percentage = '%';

    printf("age is %d", age);
    printf("age is %f", pi);
    printf("age is %c", percentage);
    return 0;
}
```

5. Input (Sum of 2 numbers)

```
#include<stdio.h>

int main() {
    int a, b;

    printf("enter a \n");
    scanf("%d", &a);

    printf("enter b \n");
```

```
    scanf("%d", &b);

    printf("sum of a & b is : %d \n", a+b);

    return 0;
}
```

6. Practice Qs 1 (Area of Square)

```
#include<stdio.h>
//area of square
int main() {
    int side;
    scanf("%d", &side);
    printf("%d", side * side);
    return 0;
}
```

7. Practice Qs 2 (Area of Circle)

```
#include<stdio.h>
//area of square
int main() {
    float radius;
    scanf("%f", &radius);
    printf("%f", 3.14 * radius * radius);
    return 0;
}
```

Instructions

These are statements in a Program

Types

Type Declaration
Instructions

Arithmetic
Instructions

Control
Instructions

Instructions

Type Declaration Instructions

VALID

```
int a = 22;  
int b = a;  
int c = b + 1;  
int d = 1, e;
```

```
int a,b,c;  
a = b = c = 1;
```

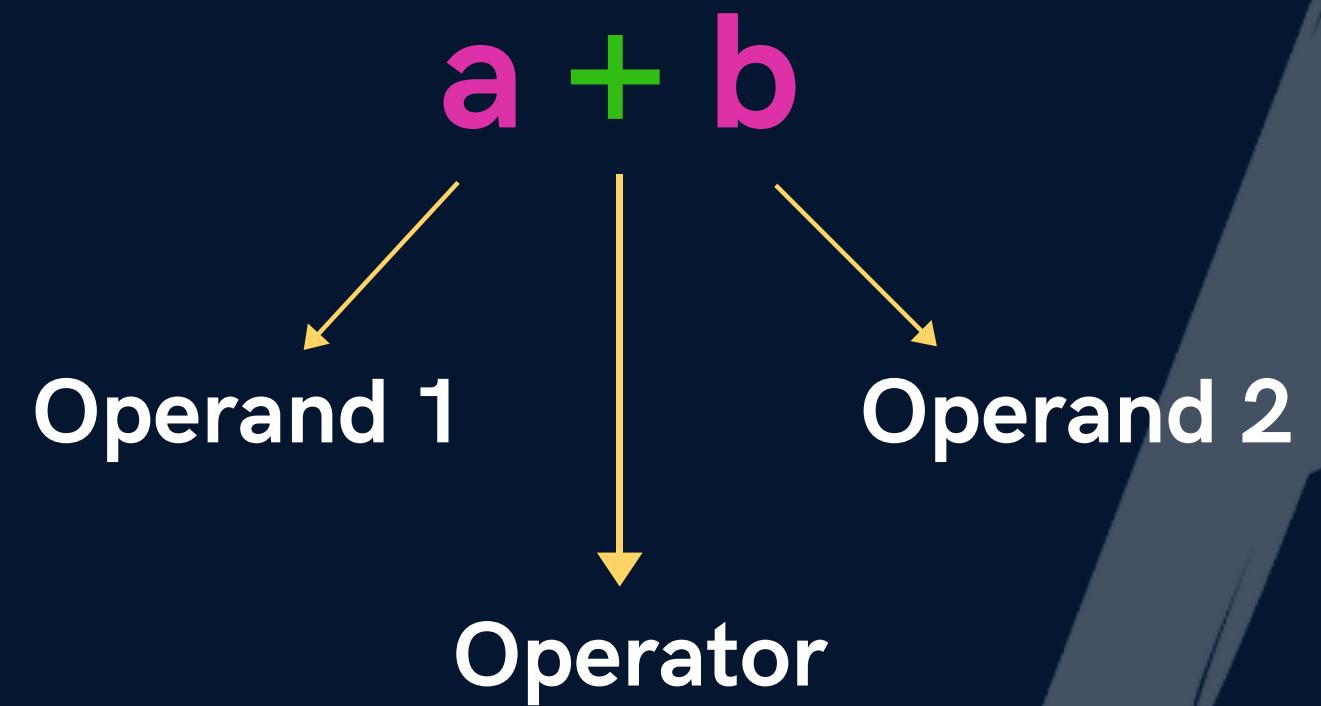
INVALID

```
int a = 22;  
int b = a;  
int c = b + 2;  
int d = 2, e;
```

```
int a,b,c = 1;
```

Declare var before using it

Arithmetic Instructions



NOTE - single variable on the LHS

Arithmetic Instructions

VALID

$a = b + c$

$a = b * c$

$a = b / c$

INVALID

$b + c = a$

$a = bc$

$a = b^c$

NOTE - $\text{pow}(x,y)$ for x to the power y

Arithmetic Instructions

★ Modular Operator %

Returns remainder for int

$$3 \% 2 = 1$$

$$-3 \% 2 = -1$$

Arithmetic Instructions

Type Conversion

int op int \longrightarrow int

int op float \longrightarrow float

float op float \longrightarrow float

Arithmetic Instructions

Operator Precedence

$\ast, /, \%$



$+, -$



$=$

$x = 4 + 9 * 10$

$x = 4 * 3 / 6 * 2$

Arithmetic Instructions

Associativity (for same precedence)

Left to Right

$$x = 4 * 3 / 6 * 2$$

Instructions

Control Instructions

Used to determine flow of program

- a. Sequence Control
- b. Decision Control
- c. Loop Control
- d. Case Control

Operators

a. Arithmetic Operators

b. Relational Operators

c. Logical Operators

d. Bitwise Operators

e. Assignment Operators

f. Ternary Operator



Operators

Relational Operators

`==`

`>, >=`

`<, <=`

`!=`



Operators

Logical Operators

&& AND

|| OR

! NOT



Operator Precendence

Priority

1

2

3

4

5

6

7

8

Operator

!

* , / , %

+ , -

< , <= , > , >=

== , !=

&&

||

=

Operators

Assignment Operators

=

+=

-=

*=

/=

%=



C Language Tutorial

(Basic to Advanced)

Topics to be covered :

Installation + Setup
Chapter 1 - Variables, Data types + Input/Output
Chapter 2 - Instructions & Operators
Chapter 3 - Conditional Statements
Chapter 4 - Loop Control Statements
Chapter 5 - Functions & Recursion
Chapter 6 - Pointers
Chapter 7 - Arrays
Chapter 8 - Strings
Chapter 9 - Structures
Chapter 10 - File I/O
Chapter 11 - Dynamic Memory Allocation

Instructions & Operators

(Chapter 2)

1. Type Declaration Instructions

```
#include<stdio.h>

int main() {
    int age = 22;
    int oldAge = age;
    int newAge = oldAge + 2;
    printf("new age is : %d", newAge);

    int rupee = 1, dollar;
    dollar = 74;

    /*
        order of declaration is important - Wrong Declaration Order
        float pi = 3.14;
        float area = pi * rad * rad;
        float rad = 3;
    */
}
```

```
// valid declaration
int age1, age2, age3;
age1 = age2 = age3 = 22;

//invalid
//int a1 = a2 = a3 = 22;

return 0;
}
```

2. Arithmetic Instructions

```
#include<stdio.h>

int main() {
    int a = 1, b = 2, c = 3;
    //valid
    a = b + c;

    //invalid
    // b + c = a;

    printf("%d \n", 3 % 2);
    printf("%d \n", -3 % 2);
    return 0;
}
```

> Type Conversion

```
#include<stdio.h>

int main() {
    printf("sum of 2 & 3 : %d", 2 + 3);
    printf("sum of 2.0 & 3 : %f", 2.0 + 3);
    printf("sum of 2.0 & 3.0 : %f", 2.0 + 3.0);
    return 0;
}
```

> Associativity

```
#include<stdio.h>

int main() {
    printf(" Output : %d", 5+2/2*3);
```

```
    return 0;  
}
```

3. Relational Operator

```
#include<stdio.h>  
  
int main() {  
    printf("%d \n", 4==4);  
  
    printf("%d \n", 4<3);  
    printf("%d \n", 3<4);  
    printf("%d \n", 4<4);  
    printf("%d \n", 4<=4);  
  
    printf("%d \n", 4>3);  
    printf("%d \n", 3>4);  
    printf("%d \n", 4>4);  
    printf("%d \n", 4>=4);  
  
    printf("%d \n", 4 !=4);  
    printf("%d \n", 3 !=4);  
    return 0;  
}
```

4. Logical Operator

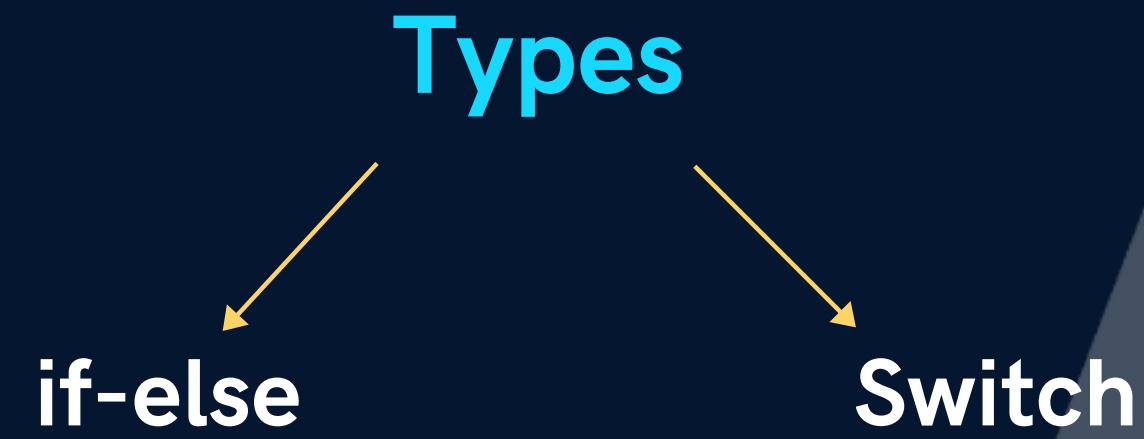
```
#include<stdio.h>  
  
int main() {  
    printf("%d \n", 3<4 && 3<5);  
    printf("%d \n", 3<4 && 5<4);  
  
    printf("%d \n", 3<4 && 5<4);  
    printf("%d \n", 3>4 && 5>4);  
    printf("%d \n", 3<4 && 3<5);  
  
    printf("%d \n", !(3<4 && 3<5));  
    printf("%d \n", !(4<3 || 5<3));  
    return 0;  
}
```

5. Assignment Operator

```
# include <stdio.h>

int main() {
    int a = 10;
    a += 10;
    printf("a+10 = %d \n", a);
    a -= 10;
    printf("a-10 = %d \n", a);
    a *= 10;
    printf("a*10 = %d \n", a);
    a /= 10;
    printf("a/10 = %d \n", a);
    a %= 10;
    printf("a%c10 = %d \n", '%', a);
    return 0;
}
```

Conditional Statements



if-else

```
if(Condition) {  
    //do something if TRUE  
}  
  
else {  
    //do something if FALSE  
}
```



Ele is optional block
can also work without {}

else if

```
if(Condition 1) {  
    //do something if TRUE  
}  
  
else if (Condition 2) {  
    //do something if 1st is FALSE & 2nd is TRUE  
}
```

Conditional Operators

Ternary

Condition ? doSomething if TRUE : doSomething if FALSE;

Conditional Operators

switch

```
switch(number) {  
    case C1: //do something  
        break;  
  
    case C2 : //do something  
        break;  
  
    default : //do something  
}
```



Conditional Operators

switch Properties

- a. Cases can be in any order
- b. Nested switch (switch inside switch) are allowed

C Language Tutorial

(Basic to Advanced)

Topics to be covered :

Installation + Setup

Chapter 1 - Variables, Data types + Input/Output

Chapter 2 - Instructions & Operators

Chapter 3 - Conditional Statements

Chapter 4 - Loop Control Statements

Chapter 5 - Functions & Recursion

Chapter 6 - Pointers

Chapter 7 - Arrays

Chapter 8 - Strings

Chapter 9 - Structures

Chapter 10 - File I/O

Chapter 11 - Dynamic Memory Allocation

Conditional Statements

(Chapter 3)

1. If-else

```
#include<stdio.h>

int main() {
    int age = 19;
    if(age >= 18) {
        printf("you are an adult");
    }
    else {
        printf("you are not an adult");
    }
    return 0;
}
```

> check if a number is odd or even

```
#include<stdio.h>

int main() {
    int number;
```

```
scanf("%d", &number);

if(number % 2 == 0) {
    printf("even");
}
else {
    printf("odd");
}
return 0;
}
```

> Use of else if

```
#include<stdio.h>

int main() {
    int age;
    printf("Enter age : ");
    scanf("%d", &age);

    if(age < 12) {
        printf("child");
    }
    else if(age < 18) {
        printf("teenager");
    }
    else {
        printf("adult");
    }
    return 0;
}
```

2. Ternary Operator

```
#include<stdio.h>

int main() {
    int age;
    printf("Enter age : ");
    scanf("%d", &age);

    age > 18 ? printf("adult \n") : printf("not adult \n");

    int number = 7;
```

```
int luckyNumber = 7;

    number == luckyNumber ? printf("you are lucky \n") : printf("you are not
lucky \n");

    return 0;
}
```

3. Switch (integer)

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

int main() {
    int day = 5;
    switch(day) {
        case 1 : printf("monday \n");
                   break;
        case 2 : printf("tuesday \n");
                   break;
        case 3 : printf("wednesday \n");
                   break;
        case 4 : printf("thursday \n");
                   break;
        case 5 : printf("friday \n");
                   break;
        case 6 : printf("saturday \n");
                   break;
        case 7 : printf("sunday \n");
                   break;
    }
    return 0;
}
```

4. Switch (character)

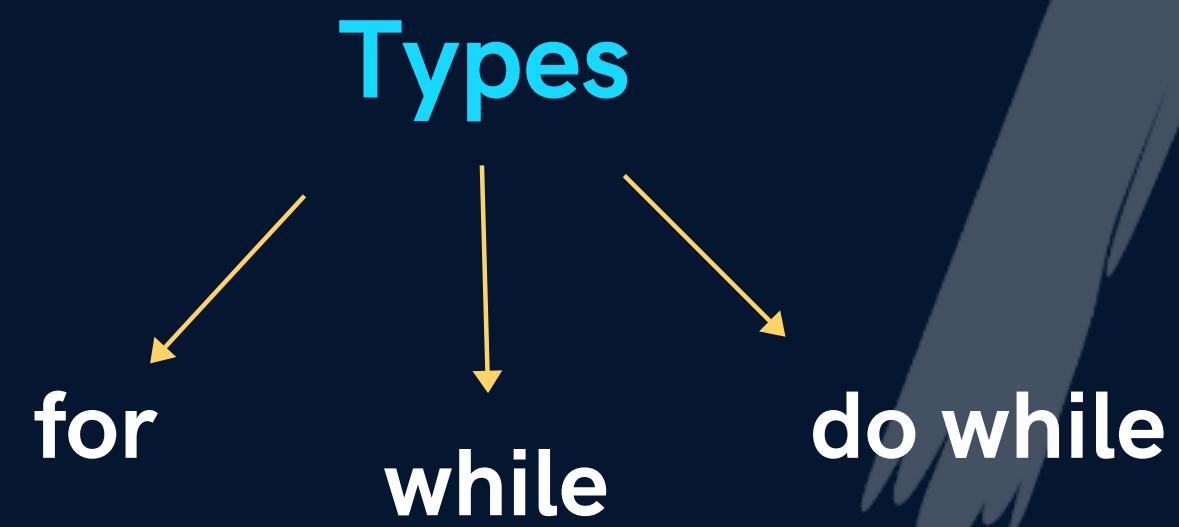
```
#include<stdio.h>
#include<math.h>

int main() {
    char day = 'f';
    switch(day) {
        case 'm' : printf("monday \n");
                    break;
```

```
case 't' : printf("tuesday \n");
            break;
case 'w' : printf("wednesday \n");
            break;
case 'T' : printf("thursday \n");
            break;
case 'f' : printf("friday \n");
            break;
case 's' : printf("saturday \n");
            break;
case 'S' : printf("sunday \n");
            break;
}
return 0;
}
```

Loop Control Instructions

To repeat some parts of the program



for Loop

```
for(initialisation; condition; updation) {
```

```
//do something
```

```
}
```

Special Things

- Increment Operator
- Decrement Operator
- Loop counter can be float
or even character
- Infinite Loop



while Loop

```
while(condition) {
```

```
//do something
```

```
}
```



A small vertical image in the bottom left corner showing a cartoon illustration of Winnie the Pooh and his friend Eeyore from the children's book series by A.A. Milne.

```
while (true)
{
}
for (; ;){
}
```

do while Loop

```
do {
```

```
//do something
```

```
} while(condition);
```



break Statement



exit the loop



continue Statement



skip to next iteration

Nested Loops

```
for( .. ) {  
    for( .. ) {  
    }  
}  
}
```



C Language Tutorial

(Basic to Advanced)

Topics to be covered :

Installation + Setup
Chapter 1 - Variables, Data types + Input/Output
Chapter 2 - Instructions & Operators
Chapter 3 - Conditional Statements
Chapter 4 - Loop Control Statements
Chapter 5 - Functions & Recursion
Chapter 6 - Pointers
Chapter 7 - Arrays
Chapter 8 - Strings
Chapter 9 - Structures
Chapter 10 - File I/O
Chapter 11 - Dynamic Memory Allocation

Loop Control Statements

(Chapter 4)

1. Syntax of 3 Loops

```
# include <stdio.h>

int main () {
    //for loop
    for(int i=1; i<=100; i++) {
        printf("%d\n", i);
    }

    //while loop
    int i=1;
    while(i<=100) {
        printf("%d\n", i);
        i++;
    }

    //do while loop
    i = 1;
    do {
```

```
    printf("%d\n", i);
    i++;
} while (i<=100);

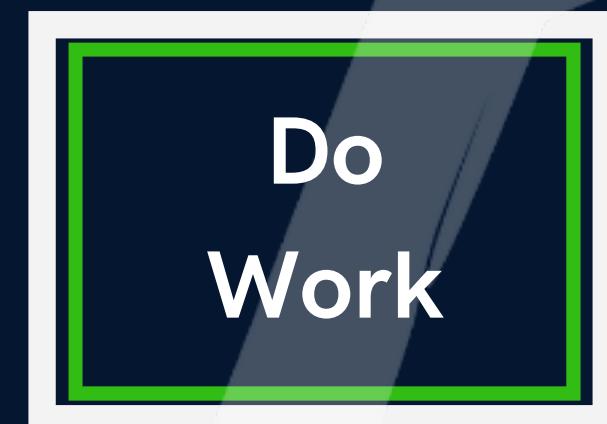
return 0;
}
```

Functions



block of code that performs particular task

Take
Argument



Return
Result

it can be used **multiple** times

increase code **reusability**

Syntax 1

Function Prototype

```
void printHello(); ←
```



> Tell the compiler

Syntax 2

Function Definition

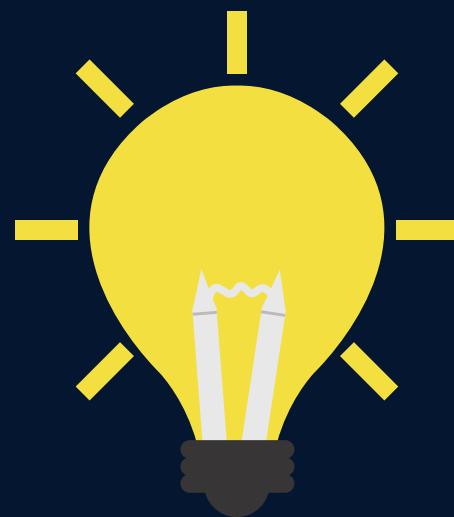
```
void printHello() {  
    printf("Hello");    ←  
}  
  
  
A stylized letter 'A' is positioned behind the code. A hand with a magnifying glass is shown from the bottom left, pointing towards the middle of the letter 'A'. A yellow arrow points from the magnifying glass towards the code line 'printf("Hello");'.
```

> Do the Work

Syntax 3

Function Call

```
int main() {  
    printHello(); ←  
    return 0;  
}
```



> Use the Work

Properties

- Execution always starts from main
- A function gets called directly or indirectly from main
- There can be multiple functions in a program

Function Types

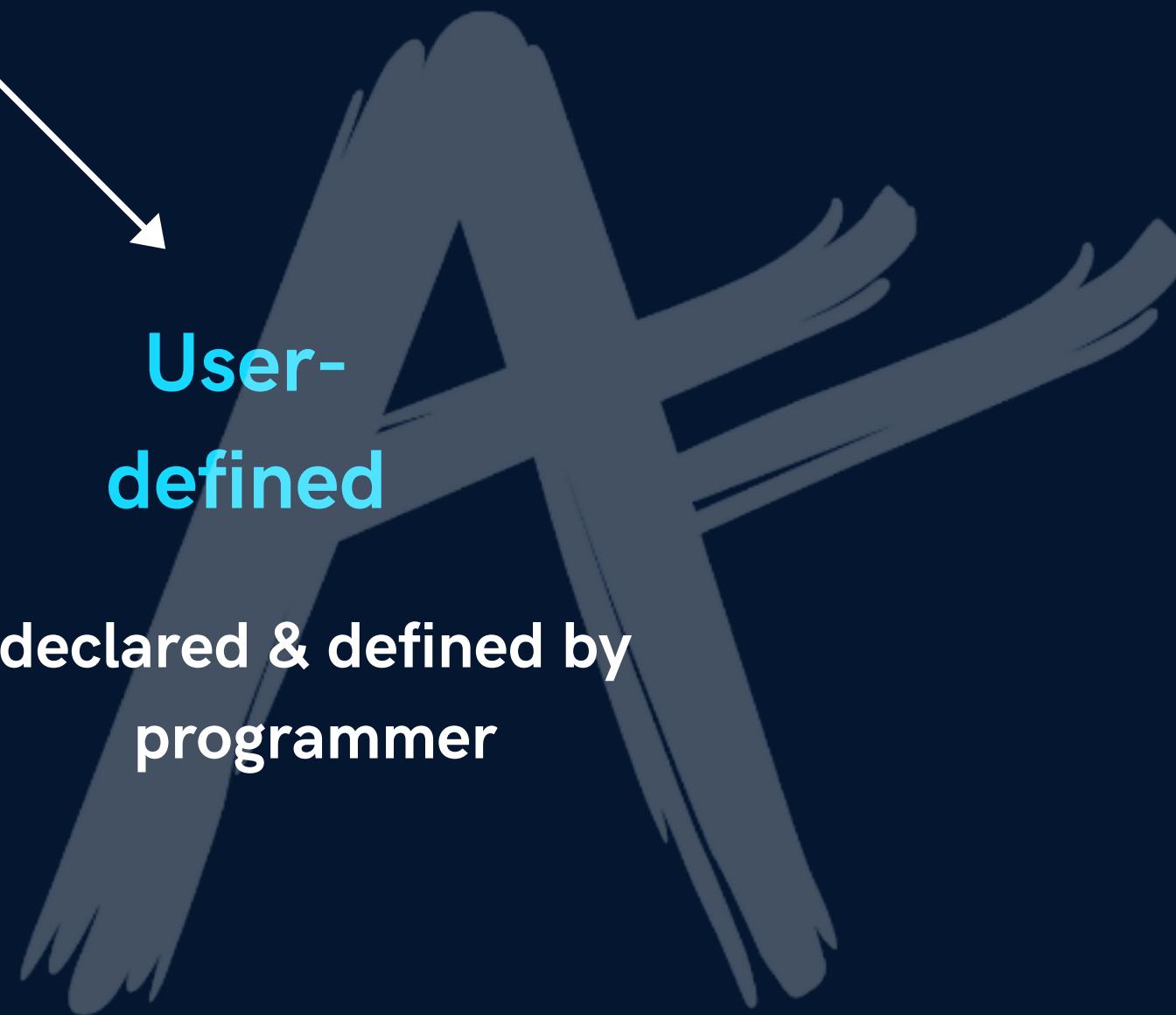
**Library
function**

**Special functions
inbuilt in C**

scanf(), printf()

**User-
defined**

**declared & defined by
programmer**



Passing Arguments

functions can take value & give some value

parameter

return value

Passing Arguments

void **printHello()**; ←

void **printTable(int n)**; ←

int **sum(int a, int b)**; ←



Passing Arguments

functions can take value & give some value

parameter

return value

Argument v/s Parameter

values that are
passed in
function call

used to send
value

actual
parameter

values in function
declaration &
definition

used to receive
value

formal
parameters

NOTE

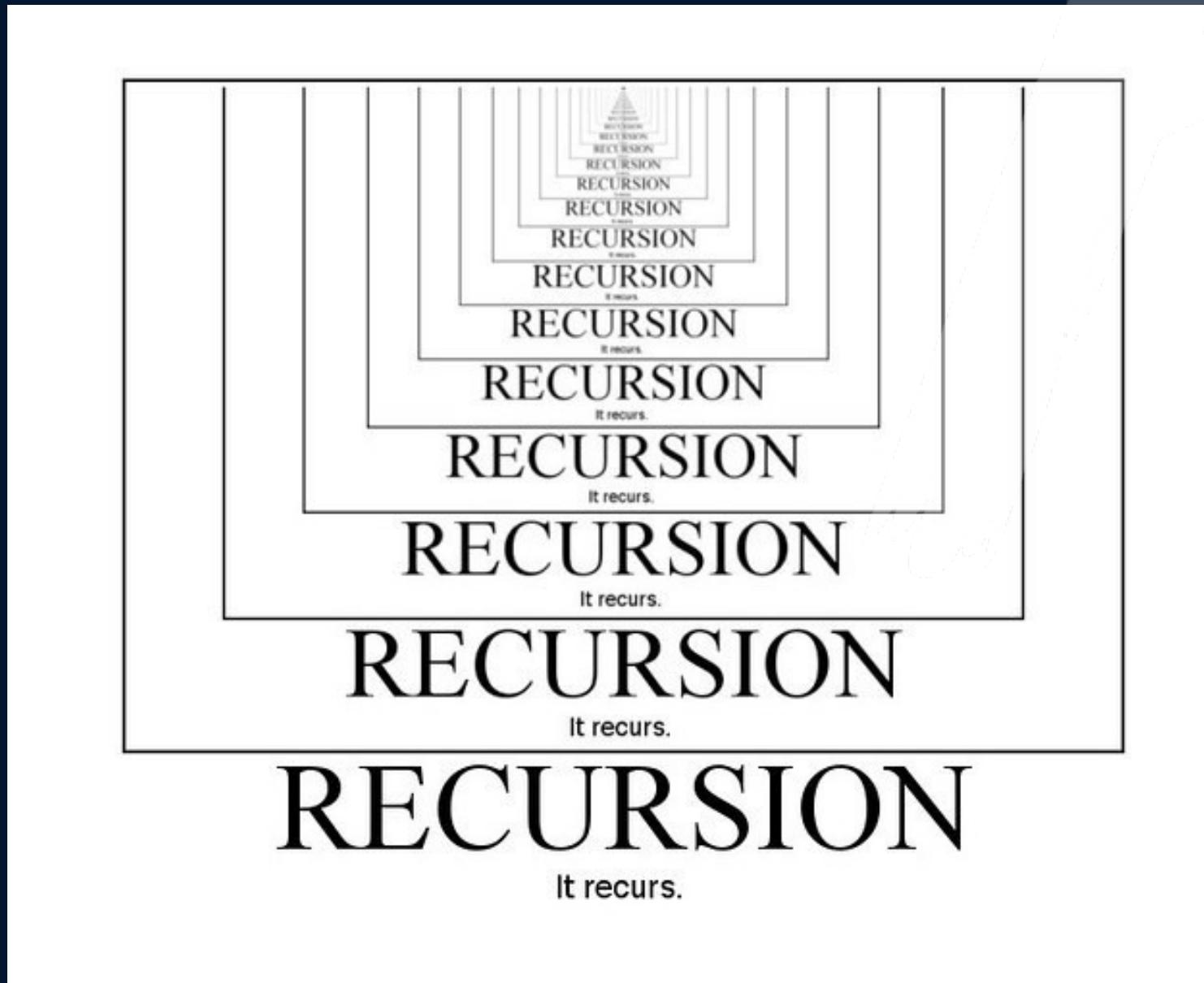
- a. Function can only return one value at a time
- b. Changes to parameters in function don't change the values in calling function.

Because a copy of argument is passed to the function

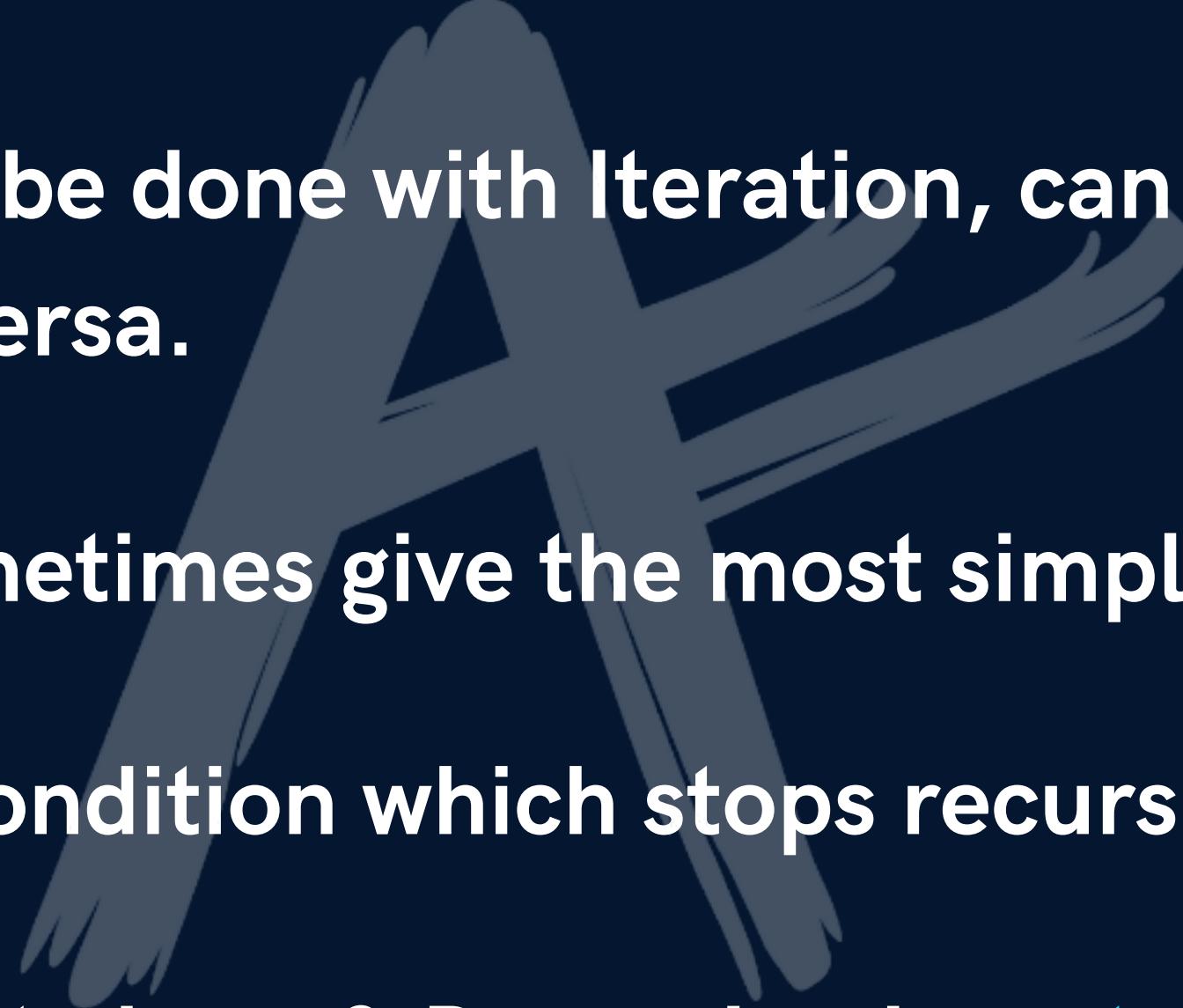
Recursion



When a **function calls itself**, it's called recursion



Properties of Recursion

- 
- a. Anything that can be done with Iteration, can be done with recursion and vice-versa.
 - b. Recursion can sometimes give the most simple solution.
 - c. **Base Case** is the condition which stops recursion.
 - d. Iteration has infinite loop & Recursion has **stack overflow**

C Language Tutorial

(Basic to Advanced)

Topics to be covered :

Installation + Setup
Chapter 1 - Variables, Data types + Input/Output
Chapter 2 - Instructions & Operators
Chapter 3 - Conditional Statements
Chapter 4 - Loop Control Statements
Chapter 5 - Functions & Recursion
Chapter 6 - Pointers
Chapter 7 - Arrays
Chapter 8 - Strings
Chapter 9 - Structures
Chapter 10 - File I/O
Chapter 11 - Dynamic Memory Allocation

Functions & Recursion

(Chapter 5)

1. Function to print Hello

```
#include<stdio.h>

//function declaration/prototype
void printHello();

int main() {
    //function call
    printHello();
    return 0;
}

//function definition
void printHello() {
    printf("Hello!\n");
}
```

2. Function to calculate square of a number

```
# include <stdio.h>
//function to calculate square of a number
int calcSquare(int n);

int main() {
    int n;
    printf("enter n : ");
    scanf("%d", &n);
    printf("square is : %d", calcSquare(n));
    return 0;
}

int calcSquare(int n) {
    return n * n;
}
```

3. Function to calculate n factorial (using recursion)

```
# include <stdio.h>
//function to print factorial of n
int factorial(int n);

int main() {
    int n;
    printf("enter n : ");
    scanf("%d", &n);
    printf("factorial is : %d", factorial(n));
    return 0;
}

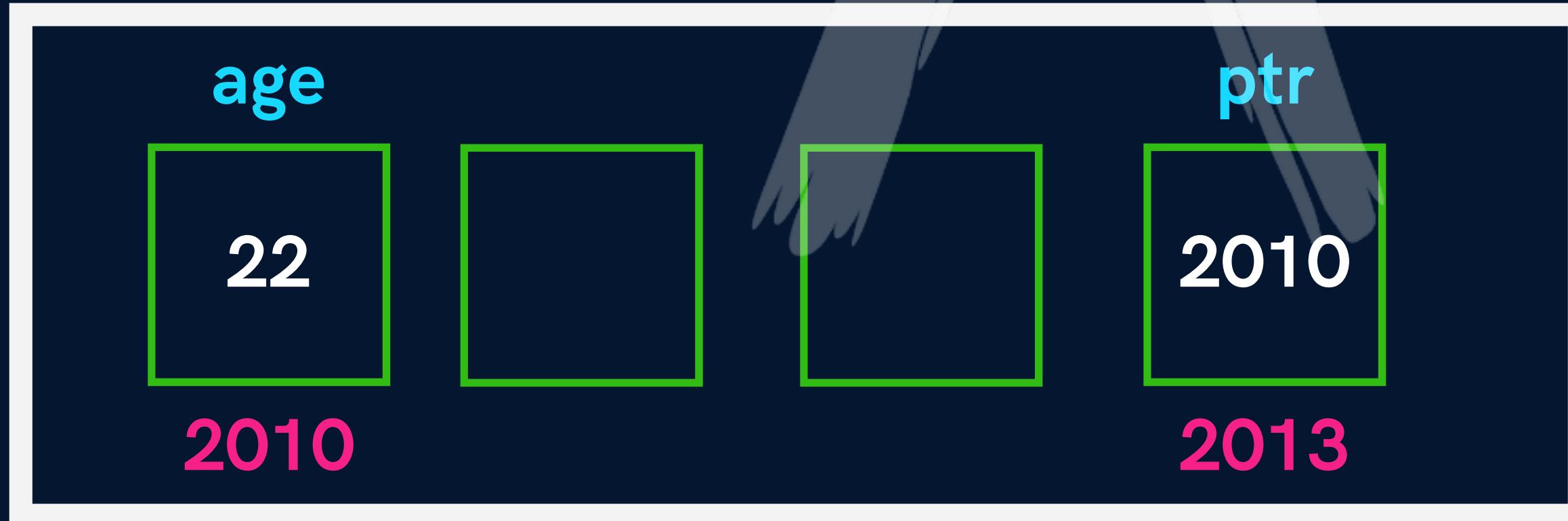
int factorial(int n) {
    if(n == 0) {
        return 1;
    }
    int factnml = factorial(n-1);
    int factn = factnml * n;
    return factn;
}
```

Pointers



A variable that stores the memory
address of another variable

Memory

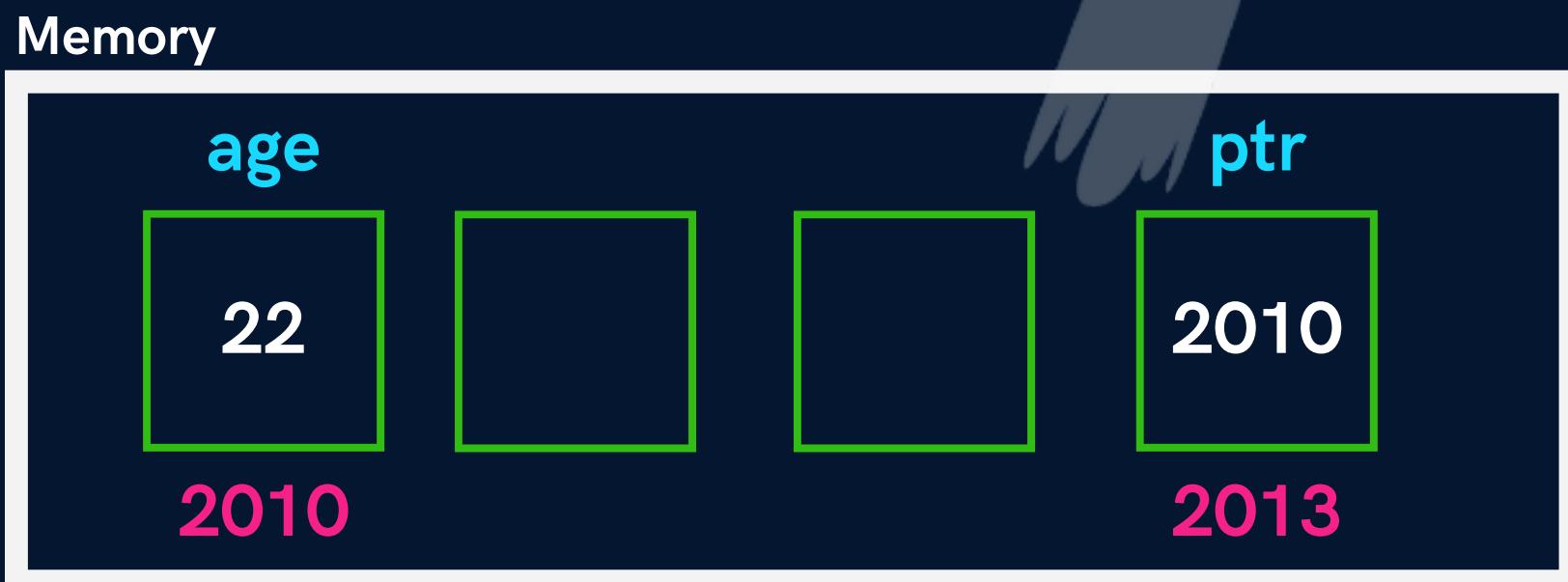


Syntax

```
int age = 22;
```

```
int *ptr = &age;
```

```
int _age = *ptr;
```



Declaring Pointers

```
int *ptr;
```

```
char *ptr;
```

```
float *ptr;
```



Format Specifier

```
printf("%p", &age);
```

```
printf("%p", ptr);
```

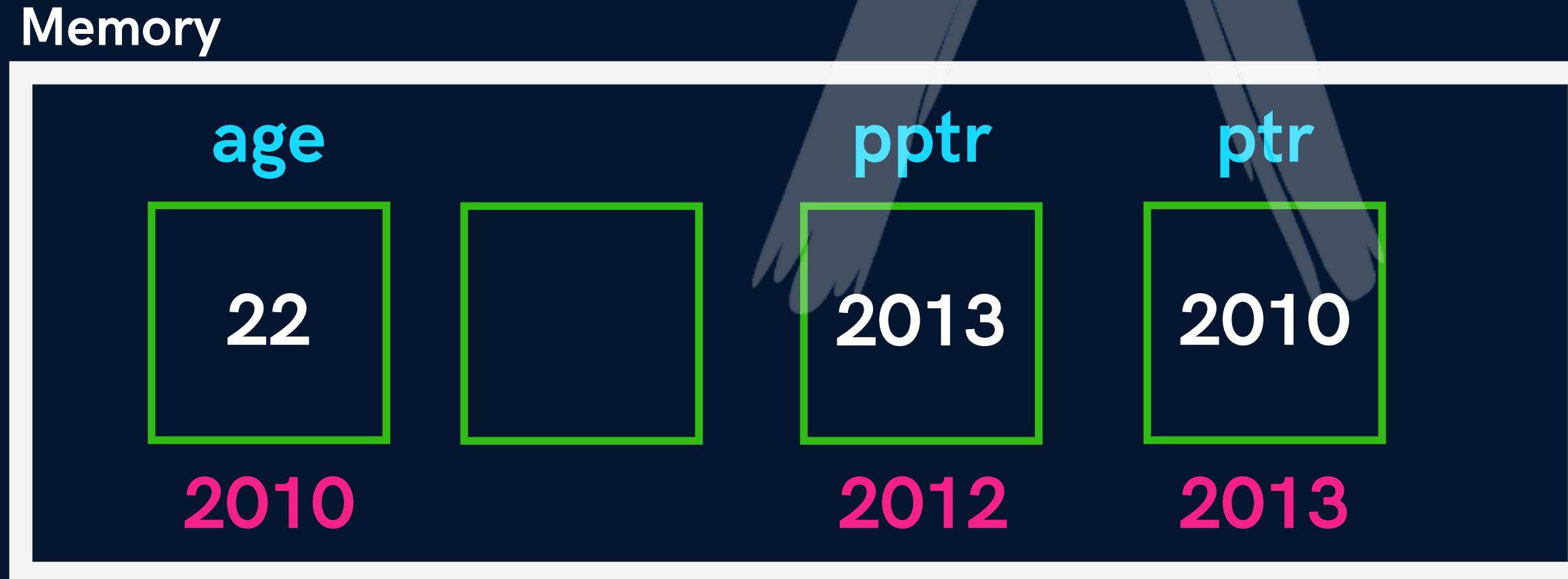
```
printf("%p", &ptr);
```



Pointer to Pointer



A variable that stores the memory
address of another **pointer**



Pointer to Pointer

Syntax

```
int **pptr;
```

```
char **pptr;
```

```
float **pptr;
```



Pointers in Function Call

**Call by
Value**

We pass value of
variable as
argument

**call by
Reference**

We pass address of
variable as
argument



C Language Tutorial

(Basic to Advanced)

Topics to be covered :

Installation + Setup
Chapter 1 - Variables, Data types + Input/Output
Chapter 2 - Instructions & Operators
Chapter 3 - Conditional Statements
Chapter 4 - Loop Control Statements
Chapter 5 - Functions & Recursion
Chapter 6 - Pointers
Chapter 7 - Arrays
Chapter 8 - Strings
Chapter 9 - Structures
Chapter 10 - File I/O
Chapter 11 - Dynamic Memory Allocation

Pointers

(Chapter 6)

1. Syntax

```
#include<stdio.h>

int main() {
    int age = 22;
    int *ptr = &age;
    int _age = *ptr;
    printf("%d\n", _age);

    //address
    printf("%p\n", &age);
    printf("%p\n", ptr);
    printf("%p\n", &ptr);

    //data
    printf("%d\n", age);
    printf("%d\n", *ptr);
    printf("%d\n", *(&age));
    return 0;
}
```

```
}
```

2. Pointers in Function call

```
# include <stdio.h>

void square(int n);
void _square(int* n);

int main() {
    int number = 4;

    //call by value
    square(number);
    printf("n is : %d\n", number);

    //call by reference
    _square(&number);
    printf("n is : %d\n", number);
    return 0;
}

void square(int n) {
    n = n * n;
    printf("square is : %d\n", n);
}

void _square(int* n) {
    *n = *n * *n;
    printf("square is : %d\n", *n);
}
```

3. Swap 2 numbers

```
# include <stdio.h>

void swap(int a, int b);
void _swap(int* a, int *b);

int main() {
    int x = 3, y = 5;

    //call by value
    swap(x, y);
```

```
printf("x = %d & y = %d\n", x, y);

//call by reference
_swap(&x, &y);
printf("x = %d & y = %d\n", x, y);
return 0;
}

void swap(int a, int b) {
    int t = a;
    a = b;
    b = a;
}

void _swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = *a;
}
```



Arrays



Collection of similar data types stored at contiguous memory locations

Syntax

```
int marks[3];
```

```
char name[10];
```

```
float price[2];
```



Input & Output

```
scanf("%d", &marks[0]);
```

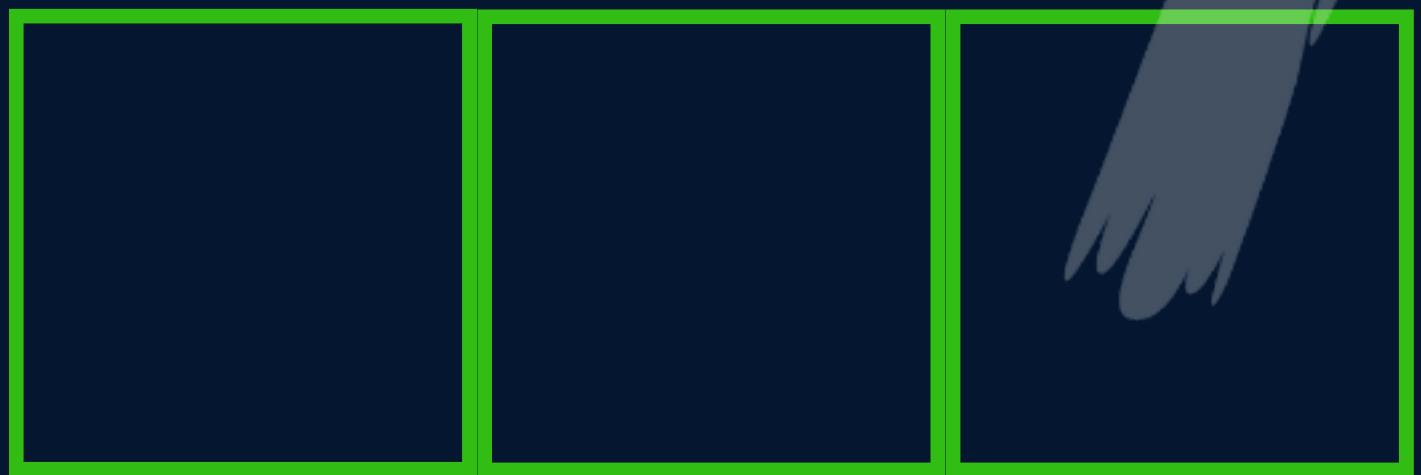
```
printf("%d", marks[0]);
```



Initialization of Array

```
int marks[ ] = {97, 98, 89};
```

```
int marks[ 3 ] = {97, 98, 89};
```



Memory Reserved :

Pointer Arithmetic

Pointer can be incremented
& decremented

CASE 1

```
int age = 22;  
int *ptr = &age;  
ptr++;
```

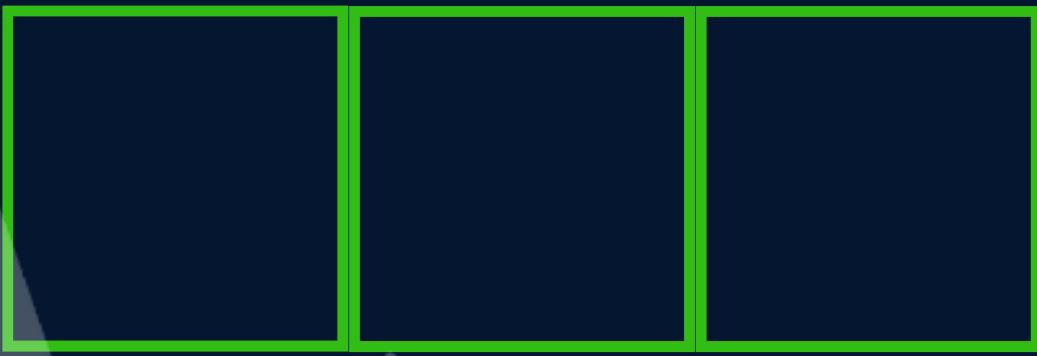
Pointer Arithmetic

CASE 2

```
float price = 20.00;  
float *ptr = &price;  
ptr++;
```

CASE 3

```
char star = '*';  
char *ptr = &star;  
ptr++;
```



Pointer Arithmetic

- We can also subtract one pointer from another
- We can also compare 2 pointers

Array is a Pointer

```
int *ptr = &arr[0];
```

```
int *ptr = arr;
```



Traverse an Array

```
int aadhar[10];
```

```
int *ptr = &aadhar[0];
```



Arrays as Function Argument

//Function Declaration

void **printNumbers** (int arr[], int n)

OR

void **printNumbers** (int *arr, int n)

//Function Call

printNumbers(arr, n);

Multidimensional Arrays

2 D Arrays

```
int arr[ ][ ] = { {1, 2}, {3, 4} }; //Declare
```

//Access

arr[0][0]

arr[0][1]

arr[1][0]

arr[1][1]

C Language Tutorial

(Basic to Advanced)

Topics to be covered :

Installation + Setup

Chapter 1 - Variables, Data types + Input/Output

Chapter 2 - Instructions & Operators

Chapter 3 - Conditional Statements

Chapter 4 - Loop Control Statements

Chapter 5 - Functions & Recursion

Chapter 6 - Pointers

Chapter 7 - Arrays

Chapter 8 - Strings

Chapter 9 - Structures

Chapter 10 - File I/O

Chapter 11 - Dynamic Memory Allocation

Arrays

(Chapter 7)

1. Syntax

```
# include <stdio.h>

int main() {
    int marks[3];
    printf("physics : ");
    scanf("%d", &marks[0]);

    printf("chem : ");
    scanf("%d", &marks[1]);

    printf("math : ");
    scanf("%d", &marks[2]);

    printf("physics = %d, ", marks[0]); //physics
    printf("chem = %d, ", marks[1]); //chem
    printf("math = %d \n", marks[2]); //math
```

```
    return 0;  
}
```

2. Pointer Arithmetic

```
# include <stdio.h>  
  
int main() {  
  
    int age = 22;  
    int *ptr = &age;  
  
    int _age = 25;  
    int *_ptr = &_age;  
  
    printf("%u\n", ptr);  
    ptr++;  
    printf("%u\n", ptr);  
    ptr--;  
    printf("%u\n", ptr);  
    ptr = ptr - _ptr;  
    printf("%u\n", ptr);  
  
    ptr = &_age;  
    printf("%d\n", ptr == _ptr);  
  
    return 0;  
}
```

3. Accessing an Array

```
# include <stdio.h>  
  
void printNumbers(int *arr, int n);  
void _printNumbers(int arr[], int n);  
  
int main() {  
    int arr[] = {1, 2, 3, 4, 5, 6};  
    printNumbers(arr, 6);  
    _printNumbers(arr, 6);  
    return 0;  
}
```

```
void printNumbers(int *arr, int n) {  
    for(int i=0; i<n; i++) {  
        printf("%d : %d\n", i, arr[i]);  
    }  
}  
  
void _printNumbers(int arr[], int n) {  
    for(int i=0; i<n; i++) {  
        printf("%d : %d\n", i, arr[i]);  
    }  
}
```

Strings



A character array terminated by a '\0' (null character)

null character denotes string termination

EXAMPLE

```
char name[ ] = {'S', 'H', 'R', 'A', 'D', 'H', 'A', '\0'};
```

```
char class[ ] = {'A', 'P', 'N', 'A', ' ', 'C', 'O', 'L', 'L', 'E', 'G', 'E', '\0'};
```

Initialising Strings

```
char name[ ] = {'S', 'H', 'R', 'A', 'D', 'H', 'A', '\0'};
```

```
char name[ ] = "SHRADHA";
```

```
char class[ ] = {'A', 'P', 'N', 'A', ' ', 'C', 'O', 'L', 'L', 'E', 'G', 'E', '\0'};
```

```
char class[ ] = "APNA COLLEGE";
```

What Happens in Memory?

```
char name[ ] = {'S', 'H', 'R', 'A', 'D', 'H', 'A', '\0'};
```

```
char name[ ] = "SHRADHA";
```

name

S	H	R	A	D	H	A	\0
---	---	---	---	---	---	---	----

2000 2001 2002 2003 2004 2005 2006 2007

String Format Specifier



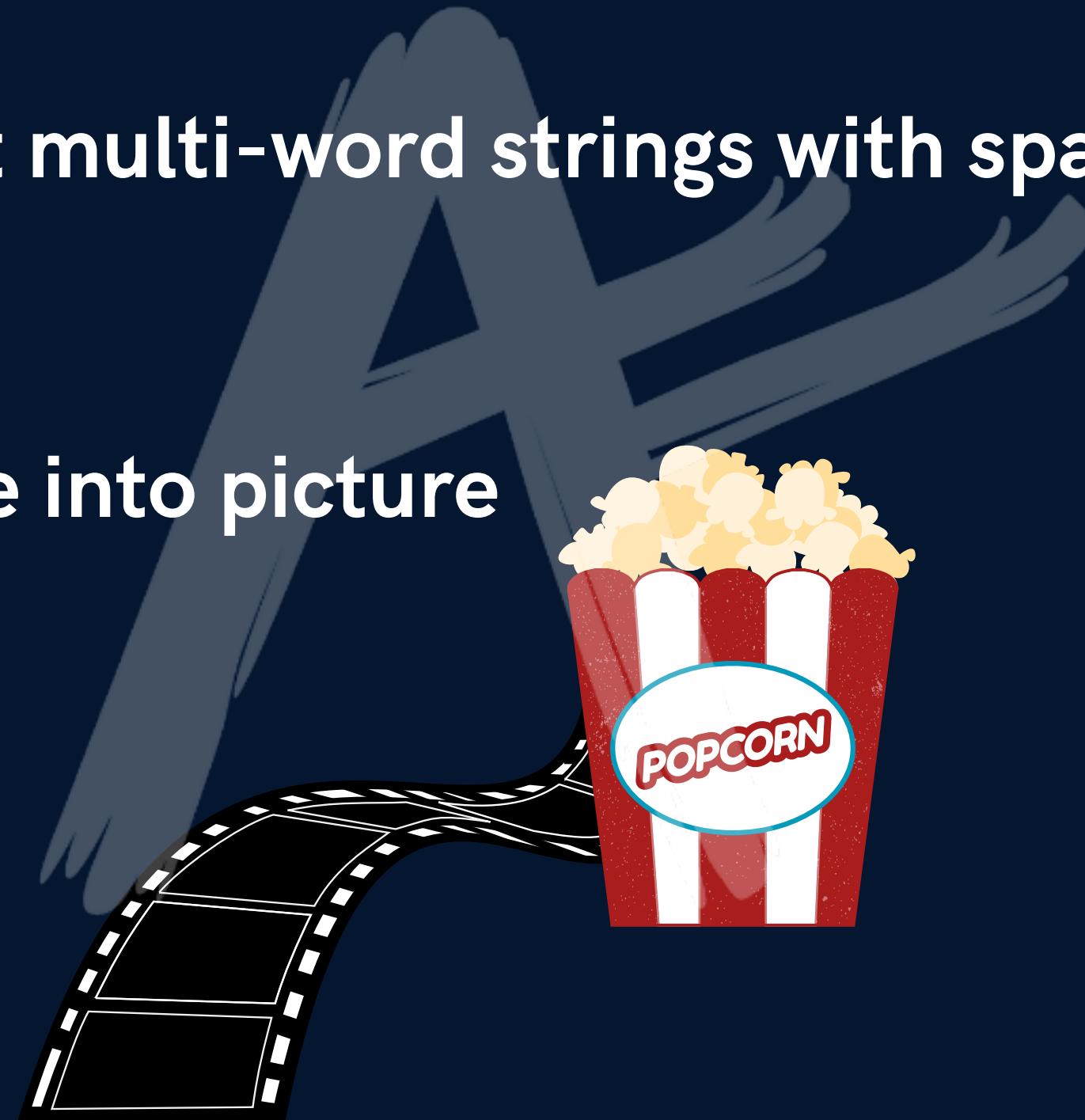
"%s"

```
char name[ ] = "Shradha";  
printf("%s", name);
```

IMPORTANT

`scanf()` **cannot** input multi-word strings with spaces

Here,
`gets()` & `puts()` come into picture



String Functions

gets(str) →

Dangerous &
Outdated

input a string
(even multiword)

puts(str)

output a string

fgets(str, n, file)

stops when $n-1$
chars input or new
line is entered

String using Pointers

```
char *str = "Hello World";
```

Store string in memory & the assigned address is stored in the char pointer 'str'

```
char *str = "Hello World"; //can be reinitialized
```

```
char str[ ] = "Hello World";  
//cannot be reinitialized
```

Standard Library Functions



<**string.h**>

1 **strlen(str)**

count number of characters excluding '\0'

Standard Library Functions



<**string.h**>

2 **strcpy(newStr, oldStr)**

copies value of old string to new string

Standard Library Functions



<**string.h**>

3 **strcat(firstStr, secStr)**

concatenates first string with second string

firstStr should be large
enough

Standard Library Functions



<**string.h**>

4 **strcpm(firstStr, secStr)**

Compares 2 strings & returns a value

0 -> **string equal**

positive -> **first > second (ASCII)**

negative -> **first < second (ASCII)**

C Language Tutorial

(Basic to Advanced)

Topics to be covered :

Installation + Setup

Chapter 1 - Variables, Data types + Input/Output

Chapter 2 - Instructions & Operators

Chapter 3 - Conditional Statements

Chapter 4 - Loop Control Statements

Chapter 5 - Functions & Recursion

Chapter 6 - Pointers

Chapter 7 - Arrays

Chapter 8 - Strings

Chapter 9 - Structures

Chapter 10 - File I/O

Chapter 11 - Dynamic Memory Allocation

Strings

(Chapter 8)

1. Strings

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

int main() {
    //declaration
    char name[] = "Shradha Khapra";
    char course[] = {'a','p', 'n', 'a', ' ', 'c', 'o', 'l', 'e', 'g', 'e',
'\'0'};

    //printing string
    for(int i=0; name[i] != '\0'; i++) {
        printf("%c", name[i]);
    }
    printf("\n");

    //printing string with pointer
    for(char *ptr=name; *ptr != '\0'; ptr++) {
        printf("%c", *ptr);
    }
}
```

```
}

printf("\n");

//printing using format specifier
printf("%s\n", name);

//input a string
char firstName[40];
printf("enter first name : ");
scanf("%s", firstName);
printf("you first name is %s\n", firstName);

char fullName[40];
printf("enter full name : ");
scanf("%s", fullName);
printf("you first name is %s\n", fullName);

// gets & puts
char fullName[40];
printf("enter full name : ");
fgets(fullName, 40, stdin);
puts(fullName);

//Library Functions
char name[] = "Shradha";
int length = strlen(name);
printf("the length of name : %d\n", length);

char oldVal[] = "oldValue";
char newVal[50];
strcpy(newVal, oldVal);
puts(newVal);

char firstStr[50] = "Hello ";
char secStr[] = "World";
strcat(firstStr, secStr);
puts(firstStr);

char str1[] = "Apple";
char str2[] = "Banana";
printf("%d\n", strcmp(str1, str2));

//enter String using %c
```

```
printf("enter string : ");
char str[100];
char ch;
int i = 0;

while(ch != '\n') {
    scanf("%c", &ch);
    str[i] = ch;
    i++;
}
str[i] = '\0';
puts(str);

return 0;
}
```

> Some more Qs

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

// void printString(char arr[]);
// int countLength(char arr[]);
// void salting(char password[]);
// void slice(char str[], int n, int m);

//int countVowels(char str[]);

void checkChar(char str[], char ch);

int main() {
    char str[] = "ApnaCollege";
    char ch = 'x';
    checkChar(str, ch);
}

void checkChar(char str[], char ch) {
    for(int i=0; str[i] != '\0'; i++) {
        if(str[i] == ch) {
            printf("character is present!");
            return;
        }
    }
}
```

```
        }
    }
    printf("character is NOT present:(");
}

// int countVowels(char str[]) {
//     int count = 0;

//     for(int i=0; str[i] != '\0'; i++) {
//         if(str[i] == 'a' || str[i] == 'e' || str[i] == 'i' ||
//             str[i] == 'o' || str[i] == 'u') {
//             count++;
//         }
//     }
//     return count;
// }

// void slice(char str[], int n, int m) { // n & m are valid value
//     char newStr[100];
//     int j = 0;
//     for(int i=n; i<=m; i++, j++) {
//         newStr[j] = str[i];
//     }
//     newStr[j] = '\0';
//     puts(newStr);
// }

// void salting(char password[]) {
//     char salt[] = "123";
//     char newPass[200];

//     strcpy(newPass, password); // newPass = "test"
//     strcat(newPass, salt); // newPass = "test" + "123";
//     puts(newPass);
// }
```

```
// }

// int countLength(char arr[]) {
//     int count = 0;
//     for(int i=0; arr[i]!='\0'; i++) {
//         count++;
//     }
//     return count-1;
// }

// void printString(char arr[]) {
//     for(int i=0; arr[i] != '\0' ;i++) {
//         printf("%c", arr[i]);
//     }
//     printf("\n");
// }
```

Structures



a collection of values of **different** data types

EXAMPLE

For a student store the following :

name (String)

roll no (Integer)

cgpa (Float)

Syntax

```
struct student {  
    char name[100];  
  
    int roll;  
  
    float cgpa;  
};
```

```
struct student s1;  
s1.cgpa = 7.5;
```

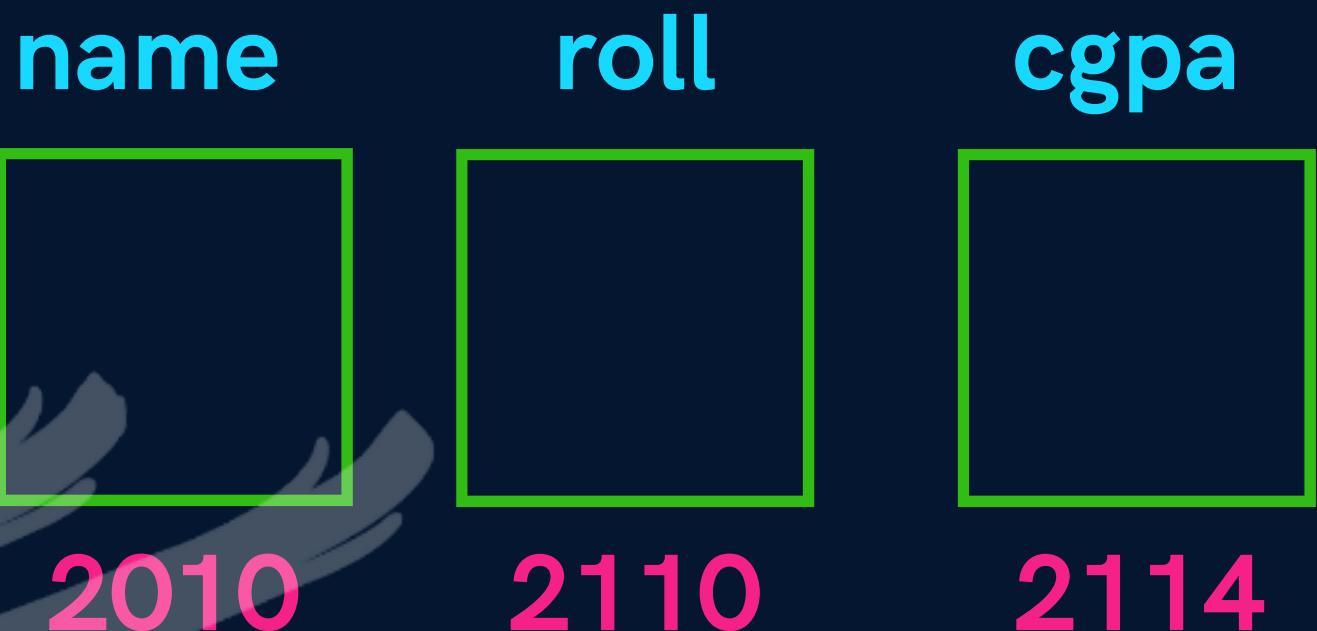
Syntax

```
struct student {  
    char name[100];  
    int roll;  
    float cgpa;  
}
```



Structures in Memory

```
struct student {  
    char name[100];  
  
    int roll;  
  
    float cgpa;  
}
```



structures are stored in contiguous memory locations

Benefits of using Structures

- Saves us from creating too many variables
- Good data management/organization

Array of Structures

```
struct student ECE[100];
```

```
struct student COE[100];
```

```
struct student IT[100];
```

ACCESS

```
IT[0].roll = 200;
```

```
IT[0].cgpa = 7.6;
```



Initializing Structures

```
struct student s1 = { "shradha", 1664, 7.9};
```

```
struct student s2 = { "rajat", 1552, 8.3};
```

```
struct student s3 = { 0 };
```

Pointers to Structures

```
struct student s1;
```

```
struct student *ptr;
```

```
ptr = &s1;
```



Arrow Operator

`(*ptr).code` \longleftrightarrow `ptr->code`

Passing structure to function

```
//Function Prototype  
void printInfo(struct student s1);
```

typedef Keyword



used to create **alias** for data types

```
typedef struct ComputerEngineeringStudent{  
    int roll;  
    float cgpa;  
    char name[100];  
} coe;
```

coe student1;

C Language Tutorial

(Basic to Advanced)

Topics to be covered :

Installation + Setup
Chapter 1 - Variables, Data types + Input/Output
Chapter 2 - Instructions & Operators
Chapter 3 - Conditional Statements
Chapter 4 - Loop Control Statements
Chapter 5 - Functions & Recursion
Chapter 6 - Pointers
Chapter 7 - Arrays
Chapter 8 - Strings
Chapter 9 - Structures
Chapter 10 - File I/O
Chapter 11 - Dynamic Memory Allocation

Structures

(Chapter 9)

1. Structures

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

struct student {
    char name[100];
    int roll;
    float cgpa;
};

typedef struct ComputerEngineeringStudent{
    int roll;
    float cgpa;
    char name[100];
} coe;

void printInfo(struct student s1);

int main() {
```

```
struct student s1;
// s1.name = "Shradha"; // not a modifiable value
strcpy(s1.name,"Shradha");
s1.roll = 64;
s1.cgpa = 9.2;

printf("student info : \n");
printf("name = %s\n", s1.name);
printf("roll no = %d\n", s1.roll);
printf("cgpa = %f\n", s1.cgpa);

//array of structures
struct student IT[60];
struct student COE[60];
struct student ECE[60];

//declaration
struct student s2 = {"Rajat", 1552, 8.6};
struct student s3 = {0};

printf("roll no of s2 = %d\n", s2.roll);
printf("roll no of s3 = %d\n", s3.roll);

//pointer to structure
struct student *ptr = &s1;
printf("student.name = %s\n", (*ptr).name);
printf("student.roll = %d\n", (*ptr).roll);
printf("student.cgpa = %f\n", (*ptr).cgpa);

//arrow operator
printf("student->name = %s\n", ptr->name);
printf("student->roll = %d\n", ptr->roll);
printf("student->cgpa = %f\n", ptr->cgpa);

//Passing structure to function
printInfo(s1);

//typedef keyword
coe student1;
student1.roll = 1664;
student1.cgpa = 6.7;
```

```
    strcpy(student1.name, "sudhir");

    return 0;
}

void printInfo(struct student s1) {
    printf("student info : \n");
    printf("name = %s\n", s1.name);
    printf("roll no = %d\n", s1.roll);
    printf("cgpa = %f\n", s1.cgpa);

    //change
    s1.roll = 1660; //but it won't be reflected to the main function
                    //as structures are passed by value
}
```

> Some more Qs

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

//user defined
typedef struct student {
    int roll;
    float cgpa;
    char name[100];
} stu ;

typedef struct computerengineeringstudent {
    int roll;
    float cgpa;
    char name[100];
} coe;

struct address {
    int houseNo;
    int block;
    char city[100];
    char state[100];
};

struct vector {
    int x;
```

```
int y;
};

void calcSum(struct vector v1, struct vector v2, struct vector sum);

struct complex {
    int real;
    int img;
};

typedef struct BankAccount {
    int accountNo;
    char name[100];
} acc ;

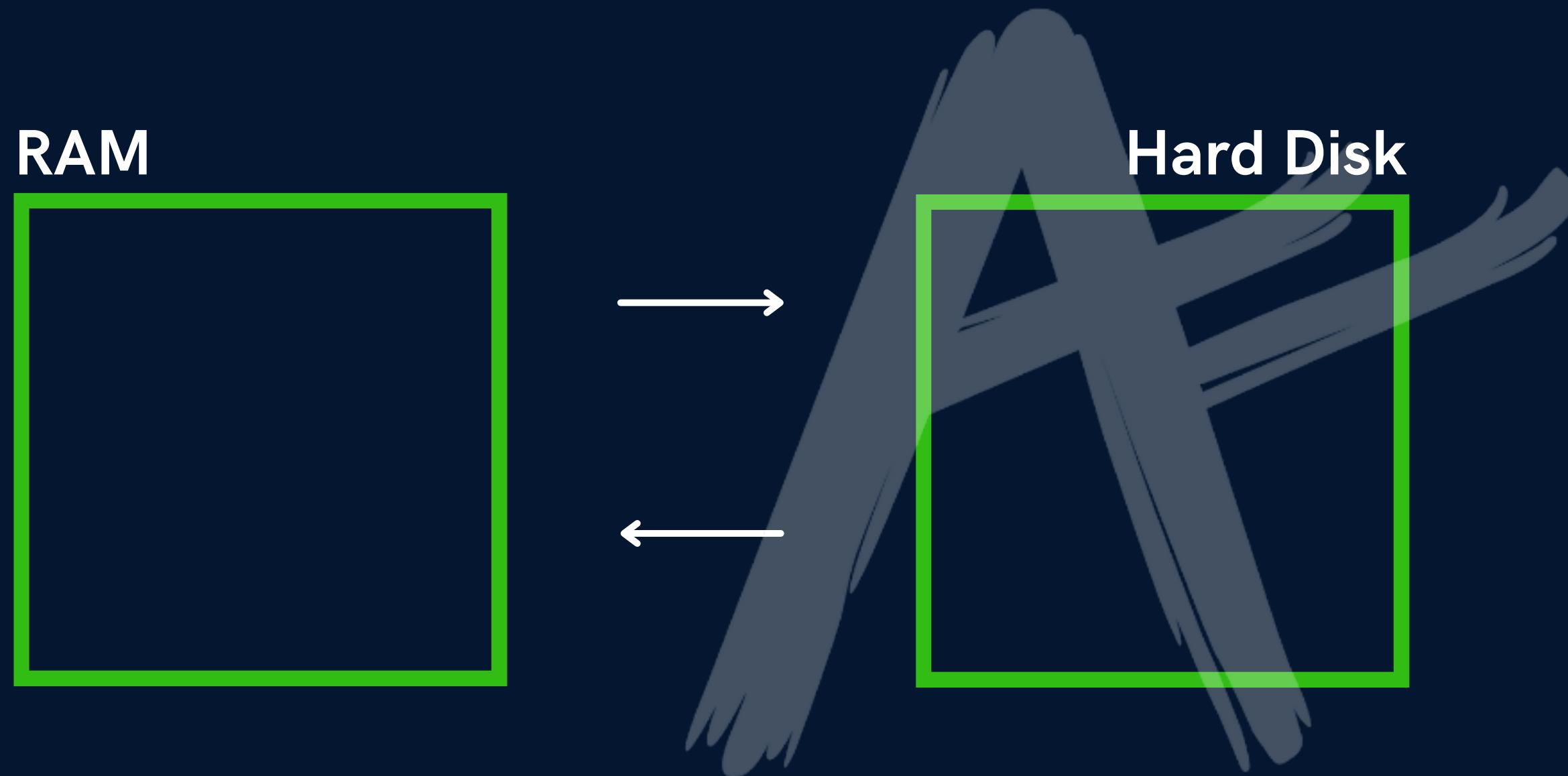
int main() {
    acc acc1 = {123, "shradha"};
    acc acc2 = {124, "rajat"};
    acc acc3 = {125, "sudhir"};

    printf("acc no = %d", acc1.accountNo);
    printf("name = %s", acc1.name);
    return 0;
}

void calcSum(struct vector v1, struct vector v2, struct vector sum) {
    sum.x = v1.x + v2.x;
    sum.y = v1.y + v2.y;

    printf("sum of x is : %d\n", sum.x);
    printf("sum of y is : %d\n", sum.y);
}
```

File IO



File IO

FILE - container in a storage device to store data

- RAM is **volatile**
- Contents are lost when program terminates
- Files are used to persist the data

Operation on Files

Create a File

Open a File

Close a File

Read from a File

Write in a File



Types of Files

Text Files

textual data

.txt, .c

Binary Files

binary data

.exe, .mp3, .jpg

File Pointer

FILE is a (hidden)structure that needs to be created for opening a file

A **FILE ptr** that points to this structure & is used to access the file.

```
FILE *fptr;
```

Opening a File

```
FILE *fptr;
```

```
fptr = fopen("filename", mode);
```

Closing a File

```
fclose(fptr);
```

File Opening Modes

"r" open to read

"rb" open to read in binary

"w" open to write

"wb" open to write in binary

"a" open to append



BEST Practice

Check if a file exists before reading from it.



Reading from a file

```
char ch;
```

```
fscanf(fp, "%c", &ch);
```

Writing to a file

```
char ch = 'A';
```

```
fprintf(fptr, "%c", ch);
```



Read & Write a char

fgetc(fpstr)

fputc('A', fpstr)

EOF (End Of File)

`fgetc` returns **EOF** to show that the file has ended

C Language Tutorial

(Basic to Advanced)

Topics to be covered :

Installation + Setup

Chapter 1 - Variables, Data types + Input/Output

Chapter 2 - Instructions & Operators

Chapter 3 - Conditional Statements

Chapter 4 - Loop Control Statements

Chapter 5 - Functions & Recursion

Chapter 6 - Pointers

Chapter 7 - Arrays

Chapter 8 - Strings

Chapter 9 - Structures

Chapter 10 - File I/O

Chapter 11 - Dynamic Memory Allocation

File I/O

(Chapter 10)

```
# include <stdio.h>

int main() {
    FILE *fptr;
    //Reading a file
    char ch;
    fptr = fopen("Test.txt", "r");
    if(fptr == NULL) {
        printf("doesn't exist!\n");
    } else {
        fscanf(fptr, "%c", &ch);
        printf("character in file is : %c\n", ch);
        fscanf(fptr, "%c", &ch);
        printf("character in file is : %c\n", ch);
        fclose(fptr);
    }

    //Writing in a file
    ch = 'M';
    fptr = fopen("NewFile.txt", "w");
}
```

```
fprintf(fptr, "%cANGO", ch);
fclose(fptr);

//fgets
fptr = fopen("NewFile.txt", "r");
printf("character in file is : %c\n", fgetc(fptr));
fclose(fptr);

//fputc
fptr = fopen("NewFile.txt", "w");
fputc('a', fptr);
fputc('p', fptr);
fputc('p', fptr);
fputc('l', fptr);
fputc('e', fptr);
fclose(fptr);

//read the full file (EOF)
fptr = fopen("NewFile.txt", "r");
ch = fgetc(fptr);
while(ch != EOF) {
    printf("%c", ch);
    ch = fgetc(fptr);
}
printf("\n");
fclose(fptr);

return 0;
}
```

Dynamic Memory Allocation



It is a way to allocate memory to a data structure during the **runtime**.

We need some functions to allocate & free memory dynamically.

Functions for DMA

- a. malloc()
- b. calloc()
- c. free()
- d. realloc()

malloc()

memory allocation

takes number of **bytes** to be allocated
& returns a pointer of type **void**

```
ptr = (*int) malloc(5 * sizeof(int));
```

calloc()

continuous allocation

initializes with 0

```
ptr = (*int) calloc(5, sizeof(int));
```

free()

We use it to free memory that is allocated
using malloc & calloc

free(ptr);

realloc()

reallocate (increase or decrease) memory
using the same pointer & size.

`ptr = realloc(ptr, newSize);`

C Language Tutorial (Basic to Advanced)

Topics to be covered :

Installation + Setup
Chapter 1 - Variables, Data types + Input/Output
Chapter 2 - Instructions & Operators
Chapter 3 - Conditional Statements
Chapter 4 - Loop Control Statements
Chapter 5 - Functions & Recursion
Chapter 6 - Pointers
Chapter 7 - Arrays
Chapter 8 - Strings
Chapter 9 - Structures
Chapter 10 - File I/O
Chapter 11 - Dynamic Memory Allocation

Dynamic Memory Allocation (Chapter 11)

```
# include <stdio.h>
# include <stdlib.h>
//Dynamic Memory Allocation

int main() {
    //sizeof function
    printf("%d\n", sizeof(int));
    printf("%d\n", sizeof(float));
    printf("%d\n", sizeof(char));

    //malloc
    // int *ptr;
    // ptr = (int *) malloc(5 * sizeof(int));

    // for(int i=0; i<5; i++) {
    //     scanf("%d", &ptr[i]);
    // }

    // for(int i=0; i<5; i++) {
    //     printf("number %d = %d\n", i+1, ptr[i]);
    // }
```

```
// }

//calloc
int *ptr = (int *) calloc(5, sizeof(int));

for(int i=0; i<5; i++) {
    printf("number %d = %d\n", i+1, ptr[i]);
}

//free
free(ptr);

return 0;
}
```

1

HOMEWORK SET

a. Write a program to calculate perimeter of rectangle.

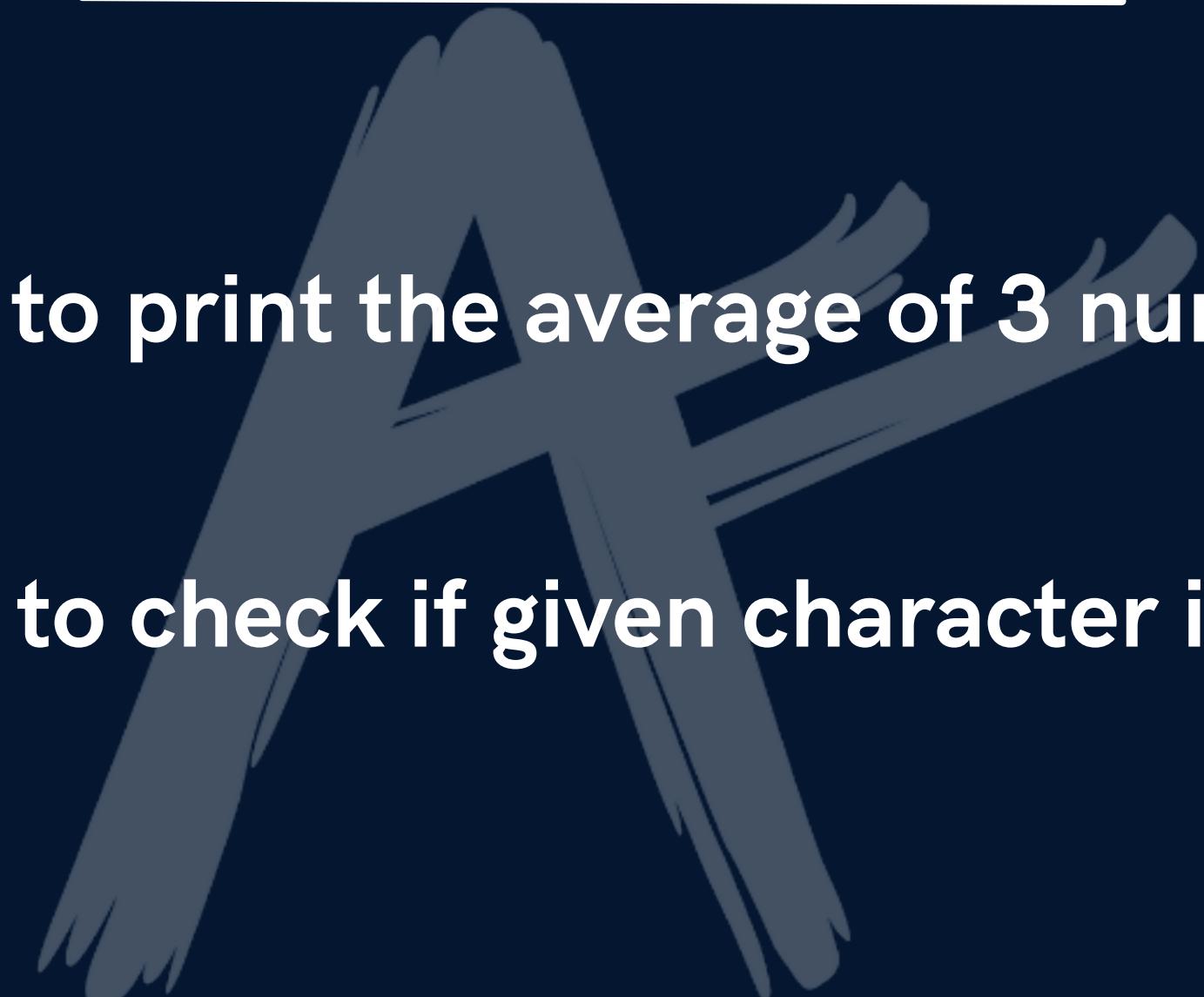
Take sides, a & b, from the user.

b. Take a number(n) from user & output its cube($n \times n \times n$).

c. Write comments for programs a & b.

2

HOMEWORK SET

- 
- a. Write a program to print the average of 3 numbers.
 - b. Write a program to check if given character is digit or not.
 - c. Write a program to print the smallest number of two.

3

HOMEWORK SET

a. Write a program to check if a given number is Armstrong number or not.

(Search what is Armstrong number)

b. Write a program to check if the given number is a natural number.

(Natural numbers start from 1)

4

HOMEWORK SET

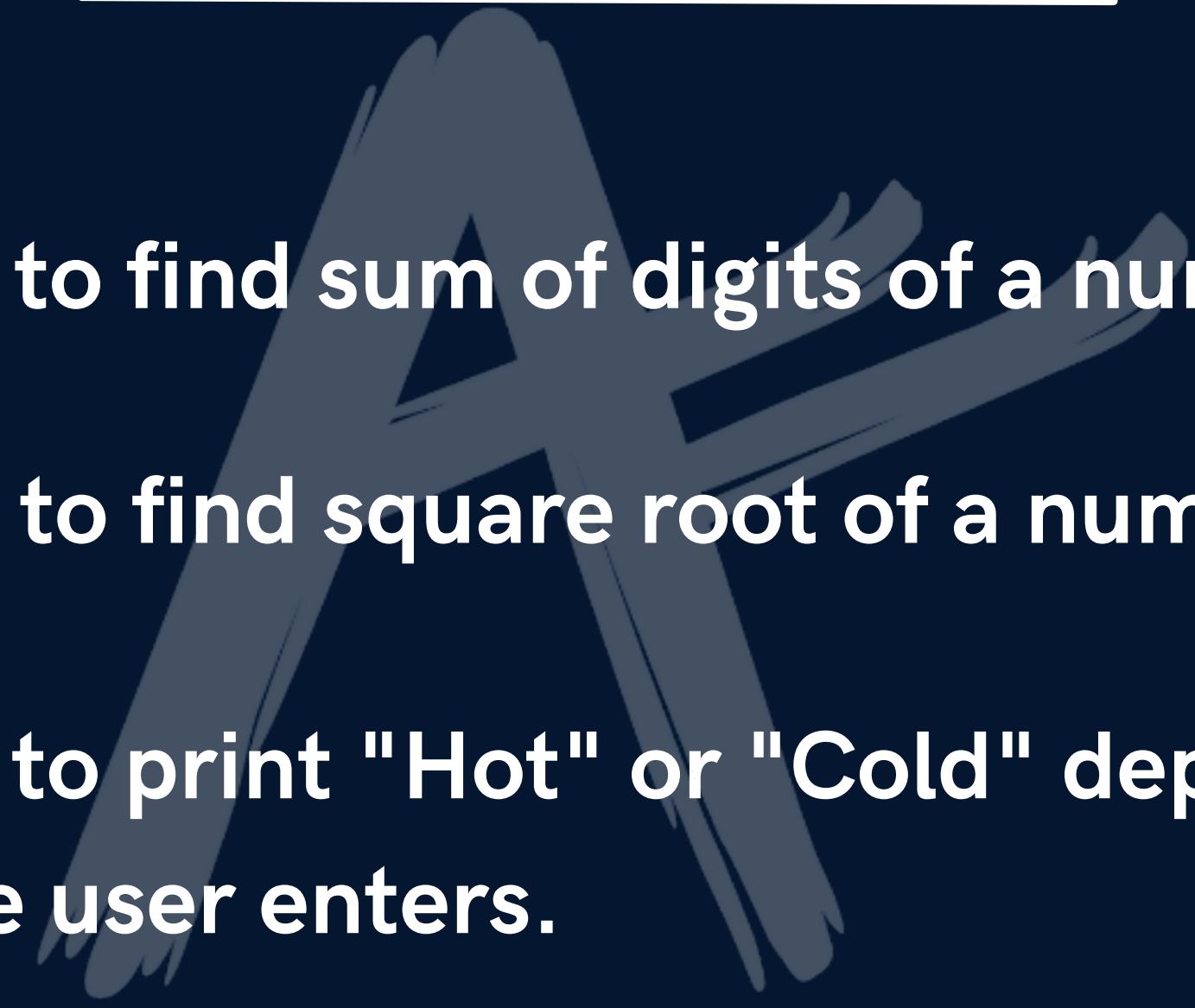
- a. Search on what a "nested loop" is & print this pattern using it.

```
* * * * *
* * * * *
* * * * *
* * * * *
```

- b. Write a program to check if a number is prime or not.
- c. Write a program to print prime numbers in a range.

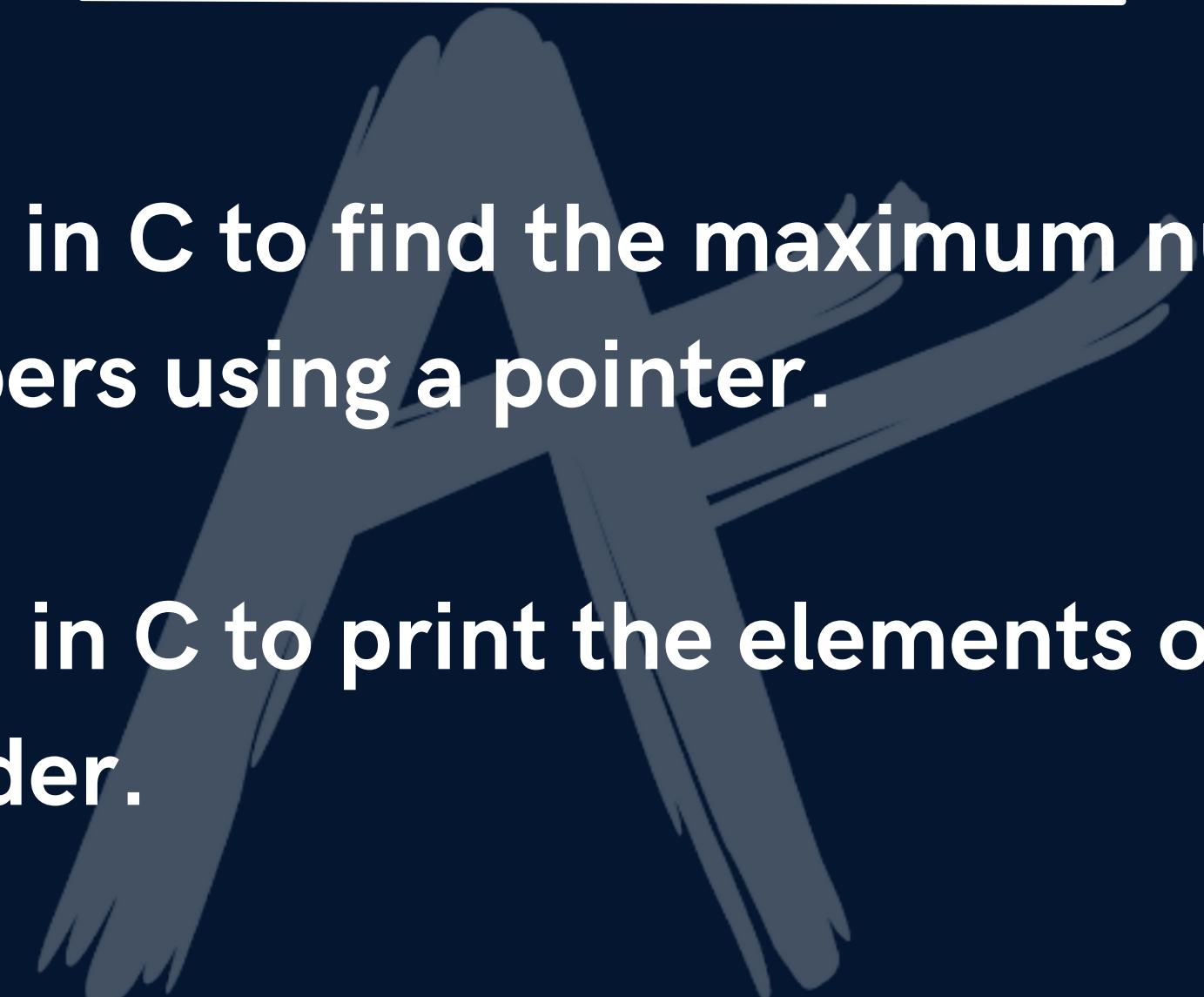
5

HOMEWORK SET

- 
- a. Write a function to find sum of digits of a number.
 - b. Write a function to find square root of a number.
 - c. Write a function to print "Hot" or "Cold" depending on the temperature user enters.
 - d. Make your own pow function.

6

HOMEWORK SET

- 
- a. Write a program in C to find the maximum number between two numbers using a pointer.
 - b. Write a program in C to print the elements of an array in reverse order.
 - c. Write a program in C to print all the letters in english alphabet using a pointer.

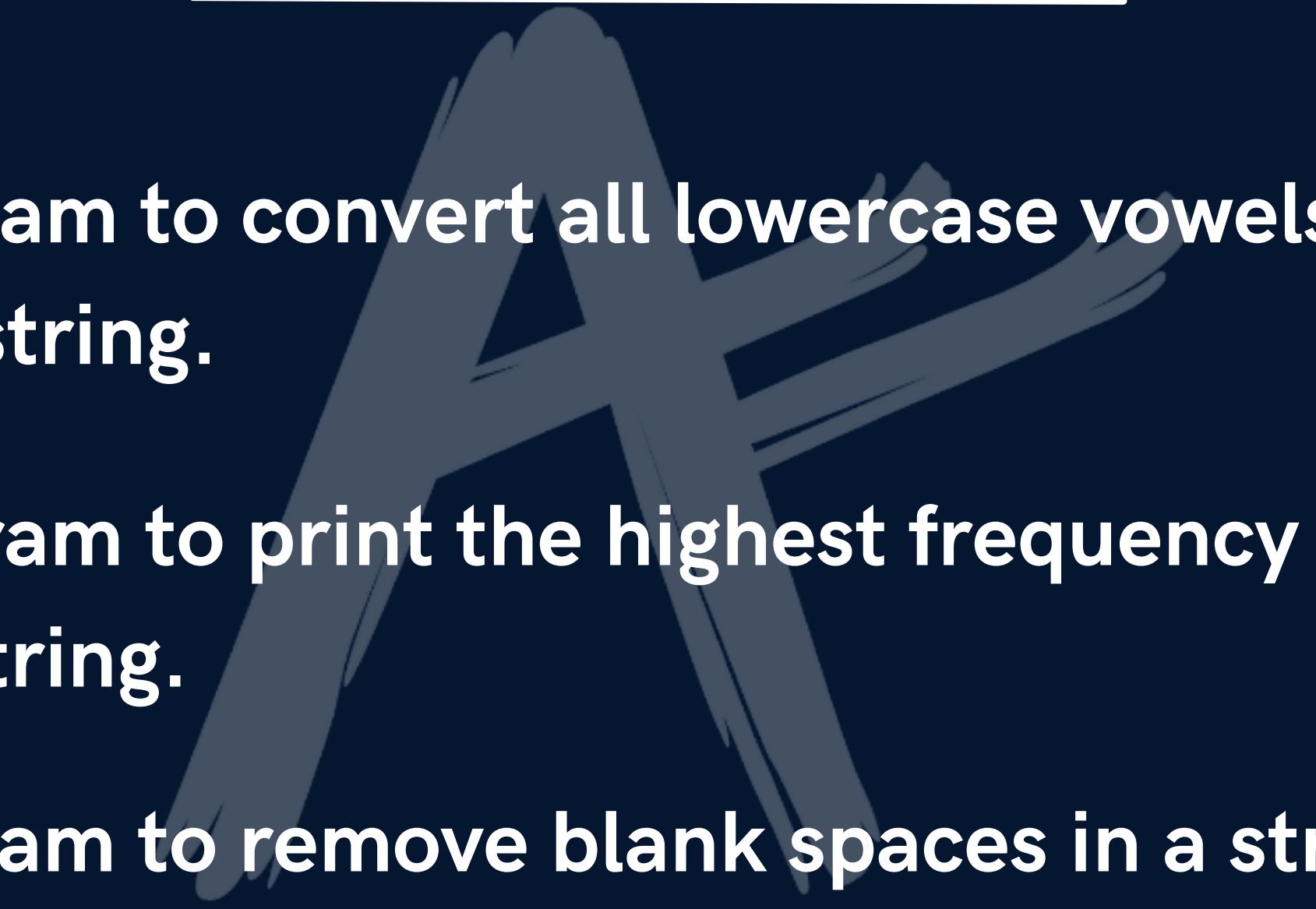
7

HOMEWORK SET

- 
- a. In an array of numbers, find how many times does a number 'x' occurs.
 - b. Write a program to print the largest number in an array.
 - c. Write a program to insert an element at the end of an array.

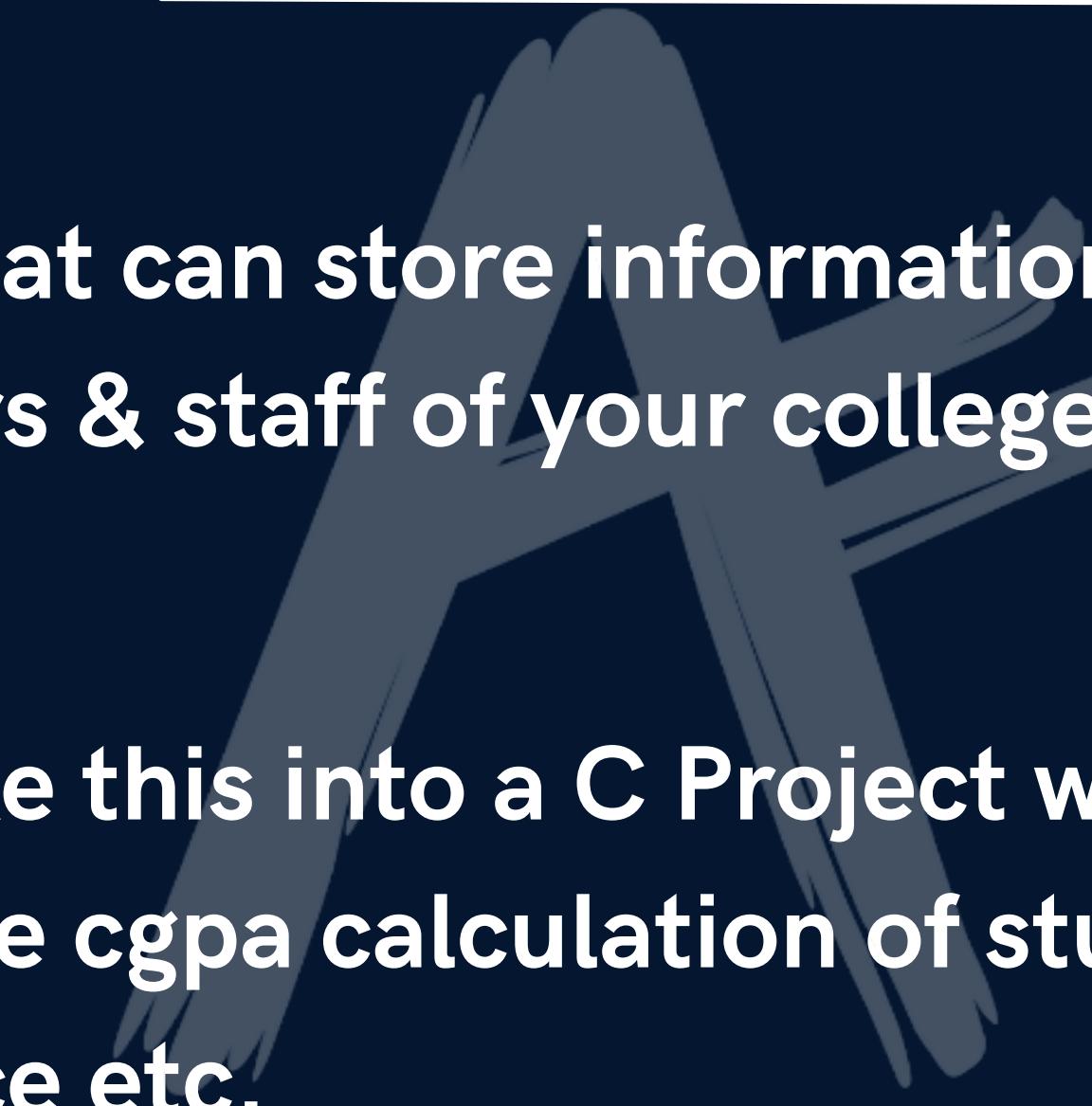
8

HOMEWORK SET

- 
- a. Write a program to convert all lowercase vowels to uppercase in a string.
 - b. Write a program to print the highest frequency character in a string.
 - c. Write a program to remove blank spaces in a string.
 - d. Write a program to replace lowercase letters with uppercase & vice versa in a string.

9

HOMEWORK SET

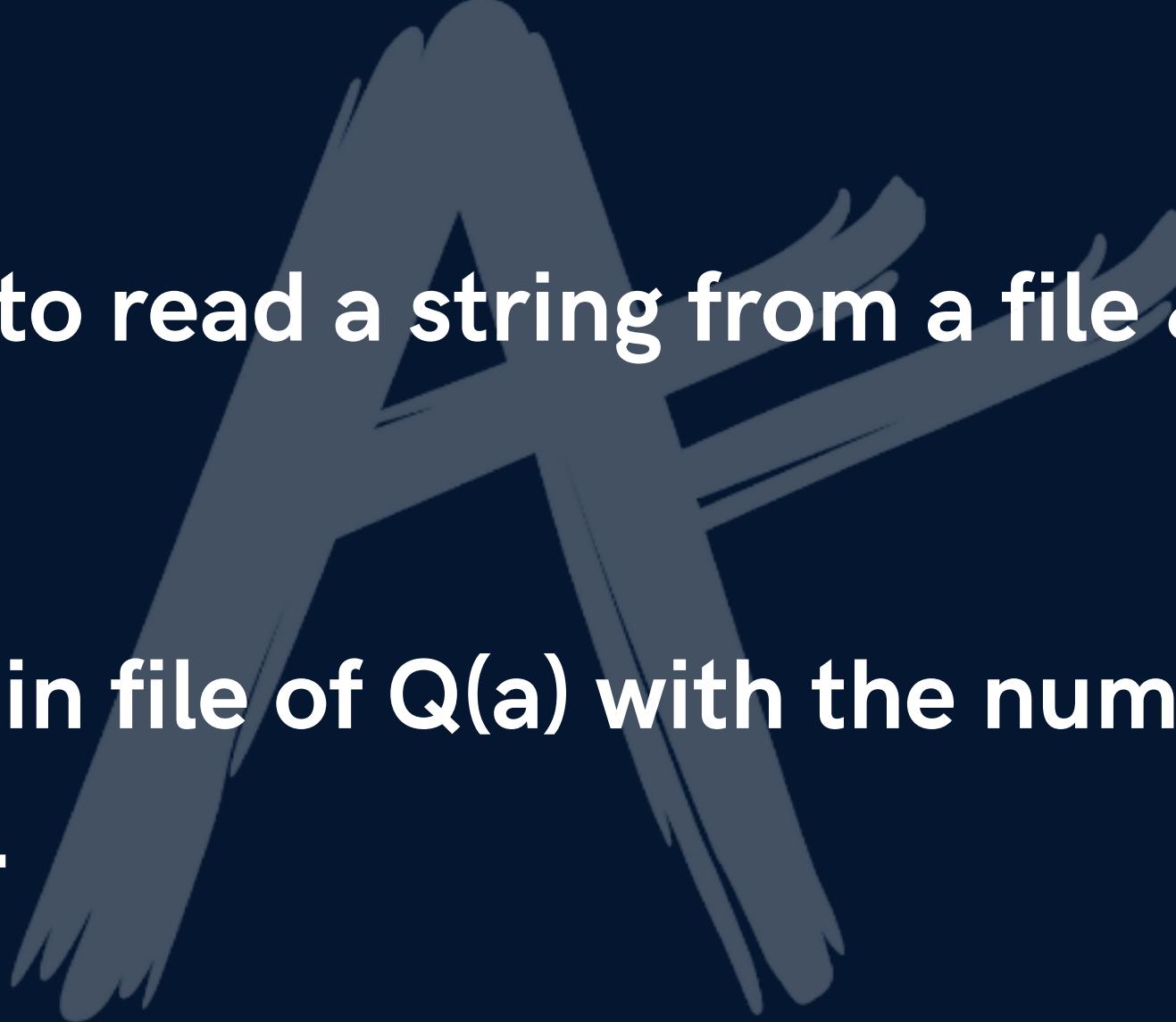


Make a system that can store information of all students, teachers & staff of your college in the form of structures.

You can also make this into a C Project with other functionalities like cgpa calculation of students, storing attendance etc.

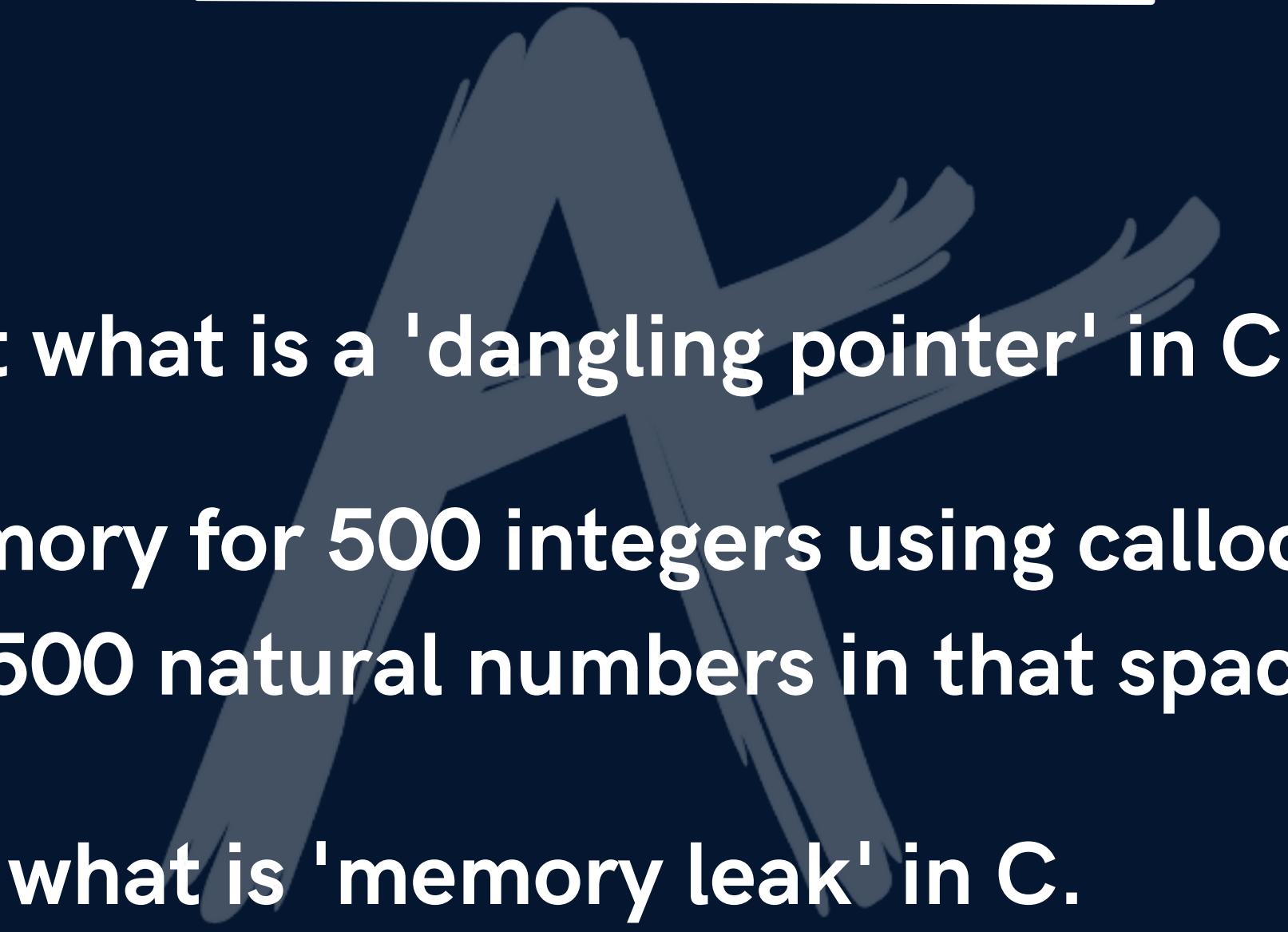
10

HOMEWORK SET

- 
- a. Write a program to read a string from a file & output to the user.
 - a. Replace the data in file of Q(a) with the number of vowels in the string.
 - c. Format the information of 5 students (name, marks, cgpa, course) in a table like structure in a file.

11

HOMEWORK SET

- 
- a. Search about what is a 'dangling pointer' in C.
 - b. Allocate memory for 500 integers using calloc & then store first 500 natural numbers in that space.
 - c. Search about what is 'memory leak' in C.
 - d. Search & find out which is better malloc() or calloc ().