



PROGRAMMING FOR EVERYONE

18TH | 19TH | 20TH SEP

A large, stylized graphic where the words 'PROGRAMMING FOR EVERYONE' and the dates '18TH | 19TH | 20TH SEP' are formed by various purple and pink rounded rectangles and circles of different sizes, creating a blocky, geometric look.

LOST
CHILDREN
WILL BE
TAUGHT
THE
C
PROGRAMMING
LANGUAGE

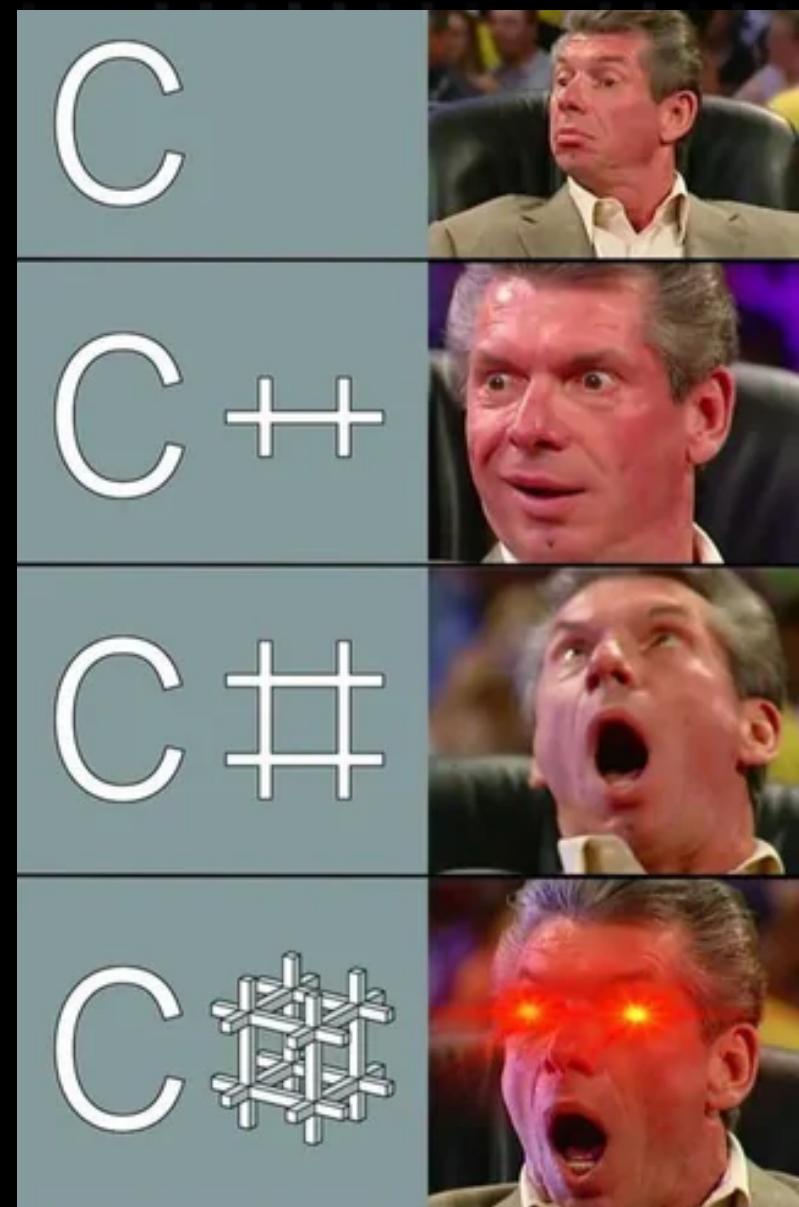
Introduction to C

C is a powerful, high-performance programming language that's widely used in system embeddings, software development, including game development, and applications requiring high performance.

Key features of C:

LOW-LEVEL MANIPULATION: IT ALLOWS FOR DIRECT MANIPULATION OF HARDWARE AND MEMORY, WHICH IS USEFUL FOR SYSTEM-LEVEL PROGRAMMING.

PERFORMANCE: C IS KNOWN FOR ITS PERFORMANCE AND EFFICIENCY, MAKING IT IDEAL FOR APPLICATIONS WHERE SPEED AND RESOURCE MANAGEMENT ARE CRITICAL.



Header Files in C

- Header files contain predefined functions and macros that can be used in programs.
- They save time by allowing reuse of common code.
- Always included at the top of a C program using #include.

Common Header Files:

- <stdio.h> → Standard Input/Output (e.g., printf, scanf).
- <string.h> → String handling (e.g., strcpy, strlen).
- <math.h> → Mathematical functions (e.g., sqrt, pow).
- <stdlib.h> → Utility functions (e.g., malloc, free, atoi).

```
#include <header_file_name>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
```

Execution

SOURCE → PREPROCESSOR → COMPILER → LINKER → EXECUTABLE

Stages of Execution in C:

Source File (.c)

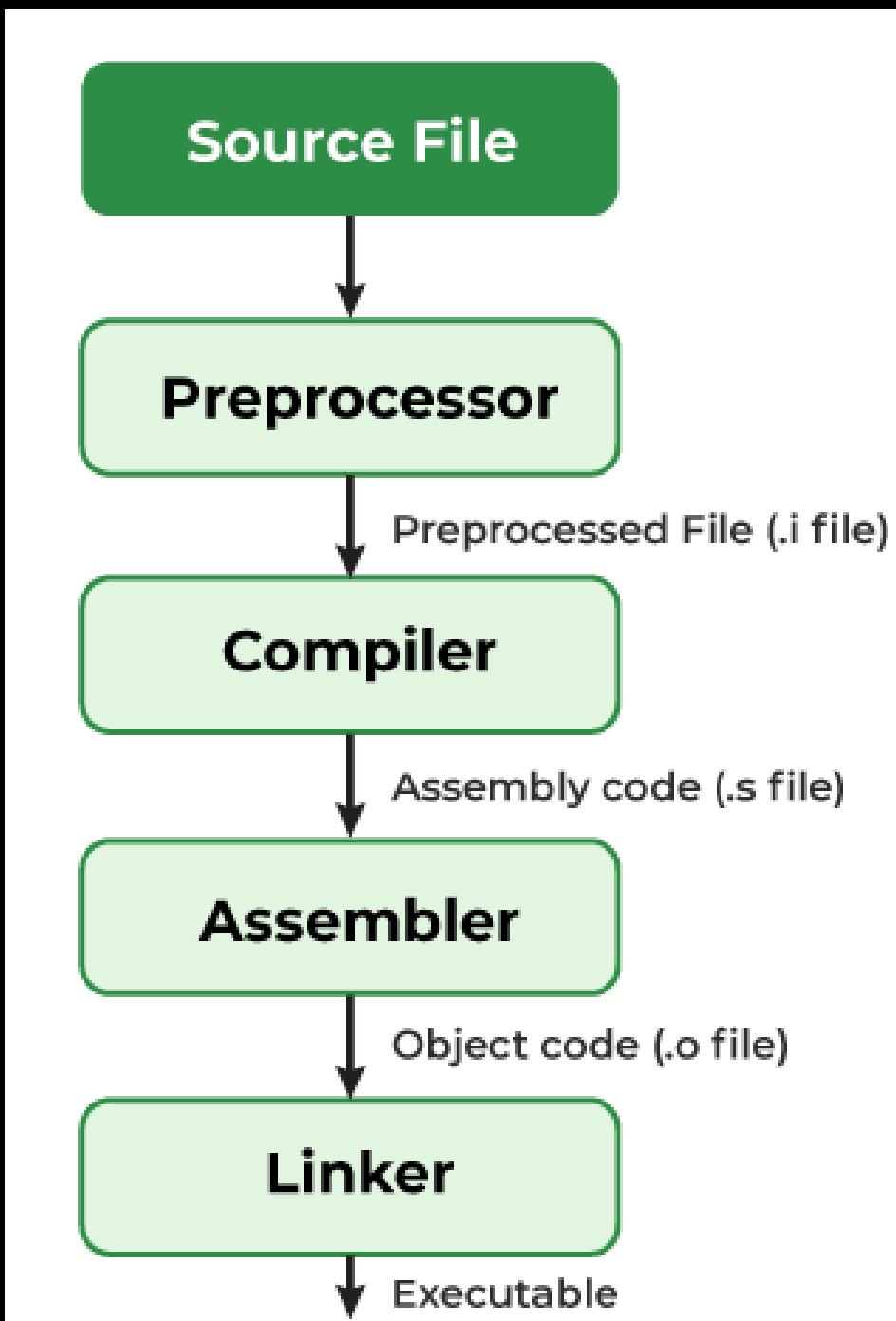
- This is your code written in C.
- Example: program.c

Preprocessor

- Handles all preprocessor directives (#include, #define, macros).
- Output: Preprocessed file (.i)

Compiler

- Converts the preprocessed code into assembly language.
- Checks for syntax errors.
- Output: Assembly file (.s)



Execution

SOURCE → PREPROCESSOR → COMPILER → LINKER → EXECUTABLE

Stages of Execution in C:

Assembler

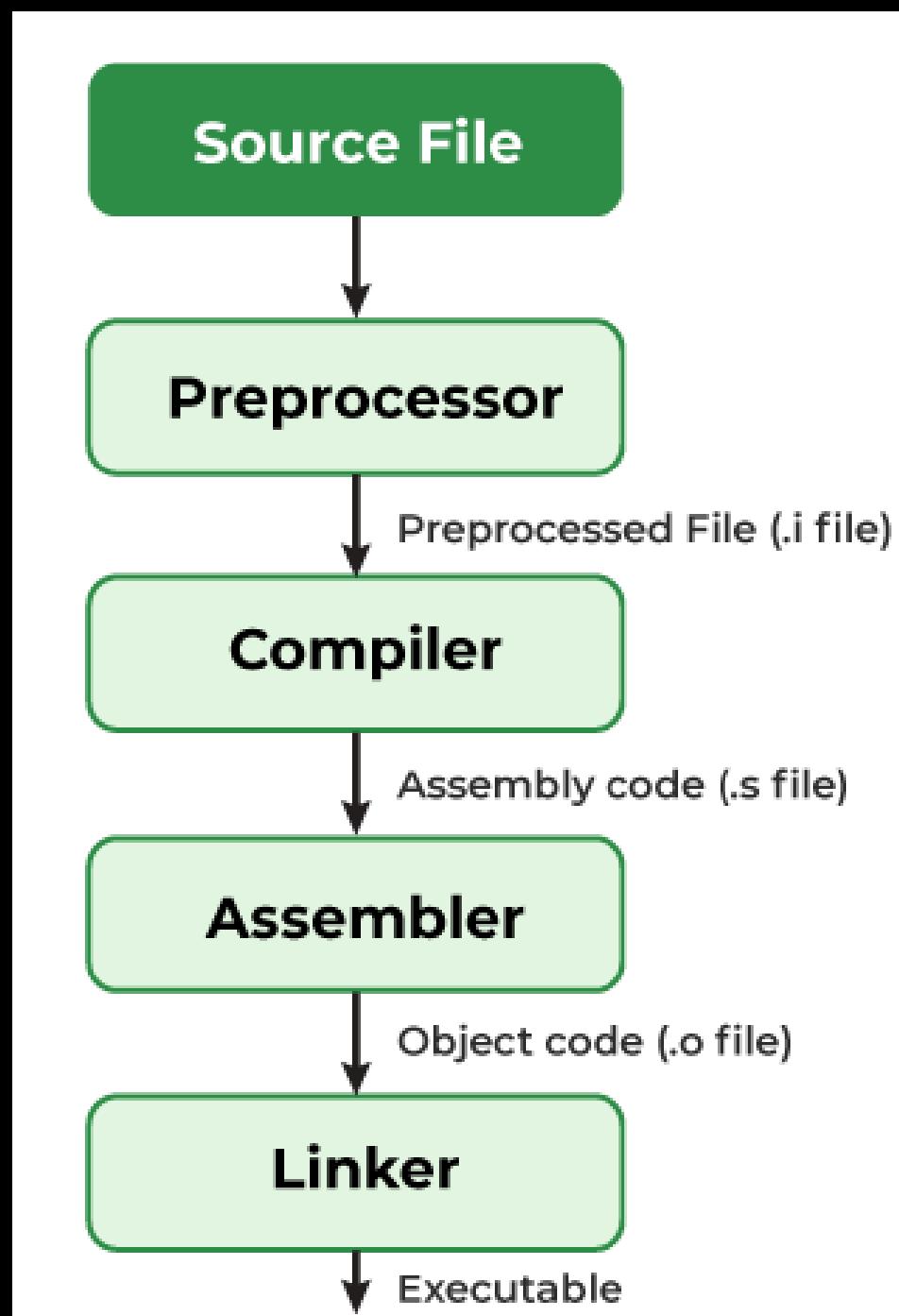
- Translates assembly code into machine code (binary).
- Creates an object file (.o).

Linker

- Combines one or more object files with required libraries.
- Resolves external references (like printf from stdio.h).
- Output: Executable file

Executable

- The final program you can run.



Syntax / Structure

Every C program has these:

- Headers
- The ‘Main’ Function
- Statements and Expressions
- Declaring variables and using various data types for them



Syntax / Structure

```
//Header
#include<stdio.h>

int main(void) // 'Main' Function
{
    printf("Hello World!");
    return 0;
}
```

printf() - Output

Function

printf()

- Used to display output on the screen.
- Defined in <stdio.h>.

Format Specifiers:

- %d → integer
- %f → float
- %c → character
- %s → string
- %lu → length

```
#include <stdio.h>

int main() {
    int age = 20;
    float height = 5.9;
    char grade = 'A';
    char name[] = "Alice";

    printf("Name: %s\n", name);
    printf("Age: %d\n", age);
    printf("Height: %f\n", height);
    printf("Grade: %c\n", grade);

    return 0;
}
```

```
Name: Alice
Age: 20
Height: 5.900000
Grade: A
```

scanf() - Input Function

scanf()

- Used to take input from the user.
- Defined in <stdio.h>.

Format Specifiers:

- %d → integer
- %f → float
- %c → character
- %s → string

```
#include <stdio.h>

int main() {
    int age;
    float height;
    char grade;
    char name[20];

    printf("Enter Name, Age, Height, Grade:\n");
    scanf("%s %d %f %c", name, &age, &height, &grade);

    printf("Name: %s\n", name);
    printf("Age: %d\n", age);
    printf("Height: %.2f\n", height);
    printf("Grade: %c\n", grade);

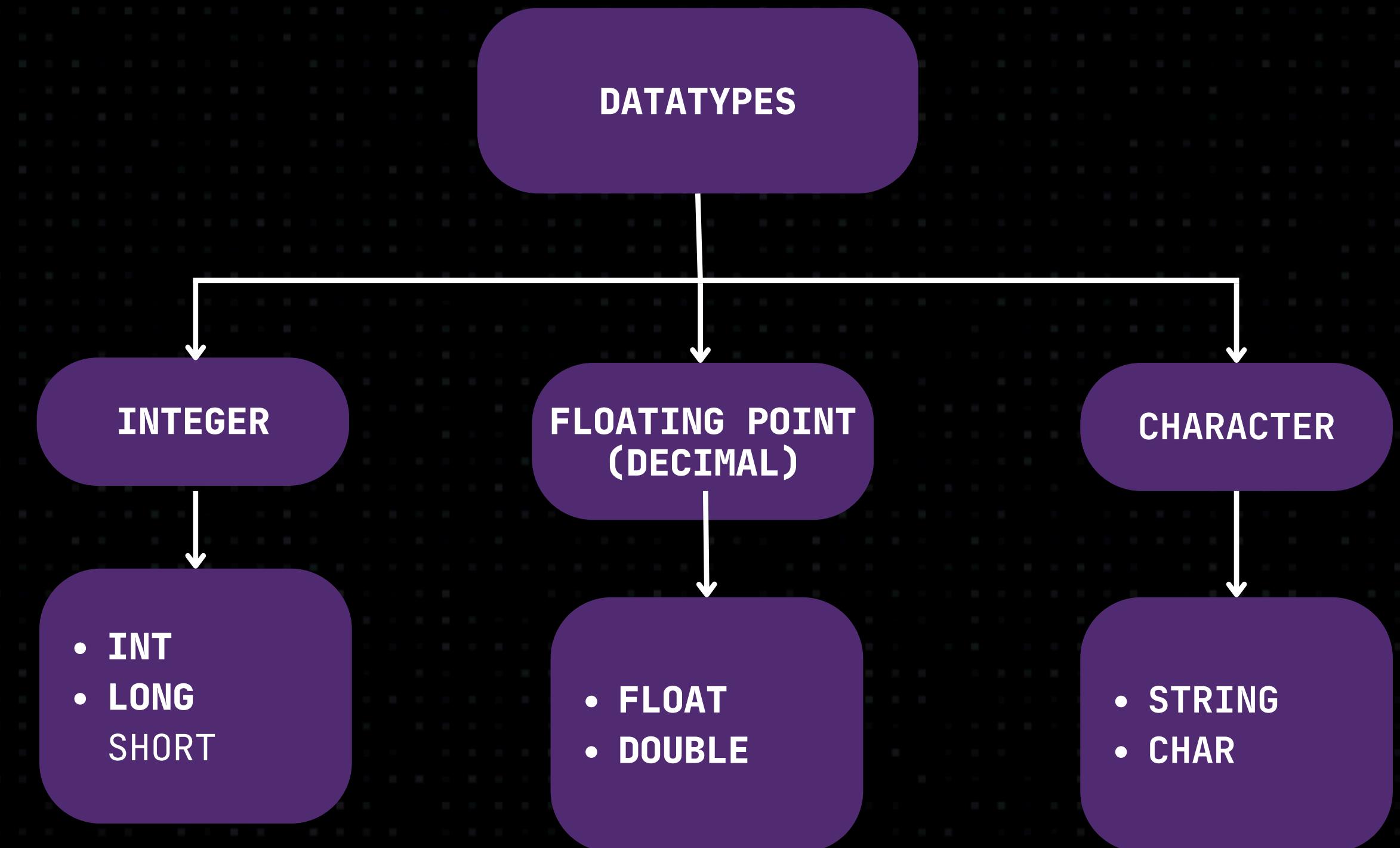
    return 0;
}
```

```
Enter Name, Age, Height, Grade:
soham
19
5.10
A
Name: soham
Age: 19
Height: 5.10
Grade: A
```

Data Types

A data type gives the type of date any variable can store.

For example, numbers, characters, decimals, etc.



Data Types

- int - stores integers (whole numbers), without decimals, such as 123 or -123
- long - (-2,147,483,648 to 2,147,483,647) / much bigger (64-bit)
- short (-32,768 to 32,767)
- float - holds floating values up to 7 digits.
- double - that is used to store floating-point values up to 15 digits
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- string - stores text, such as "Hello World". String values are surrounded by double quotes
- bool - stores values with two states: true or false

Data Types

```
int myNum = 15; // Integer variable
float myFloat = 9.8; // Floating-point variable
double myDouble = 3.141592653589793; // Double variable
char myLetter = 'A'; // Character variable
char myText[] = "Welcome to PFE!"; // Character Array (String) variable
```

```
#include <stdio.h>

int main() {
    printf("Size of short = %lu bytes\n", sizeof(short));
    printf("Size of int = %lu bytes\n", sizeof(int));
    printf("Size of long = %lu bytes\n", sizeof(long));
    printf("Size of long long = %lu bytes\n", sizeof(long long));
    printf("Size of float = %lu bytes\n", sizeof(float));
    printf("Size of double = %lu bytes\n", sizeof(double));
    printf("Size of long double = %lu bytes\n", sizeof(long double));
    return 0;
}
```

ASCII

(AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE)

- Definition: A character encoding standard used in computers and communication devices.
- Purpose: Represents text and control characters using numbers.
- Developed: In 1963 by ANSI (American National Standards Institute).

Key Features

- Uses 7 bits to represent characters (128 unique codes: 0–127).
- Includes:
 - Control Characters (0–31, 127) → e.g., newline, tab, backspace.
 - Printable Characters (32–126) → letters, digits, symbols.
- Extended ASCII uses 8 bits (256 codes), adding more symbols and graphical characters.

7 bits = 7 binary digits

- Example: 1001011 (this is a 7-bit binary number).

- 1000001 (binary 65) → A
- 1100001 (binary 97) → a

Why Important?

- Foundation of text encoding in early computers.
- Still forms the basis for Unicode and other modern encodings.

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	00	[NULL]	32	20	[SPACE]	64	40	@	96	60	~
1	01	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	02	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	03	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	04	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	05	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	06	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	07	[BELL]	39	27	'	71	47	G	103	67	g
8	08	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	09	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	0A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	0B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	0C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	0D	[CARRIAGE RETURN]	45	2D	.	77	4D	M	109	6D	m
14	0E	[SHIFT OUT]	46	2E	-	78	4E	N	110	6E	n
15	0F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	-
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	{DEL}

Global and Local Variables

1. Local Variables

- A local variable is declared inside a function, block, or method and is accessible only within that scope.
- It is created when the block or function is called and destroyed when the block or function finishes execution.
- Scope: Limited to the function or block where it is declared.
- Lifetime: Exists only during the function or block execution.

Global and Local Variables

2. Global Variables

- A global variable is declared outside of all functions and is accessible from any function within the same program after its declaration.
- Scope: Available throughout the entire program, starting from the point of declaration.
- Lifetime: Exists for the entire duration of the program (from start to end).

Global and Local Variables

1. Local Variables

```
void display(){
    int num = 10; //Local variable to the display() function
    printf("Inside display: %d\n", num);
}

int main(){
    int num = 5; //Local variable to main() function
    printf("Inside main: %d\n", num);

    display();
    return 0;
}
```

Output:

Inside main: 5

Inside display: 10

Global and Local Variables

2. Global Variables

```
int globalVar = 100;

int func1(){
    printf("Global Variable inside func1: %d\n", globalVar);
}

int main()
{
    printf("Global Variable inside main: %d\n", globalVar);

    globalVar = 200;
    func1();

    return 0;
}
```

Output:

```
Global Variable inside main: 100
Global Variable inside func1: 200
```

STRUCTURES

In C, a structure is a user-defined data type that groups together different types of data under a single name. These data elements, called members or fields, can be of different types (int, float, char, etc.).

Key points:

1. Structure definition: It defines the blueprint of the structure.
2. Structure variable: A structure variable holds data based on the defined structure.
3. Accessing members: Members are accessed using the dot (.) operator.

Benefits:

- You can store different types of data together.
- Makes your code more organized and readable.

Syntax:

```
1 struct StructureName {  
2     data_type member1;  
3     data_type member2;  
4     // more members  
5 };
```

Example:

```
9 struct Person {  
10    char name[50];  
11    int age;  
12    float height;  
13 };
```

STRUCTURES

```
struct Person {  
    char name[50];  
    int age;  
    float height;  
};  
  
int main() {  
    // Array of structures  
    struct Person group[3] = {  
        {"Alice", 20, 5.5},  
        {"Bob", 22, 5.9},  
        {"Charlie", 25, 6.1}  
    };  
  
    // Accessing members  
    printf("First person's name: %s\n", group[0].name);  
    printf("Second person's age: %d\n", group[1].age);  
    printf("Third person's height: %.2f\n", group[2].height);  
  
    return 0;  
}
```

- `group[0]` → refers to the first structure in the array.
- `group[0].name` → accesses the name field of the first person.
- Similarly `group[1].age` → age of the second person.

Operators

OPERATOR	Type
<code>++, --</code>	Unary Operator
<code>+, -, *, /, %</code>	Arithmetic Operator
<code><, <=, >, >=, ==, !=</code>	Relational Operator
<code>&&, , !</code>	Logical Operator
<code>&, , <<, >>, -, ^</code>	Bitwise Operator
<code>=, +=, -=, *=, %=</code>	Assignment Operator
<code>?:</code>	Ternary Operator

Operators Precedence

OPERATOR	Precedence
<code>++, --</code>	First
<code>*, /, %</code>	Second
<code>+, -</code>	Third
<code><, <=, >, >=</code>	Fourth
<code>==, !=</code>	Fifth
<code>&&</code>	Sixth
<code> </code>	Seventh
<code>=</code>	Last

Unary

Binary