1. Introduction to C Programming

The C programming language is one of the most influential and widely used programming languages in the history of computer science. **Developed in the early 1970s by Dennis Ritchie at Bell Labs, C** played a pivotal role in the development of modern operating systems and system software. Here is a deep history of the C programming language:

Origins:

- In the late 1960s, Bell Labs developed an operating system called Multics, which was written in an assembly language. However, due to concerns about performance and portability, Bell Labs decided to develop a higher-level language for system programming.
- Ken Thompson, together with Dennis Ritchie and other researchers at Bell Labs, started working on a language called B, which was influenced by the earlier BCPL programming language.
- B had limitations and lacked some features required for systems programming. To address these limitations, Ritchie developed a new language called C in 1972.

Development and Evolution:

- C was initially developed on a Digital Equipment Corporation (DEC) PDP-11 computer using assembly language. As the language grew, it became self-hosting, meaning the C compiler was implemented in C itself.
- Ritchie and Brian Kernighan published "The C Programming Language" (often referred to as "K&R C") in 1978. This book became the definitive guide to the language and played a significant role in popularizing C among programmers.
- In 1978, the American National Standards Institute (ANSI) formed a committee to standardize the C language. The resulting standard, known as ANSI C or C89, was published in 1989.
- In 1990, the International Organization for Standardization (ISO) adopted the ANSI C standard, resulting in the publication of ISO/IEC 9899:1990, also known as C90.
- C99, the next major revision of the C language, was published as ISO/IEC 9899:1999. It introduced several new features, such as variable-length arrays, inline functions, and support for complex numbers.
- The latest major revision, C11, was published in 2011. It introduced additional features, including support for threads, atomic operations, and improved Unicode support.

Influence and Popularity:

- C's design and efficiency made it an ideal language for system programming and operating systems. It was used to implement the UNIX operating system, which played a significant role in the development of modern computing.
- C became popular among programmers due to its simplicity, efficiency, and portability. Its low-level features and direct memory access allowed programmers to write high-performance code.
- C's influence extends to many subsequent programming languages. C++ was developed as an extension of C, adding object-oriented programming features. Objective-C, used for macOS and iOS development, is also an extension of C.
- C has been widely used in various domains, including embedded systems, device drivers, game development, and scientific programming.

Legacy and Continued Relevance:

- C remains a fundamental language in computer science education. It provides a solid foundation in programming concepts, such as pointers, memory management, and low-level operations.
- Many modern languages, such as Java, C#, and Python, have borrowed syntax and concepts from C. Understanding C provides a strong basis for learning these languages.
- Despite the emergence of newer languages, C continues to be widely used in performance-critical applications and systems programming where direct control over hardware is required.

The history of the C programming language highlights its pivotal role in the development of computer systems and software. Its simplicity, efficiency, and influence have made it a cornerstone of modern programming.

Types of Programming Languages:

Programming languages are tools that developers use to instruct computers to perform specific tasks. There are numerous programming languages, each designed for different purposes and with varying levels of abstraction. Here are some of the major types of programming languages:

Low-Level Languages:

- Machine Language: The lowest-level programming language, consisting of binary code directly executable by the computer's central processing unit (CPU).
- Assembly Language: A low-level language that uses symbolic instructions instead of binary code, making it more human-readable. It is specific to a particular CPU architecture.

High-Level Languages:

- C: A procedural programming language that provides a good balance between low-level and high-level features.
- C++: An extension of C that includes object-oriented programming (OOP) features.
- Java: A versatile, platform-independent language known for its "write once, run anywhere" principle.
- Python: A high-level, dynamically typed language that emphasizes readability and simplicity. It is widely used for web development, data science, and artificial intelligence.
- JavaScript: A scripting language primarily used for web development to make websites interactive.
- Ruby: A dynamic, object-oriented language known for its simplicity and productivity.
- Swift: Developed by Apple for iOS and macOS app development, known for its speed and safety.
- Kotlin: A modern language that interoperates with Java and is officially supported for Android app development.

Functional Programming Languages:

- Haskell: A purely functional programming language with a strong type system.
- Lisp: A family of programming languages with a unique parenthetical syntax, known for its powerful macro system.

Object-Oriented Programming (OOP) Languages:

- Java: Besides being high-level, Java is also an object-oriented language.
- C++: Combines procedural and object-oriented programming features.
- Python: Supports both procedural and object-oriented programming paradigms.
- C#: Developed by Microsoft, it is used for Windows application development and is also common in game development with Unity.

Scripting Languages:

- Python: Widely used for scripting, automation, and web development.
- Ruby: Known for its concise and readable syntax, often used for scripting and web development.
- Bash: Specifically designed for shell scripting in Unix-like operating systems.

Markup Languages:

- HTML (Hypertext Markup Language): Used for creating the structure of web pages.
- XML (eXtensible Markup Language): A versatile markup language for representing structured data.

Domain-Specific Languages (DSLs):

- SQL (Structured Query Language): Used for managing and manipulating relational databases.

- CSS (Cascading Style Sheets): A language for styling HTML documents in web development.

Concurrency-Oriented Languages:

- Erlang: Designed for building scalable and fault-tolerant distributed systems.
- Go (Golang): Developed by Google, known for its simplicity and efficiency in concurrent programming.

Statistical and Data Analysis Languages:

- R: Primarily used for statistical analysis and data visualization.
- Julia: Known for its speed and efficiency in numerical and scientific computing.

Esoteric Languages:

- Brainfuck: An esoteric language with minimalistic syntax, designed as a challenge for programmers.

These categories are not mutually exclusive, and many languages fall into multiple categories based on their features and applications. Additionally, new languages continue to emerge as technology evolves, addressing specific needs and challenges in the development landscape.

Where C Programming Stands:

The C programming language is a general-purpose, procedural programming language. It is often categorized as a high-level programming language, but it also includes some features found in low-level languages. Here's a breakdown of where C stands in various programming language categories:

Procedural Programming:

C is primarily a procedural programming language. It follows a procedural paradigm, where the program is structured as a series of procedures or functions, each performing a specific task.

Low-Level Features: While C is considered a high-level language, it provides low-level features like direct memory manipulation through pointers and bitwise operations, allowing for fine-grained control over hardware.

Imperative Programming: C is an imperative language, meaning that programs are composed of sequences of statements that change a program's state.

Middle-Level Language: C is often referred to as a "middle-level" language because it combines both high-level and low-level language features. It allows for efficient manipulation of hardware resources while providing abstractions that make it more readable and portable than low-level languages like assembly.

General-Purpose: C is a general-purpose programming language, meaning it can be used for a wide range of applications. It is commonly used for system programming, embedded systems, and application development.

Not Object-Oriented: Unlike some other languages like C++ or Java, C is not an object-oriented programming (OOP) language. It lacks built-in support for features such as classes and objects.

In summary, C stands as a versatile and widely used programming language that combines elements of both high-level and low-level languages. Its design principles prioritize efficiency, control over hardware, and portability, making it suitable for a variety of applications, especially those that require close interaction with hardware or performance-critical tasks.

Version C Programming Language:

The C programming language has evolved over the years through various versions and standards. Here's a brief overview of the version history of the C programming language:

1. Original Version (1972):

- Developed by Dennis Ritchie at Bell Labs.
- Initially used to implement the Unix operating system.
- Informally referred to as "K&R C" after the authors Brian Kernighan and Dennis Ritchie.

2. ANSI C (1989):

- The American National Standards Institute (ANSI) standardized the C language in 1989.
- Known as ANSI C or C89.
- This version introduced several new features, including function prototypes, the void keyword, and the header file <stddef.h>.

3. C99 (1999):

- The ISO/IEC 9899:1999 standard, commonly known as C99, introduced several new features to the language.
- New features included single-line comments starting with //, variable declarations anywhere in a block, new data types like long long int, and additional standard library functions.

4. C11 (2011):

- The ISO/IEC 9899:2011 standard, known as C11, introduced further enhancements to the language.
- New features included _Generic keyword, type-generic expressions (generics), and multi-threading support through the <threads.h> library.
- It also incorporated features from the C99 Technical Corrigendum 3.

5. C17 (2017):

- The ISO/IEC 9899:2017 standard, often referred to as C17, is a bug-fix release to C11.
- It includes corrections and clarifications to the C11 standard but does not introduce major new features.

6. C23 (Expected in 2023):

C23 is the upcoming revision of the C programming language standard, expected to be published in 2024. It will be the successor to the current standard, C18 (ISO/IEC 9899:2018), which was published in 2018.

C23 is expected to add a number of new features, including:

- Modules: Modules provide a more structured way to organize C code, making it easier to reuse code and manage large projects.
- Dynamic arrays: Dynamic arrays are arrays whose size can be changed at runtime. This makes them more flexible than static arrays, which have a fixed size at compile time.
- Designated initializers: Designated initializers allow you to initialize specific elements of an array or structure without having to explicitly initialize all of the elements.
- _Static_assert macro: The _Static_assert macro allows you to express compile-time assertions about your code. This can help to catch errors early in the development process.
- In addition to these new features, C23 is also expected to make a number of improvements to existing features, such as:
- Improved support for const: C23 will make it easier to use the const qualifier to restrict the modification of data.
- Standardization of the typeof() operator: The typeof() operator, which returns the type of an expression, will be standardized in C23.
- Clarification of the compatibility rules for structure, union, and enumerated types: The compatibility rules for structure, union, and enumerated types will be clarified in C23.
- C23 is still under development, and the final list of features is not yet finalized. However, the features that are currently being considered are expected to make C a more powerful, flexible, and easier-to-use language.

How to Learn C Programming:

Like an English, C programming is not our mother tongue. So, we have to take efforts to learn the C programming same as that of we have taken to learn the English language. Let's see the learning path English.

```
English  ==>  Character set --> Words --> Sentence --> Paragraph --> Chapter --> Book--> Library
```

In a similar manner we have to proceed in C Programming as,

```
                                  Constant
C programming ==> Character Set -->  Keyword  --> Instruction --> Program --> Module --> Software
                                  Variable
```

C Character Set:

The character set of a programming language refers to the set of characters and symbols that are valid and meaningful in that programming language. Each programming language has its own specific character set, which includes letters, digits, punctuation marks, special symbols, and other elements used for writing code. The character set is defined by the language's specifications and standards, and it plays a crucial role in determining how programs are written and interpreted or compiled.

In C programming, the character set refers to the set of characters that can be used in C programs. The character set is based on the ASCII (American Standard Code for Information Interchange) character encoding, which assigns a unique numeric value to each character. The ASCII character set includes both printable and non-printable characters. Here are some key aspects of the character set in C programming:

1. Basic Characters:

- The character set includes basic characters such as letters(alphabets) (both uppercase and lowercase), digits, and Special Symbols.
- For example:
    - Alphabets:
        - Uppercase A to Z
        - Lowercase a to z
    - Digits: 0 to 9
    - Special Symbols: !@#$%^&*(_)+ ....

2. Escape Sequences:

- C programming allows the use of escape sequences to represent characters that are difficult to type directly or are non-printable.
- Examples of escape sequences include \n (newline), \t (tab), \' (single quote), \" (double quote), and \\ (backslash).

3. Special Characters:

Certain characters have special meanings in C programming. For example:

- ' (single quote) is used to represent character constants.
- " (double quote) is used to define string literals.
- \ (backslash) is used for escape sequences and as a part of special character representations.

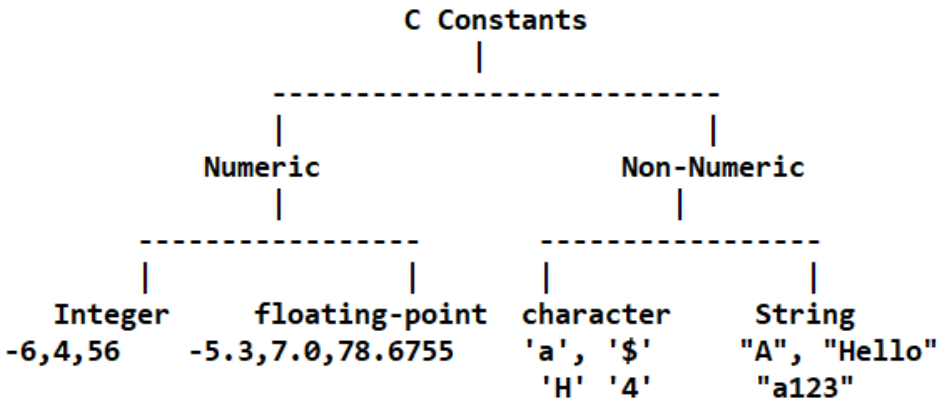
4. Control Characters:

- The ASCII character set includes control characters, which are non-printable characters used for control purposes. Examples include newline (\n), carriage return (\r), and tab (\t).

Control Characters				Graphic Symbols											
Name	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex
NUL	0	0000000	00	space	32	0100000	20	@	64	1000000	40	'	96	1100000	60
SOH	1	0000001	01	!	33	0100001	21	A	65	1000001	41	a	97	1100001	61
STX	2	0000010	02	"	34	0100010	22	B	66	1000010	42	b	98	1100010	62
ETX	3	0000011	03	#	35	0100011	23	C	67	1000011	43	c	99	1100011	63
EOT	4	0000100	04	$	36	0100100	24	D	68	1000100	44	d	100	1100100	64
ENQ	5	0000101	05	%	37	0100101	25	E	69	1000101	45	e	101	1100101	65
ACK	6	0000110	06	&	38	0100110	26	F	70	1000110	46	f	102	1100110	66
BEL	7	0000111	07	'	39	0100111	27	G	71	1000111	47	g	103	1100111	67
BS	8	0001000	08	(	40	0101000	28	H	72	1001000	48	h	104	1101000	68
HT	9	0001001	09	)	41	0101001	29	I	73	1001001	49	i	105	1101001	69
LF	10	0001010	0A	*	42	0101010	2A	J	74	1001010	4A	j	106	1101010	6A
VT	11	0001011	0B	+	43	0101011	2B	K	75	1001011	4B	k	107	1101011	6B
FF	12	0001100	0C	,	44	0101100	2C	L	76	1001100	4C	l	108	1101100	6C
CR	13	0001101	0D	–	45	0101101	2D	M	77	1001101	4D	m	109	1101101	6D
SO	14	0001110	0E	.	46	0101110	2E	N	78	1001110	4E	n	110	1101110	6E
SI	15	0001111	0F	/	47	0101111	2F	O	79	1001111	4F	o	111	1101111	6F
DLE	16	0010000	10	0	48	0110000	30	P	80	1010000	50	p	112	1110000	70
DC1	17	0010001	11	1	49	0110001	31	Q	81	1010001	51	q	113	1110001	71
DC2	18	0010010	12	2	50	0110010	32	R	82	1010010	52	r	114	1110010	72
DC3	19	0010011	13	3	51	0110011	33	S	83	1010011	53	s	115	1110011	73
DC4	20	0010100	14	4	52	0110100	34	T	84	1010100	54	t	116	1110100	74
NAK	21	0010101	15	5	53	0110101	35	U	85	1010101	55	u	117	1110101	75
SYN	22	0010110	16	6	54	0110110	36	V	86	1010110	56	v	118	1110110	76
ETB	23	0010111	17	7	55	0110111	37	W	87	1010111	57	w	119	1110111	77
CAN	24	0011000	18	8	56	0111000	38	X	88	1011000	58	x	120	1111000	78
EM	25	0011001	19	9	57	0111001	39	Y	89	1011001	59	y	121	1111001	79
SUB	26	0011010	1A	:	58	0111010	3A	Z	90	1011010	5A	z	122	1111010	7A
ESC	27	0011011	1B	;	59	0111011	3B	[	91	1011011	5B	{	123	1111011	7B
FS	28	0011100	1C	<	60	0111100	3C	\	92	1011100	5C	|	124	1111100	7C
GS	29	0011101	1D	=	61	0111101	3D	]	93	1011101	5D	}	125	1111101	7D
RS	30	0011110	1E	>	62	0111110	3E	^	94	1011110	5E	~	126	1111110	7E
US	31	0011111	1F	?	63	0111111	3F	_	95	1011111	5F	Del	127	1111111	7F

Constant, variable and keyword in C:

Constant: Constants are elements in the program which having fix value, and which cannot be changed or altered. There are two different types of constants: Numeric and Non-Numeric.



Variable: Variable is element in the program which may change during the program execution. We, the formula of simple interest

- si = (p*r*n) / 100 ;
  - Variables ---> p, r, n, si
  - Constant ---> 100

As the values of p, r, n and si can be changed it is known as variable and 100 is fix value, its meaning can not be changed or altered therefore it is called as constant.

Keyword: Keywords are reserved words, whose meaning is already known to, explained to or defined to compiler. As Keywords special words they cannot be used for any other purpose. There are 32 keywords in C. and These are as shown

Auto	Double	Int	Struct
Break	Else	Long	Switch
Case	Enum	Register	Typedef
Char	Extern	Return	Union
Const	Float	Short	Unsigned
Continue	For	Signed	Void
Default	Goto	Sizeof	Volatile
Do	If	Static	While

Instruction in C Programming:

An instruction is a set of operations or a command that directs a computer's CPU (Central Processing Unit) to perform a specific action. Instructions are the fundamental building blocks of machine code i.e., Program.

There are 4 different types of instructions.

- Input-Output Instruction
- Type Declaration. Instruction
- Arithmetic Instruction
- Control Instruction

Input-Output Instruction: The aim behind this instruction, is to perform the input and output operation. Means take the data from outside word and store it into the computer's memory. And represent the data present in the memory to outside word.

Let's start with output instruction output instruction First.

Output Instruction: As stated earlier this instruction is used to display the data from the computer's memory to the standard output device i.e., Screen or Monitor. As a novice programmer it is hardly impossible to write a code to create the communication channel between your program and screen. So, C uses functions from the standard input/output library (`stdio.h`) to perform output operations. The most commonly used function for output in C is `printf()`.

Syntax of printf():

```
printf("<format_string>");
```

Here, <format_string> is the string that you want to display on screen.

Let's see the First Program i.e., Welcome Program

```
#include <stdio.h>
int main()
{
   printf("Hello, World!"); // This line outputs "Hello, World!" and a newline character
   return 0;
}
```

In this example:

- #include <stdio.h> includes the standard input/output library.
- printf("Hello, World!"); is the statement that outputs the text "Hello, World!" to the console.
- *return 0; Returns int value 0 to operating system, which indicates successful execution*
- when compiled and executed, will display "Hello, World!" on the console.

Keep in mind that the choice of output function may vary depending on the context and requirements. Other output functions include `puts()` and `putc()`.

Comment in C Programming:

In C programming, comments are used to provide explanations or annotations within the code. Comments are ignored by the compiler and serve the purpose of making the code more understandable for programmers. There are two types of comments in C:

1. **Single-line comments:**

Single-line comments begin with // and continue until the end of the line. Anything written after // on that line is considered a comment.

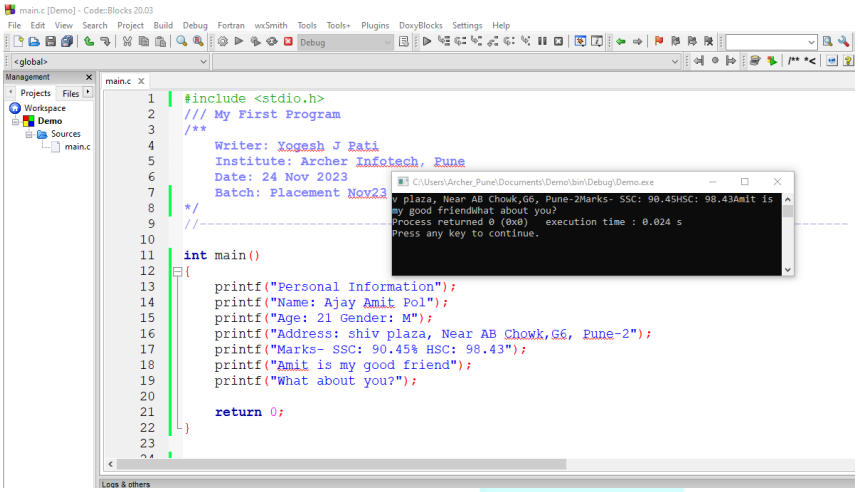
```
// This is a single-line comment
Int main()
{
int x = 10; // This comment is at the end of a line of code
...
}
```

2. **Multi-line comments:**

Multi-line comments start with /* and end with */. Everything between /* and */ is treated as a comment, even if it spans multiple lines.

```
/* This is a
   multi-line comment */
int main()
{
  int y = 20;
/*
---------
*/
  return 0;
}
```

Write a program to display the Your personal details on screen.



This will display the unformatted output as shown. To display the output in well formatted manner, *Escape Sequence Characters* are used.

Escape Sequence Characters in C:

Escape sequences are a special set of characters in the C programming language that are used to represent non-printable characters or to control the formatting of text. Escape sequences are always preceded by a backslash (\) character.

Escape Sequence	Meaning	Description
`\n`	Newline	Moves the cursor to the beginning of the next line.
`\t`	Horizontal Tab	Inserts a horizontal tab, used for creating space or alignment.
`\a`	Bell	Produces an audible alert or bell sound.
`\b`	Backspace	Moves the cursor back one position.
`\v`	Vertical Tab	Inserts a vertical tab.
`\r`	Carriage Return	Moves the cursor to the beginning of the current line.
`\f`	Form Feed	Advances to the next page or form.
`\'`	Single Quote	Prints a single quote character.
`\"`	Double Quote	Prints a double quote character.
`\\`	Backslash	Prints a backslash character.
`\?`	Question Mark	Prints a question mark character.
`%%`	Percent Sign	Prints a percent sign character.
`\0`	Null Character	Represents the null character, often used to terminate strings.

Above program with Escape Sequence Characters used within it.

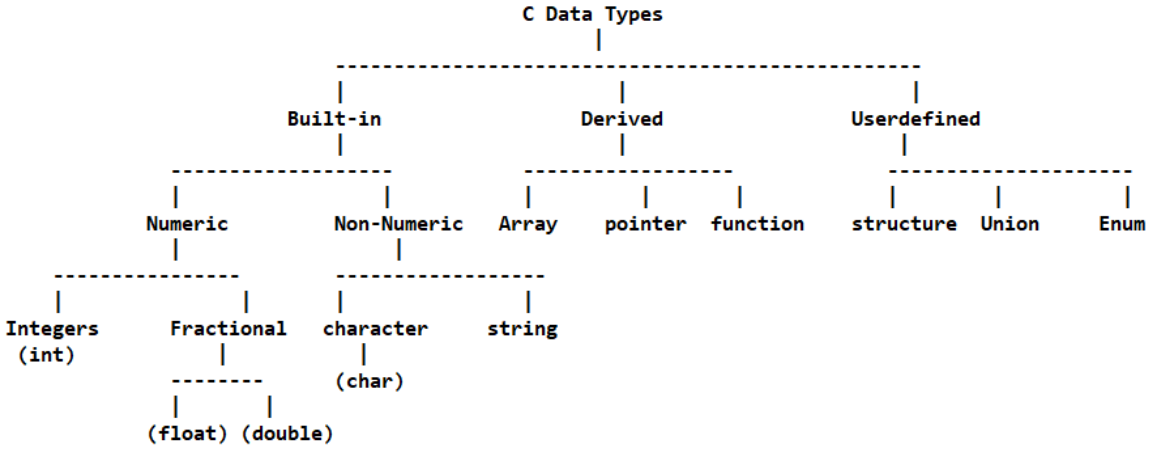
```c
#include <stdio.h>
int main()
{
    printf("\n ****** Personal Information ******\a\a\a");
    printf("\n Name: Ajay Amit Pol");
    printf("\n Age: 21 \t Gender: \'M\' ");
    printf("\n Address: \"shiv plaza\", Near AB Chowk,G\\6, Pune-2");
    printf("\n Marks- SSC: 90.45%% \t HSC: 98.43%%");
    printf("\n Amitraj is my good friend \r ABHI");
    printf("\n What about you\?");
    printf("\n ABC\bDEFGH\b\bIJKLMNOP");
    return 0;
}
```

```
Select C:\Users\Archer_Pune\Documents\Demo\bi...

****** Personal Information ******
Name: Ajay Amit Pol
Age: 21         Gender: 'M'
Address: "shiv plaza", Near AB Chowk,G\6, Pune-2
Marks- SSC: 90.45%      HSC: 98.43%
ABHIraj is my good friend
What about you?
ABDEFIJKLMNOP
Process returned 0 (0x0)   execution time : 0.028 s
Press any key to continue.
```

Type Declaration Instruction: we are writing the program means we want to take the data from user, store it into the computer's memory, process it and display the result. As a part of this process we used printf() to display the data from memory on to the screen. Now we want to take the data as an input from user, and store it in to the memory. To store that input data memory is allocated using type declaration instruction. Means in simple words, the Aim of the type declaration instruction is memory allocation to store the data.

The data, which is input to computer, is divided into different groups or categories on the basis of memory requirement to store the data, where individual group is known as type and collectively it is known as data types. The diagram shows different datatypes in C and there are several keywords used to represent different data types.



Keywords for data types in C play a crucial role in specifying the type of data that a variable can hold. These keywords are used in type declaration instructions, helping the compiler understand how to allocate memory for the variable and number of bytes.

Type	Bits	Minimal Range
char	8	–127 to 127
unsigned char	8	0 to 255
signed char	8	–127 to 127
int	16 or 32	–32,767 to 32,767
unsigned int	16 or 32	0 to 65,535
signed int	16 or 32	Same as **int**
short int	16	–32,767 to 32,767
unsigned short int	16	0 to 65,535
signed short int	16	Same as **short int**
long int	32	–2,147,483,647 to 2,147,483,647
long long int	64	$-(2^{63}-1)$ to $2^{63}-1$ (Added by C99)
signed long int	32	Same as **long int**
unsigned long int	32	0 to 4,294,967,295
unsigned long long int	64	$2^{64}-1$ (Added by C99)
float	32	1E–37 to 1E+37 with six digits of precision
double	64	1E–37 to 1E+37 with ten digits of precision
long double	80	1E–37 to 1E+37 with ten digits of precision

Type Modifiers in C: In C programming, type modifiers such as short, long, signed, and unsigned are used to modify the behaviour of basic data types. These modifiers affect the size, range, and sign (for integers) of the data types.

Sure, here is a table explaining type modifiers short, long, signed, and unsigned in C Programming:

- **short:** Modifies the size of the base data type to half its original size. e.g., short int and format specifiers are %hd or %hi.
- **long:** Modifies the size of the base data type to double its original size. e.g., long int and format specifiers are %ld or %li.
- **signed:** Indicates that the base data type can store both positive and negative values. e.g., signed int and format specifiers are %d or %i.
- **unsigned:** Indicates that the base data type can only store non-negative values (including zero). e.g., unsigned int and format specifiers are %u

Here are some additional notes:

- The short and long modifiers can be applied to the int, char, and double data types.
- The signed and unsigned modifiers can be applied to the char, int, short, and long data types.
- The default data type for integers is signed int.
- The default data type for characters is signed char.

Syntax of Type Declaration Instruction:

<data_type> <identifier>;

<data_type>: Write the keyword of data type.

<identifier>: Assigned the name to allocated location.

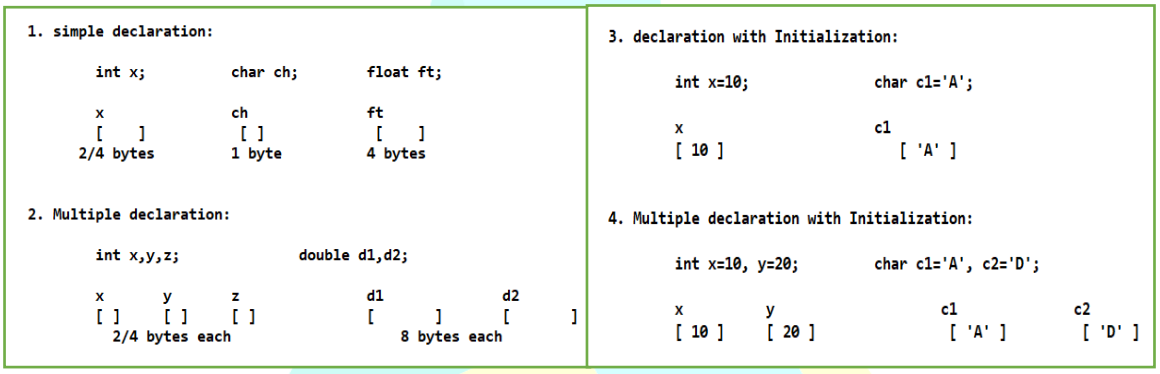
Identifier: In C programming, an identifier is a name given to a program element, such as a

variable, function, array, or any other user-defined item. Identifiers are used to uniquely identify and refer to various elements within a C program.

Rules for naming identifiers in C:

- An identifier must start with a letter (uppercase or lowercase) or an underscore _.
- After the initial letter or underscore, an identifier can be followed by letters, digits, or additional underscores.
- C is case-sensitive, so uppercase and lowercase letters are treated as distinct. For example, myVar and MyVar would be considered different identifiers.
- Spaces and most special characters (except underscores) are not allowed within identifiers. For example, my variable is not a valid identifier.
- Identifiers cannot be the same as C keywords or reserved words. For example, you cannot name a variable int or while as these are reserved words in C.

There are Four different subtypes of type declaration instruction.



Write a program to initialize different types of variables and display their values

```c
#include <stdio.h>
int main()
{
    int x=340;
    char ch='A';
    float ft=4.5;
    double db=54.754545;

    printf("\n value of x is: %d",x);
    printf("\n ch=%c \t ft=%0.3f \n Val of db is %lf",ch,ft,db);

    return 0;
}
```

```
C:\Users\Archer_Pune\Documents\Demo\bin\Deb...
value of x is: 340
ch=A    ft=4.500
Val of db is 54.754545
Process returned 0 (0x0)   execution time : 0.023 s
Press any key to continue.
```