Shastri 4th Semester

Computer Science

Unit: 2nd

INTRODUCTION OF "C LANGUAGE"

C is a high-level programming language that was developed in the early 1970s by Dennis Ritchie at Bell Labs. C is a procedural programming language, which means that it is designed to support the creation of algorithms that are executed step-by-step. C is widely used for a variety of applications, including operating systems, device drivers, embedded systems, and high-performance scientific and engineering applications. One of the key features of the C language is its low-level access to computer memory, which makes it a popular choice for systems programming. C is also widely used as a systems programming language because it is relatively lightweight and efficient.

HISTORY OF C LANGUAGE

C programming language was developed in the early 1970s by Dennis Ritchie at Bell Labs. The development of C can be traced back to the early 1970s, when Ritchie, as part of a team at Bell Labs, began working on a new operating system called UNIX. The team wanted to create a high-level programming language that could be used to write the system's software. The C programming language was created as part of this project, and it was used to write the UNIX operating system, as well as many of the tools that were used to develop it.

The first version of C was called "K&R C," named after the authors of the book "The C Programming Language," which was published in 1978. The book became a popular reference for the C language and it described the then-new language, which was still called "C with Classes" at the time.

The C programming language quickly gained popularity, and it was widely adopted for systems programming, as well as for writing other types of high-performance software. C compilers were developed for many different platforms, including mainframes, minicomputers, and microcomputers, making C one of the most widely available and widely used programming languages of all time.

Throughout its history, C has undergone several standardization processes, the first in 1989, known as ANSI C which was later ISO standardized in 1990. And the latest is C11 which was published in 2011.

C programming language continues to be widely used today, particularly in systems programming and the development of embedded systems and other low-level applications. The language also forms the basis of many popular programming languages, such as C++ and Objective-C, which have incorporated many of C's features and concepts into their own designs.

FEATURES OF "C LANGUAGE"

- C is a procedural programming language, which means that it is designed to support the creation of algorithms that are executed step-by-step.
- C is a high-level programming language that provides a structured approach to program design.
- C provides low-level access to computer memory, which makes it a popular choice for systems programming.
- C has a relatively small and simple set of keywords, making it easy to learn and understand.
- C is an efficient language, both in terms of the speed of execution and the use of computer memory.
- C has a rich set of built-in operators and functions for performing various operations, like arithmetic, logical, bitwise operations etc.
- C has the ability to handle low-level activities, like memory management and bit manipulation.
- C allows you to create reusable code in the form of functions and libraries.
- C is a portable language which allows you to run the same code on different machines with little or no modification.

- C has an active user community and a wide variety of libraries, frameworks and open-source software available that can be easily integrated into C programs.
- C allows for the creation of high-performance, efficient code through its control over memory management and low-level operations.

- C is widely used for systems programming, including the development of operating systems, device drivers, and embedded systems.

- C is a flexible language that supports both structured programming and pointer manipulation.

- C has a rich set of operators and a standard library which makes it easy to handle complex operations and data structures.

- C is compatible with most platforms and operating systems, making it a highly portable language.

- C allows for direct manipulation of hardware and memory, making it suitable for low-level programming tasks such as device drivers and embedded systems.

- C encourages modularity, scalability, and maintainability of source code by using functions, structured data types, and separate compilation.

- C has been actively used for a long time, and thus a lot of third-party libraries and tools are available for various purposes.

- C has C++ as its superset, so C code can be easily integrated with C++ code.

- C has a simple, clean syntax which makes it a great choice for both beginners and experienced programmers.

C language: Character Set

The character set of a programming language defines the set of characters that are used to represent the source code of a program. In the case of the C language, the standard character set includes the following:

- Upper and lowercase letters of the English alphabet (A-Z, a-z)
- Digits (0-9)
- Special characters such as punctuation marks, mathematical symbols, and whitespace characters
- A set of control characters, such as newline, tab, and backspace.

In addition to the standard character set, C also supports the use of wide characters and multibyte characters. Wide characters are used to represent characters from other character sets, such as characters from Asian languages. Multibyte characters are used to represent characters that are not part of the standard character set, such as non-English letters and special symbols.

It's worth noting that the C language standard also defines the behavior of the C implementation when it comes to character set and character handling. Some implementations may have slight variations in the character set, but the most widely used ASCII and extended ASCII character sets are commonly used in C language.

It's also important to note that while C is a powerful language, it's not the best choice for handling Unicode characters and internationalization, and libraries like ICU or GLib are commonly used to handle such cases.

PRE-PROCESSORS OF C LANGUAGE

In the C programming language, pre-processors are a type of program that processes the source code before it is passed to the compiler. They are used to make modifications to the source code, such as including other files, defining macros, and expanding macro invocations.

There are several pre-processors in C, which include:

#include: This pre-processor is used to include the contents of another file in the source code. This is commonly used to include header files, which contain function declarations and other information that is needed by the program.

#define: This pre-processor is used to define macros, which are a way of replacing one piece of text with another. Macros are commonly used to define constants, as well as to create short-hand notation for commonly used code.

#ifdef and #ifndef: These pre-processors are used to conditionally include or exclude portions of the code. They allow you to write code that is only compiled if a certain macro is defined, or if it is not defined.

#pragma: This pre-processor is used to issue special commands to the compiler. This might include turning on or off certain warning or optimization flags.

#error and #warning: These pre-processors can be used to include error and warning messages in the source code. These messages will be displayed when the code is pre-processed, allowing you to catch and address any issues before the code is compiled.

In short, pre-processors allow to control of the compilation process and prepare the source code for it, by replacing macros, including or excluding code, or even adding custom error/warning messages.

TYPES OF PRE-PROCESSORS OF C LANGUAGE

The C programming language has several types of pre-processors, which can be broadly categorized into three main types:

File inclusion pre-processors: These pre-processors include the contents of another file in the source code. The #include pre-processor is an example of this. It is used to include the contents of header files, which contain function declarations and other information that is needed by the program.

Macro pre-processors: These pre-processors define and manipulate macros, which are a way of replacing one piece of text with another. Macros are commonly used to define constants, as well as to create short-hand notation for commonly used code. Examples of this type of pre-processors are #define, #undef, #ifdef, #ifndef

Conditional compilation pre-processors: These pre-processors are used to conditionally include or exclude portions of the code. They allow you to write code that is only compiled if a certain macro is defined, or if it is not defined. Examples of this type of pre-processors are #if, #ifdef, #ifndef, #else, #elif, #endif

Other pre-processors: This includes #pragma which is used to issue special commands to the compiler, #error, and #warning which can be used to include error and warning messages in the source code.

The pre-processors are processed by the preprocessor which is a separate program that runs before the compiler, in the compilation process and modifies the source code.

HEADER FILES

In the C programming language, a header file is a file that contains declarations for functions, variables, and other constructs that can be used in a C program. These declarations, also known as prototypes, provide the compiler with the information it needs to understand how to use the functions, variables, and other constructs that are defined in the header file.

Header files typically have a .h file extension, for example, stdio.h, string.h, etc.

Header files can be included in a C program using the #include pre-processor directive. For example, the following line will include the contents of the studio.h header file in the program:

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

Some common uses of header files include:

- Declaring functions, variables, and constants that are provided by a library.
- Providing definitions for structs, enumerations, and other user-defined data types.
- Declaring external variables and functions.
- Declaring global variables, constants and function prototypes that are used in multiple source files, so that they can be shared across multiple source files
- Declaring Macros and Inline functions
- Including a header file has the effect of copying the contents of the header file into the source code file at the point where the #include directive appears. This allows the declarations in the header file to be used in the rest of the program and allows the compiler to properly link the functions and variables defined in the header file to the rest of the program.

C LANGUAGE: VARIABLES AND IDENTIFIERS

In C language, a variable is a named location in memory that can be used to store a value. A variable must be declared before it can be used, and its type determines the kind of value it can store. The basic types in C language include int, float, double, char, and _Bool.

An identifier is a name given to a variable, function or other constructs in C language. Identifiers are used to refer to these entities in the program.

C language has a set of rules for naming identifiers which include:

- An identifier can only contain letters, digits, and underscores.
- The first character of an identifier must be a letter or an underscore.
- Identifiers are case-sensitive, meaning that "myVariable" and "myvariable" are two different identifiers.
- Identifiers cannot be a keyword or predefined identifier. C has a set of keywords such as if, while, for, etc. which have specific meaning and cannot be used as identifiers.

For example, the following are valid variable declarations in C:

int age; float price; double pi = 3.14; char letter = 'A';

It's worth noting that good variable naming conventions can make the code more readable and maintainable. It's a good practice to use meaningful, descriptive names for variables and avoid using short or generic names like "x" or "temp".

Also, the C standard recommends using camelCase or snake_case conventions for variable names and camelCase or PascalCase conventions for naming functions and structs.

C LANGUAGE VARIABLES WITH EXAMPLES

A variable in C is a place in memory where a programmer can store a value. Variables have a name and a data type, which tells the compiler the size and layout of the variable's memory, as well as how to interpret the stored value.

For example, consider the following line of code:

int age = 25;

This creates a variable called "age" with the data type of "int", and assigns it the value of 25.

The syntax for creating a variable in C is

data_type variable_name = value;

Where data_type can be one of C's standard datatypes such as int, float, char, etc, variable_name is the name of the variable and value is the initial value of the variable.

Here are some examples of C variables:

int x = 10;

float y = 3.14;

char ch = 'A';

bool b = true;

The first line creates an integer variable named "x" and assigns it the value of 10.

The second line creates a floating-point variable named "y" and assigns it the value of 3.14.

The third line creates a character variable named "ch" and assigns it the value of 'A'

The forth line creates a Boolean variable named "b" and assigns it the value of true

Once a variable is created, its value can be accessed and modified throughout the program.

THE GLOBAL AND LOCAL VARIABLES OF THE C LANGUAGE

In C, a variable can be defined as either a global variable or a local variable, depending on where it is declared in the program.

Global variables: Global variables are declared outside of any function, usually at the top of the file before any functions. Global variables are available for use throughout the entire program, including within all functions. The value of a global variable can be accessed and modified by any function in the program, and it retains its value even after a function call is completed.

For example:

```
int global_var = 5;

int main() {

  global_var = 10;

  return 0;

}
```

Here, the variable global_var is a global variable and can be accessed by any function in the program,

Local variables: On the other hand, local variables are declared within a function, and they are only available for use within that function. They are also known as automatic or stack variables. The value of a local variable is only accessible within the function in which it is declared, and the variable is destroyed when the function call is completed.

For example:

```
int main() {

  int local_var = 5;

  return 0;

}
```

Here, the variable local_var is a local variable and can only be accessed within the main function.

It is a good practice to use local variables as much as possible, because they are easier to debug, and can prevent unexpected changes or conflicts in the values of the global variables.

In C, variables can be grouped into several categories based on their storage class and scope. The storage class of a variable determines where the variable is stored in memory, and the scope of a variable determines where the variable can be accessed in the program.

Automatic variables: Automatic variables are also known as local variables, which are declared inside a function and are created and initialized every time the function is called, and their values are destroyed when the function returns. They are stored in the stack memory.

Static variables: Static variables are also known as internal linkage variables, which are also declared inside a function but their value persists between function calls, are only created once when the program is loaded, and are destroyed when the program ends. They are stored in the data section of the program

External variables: External variables are also known as global variables, which are declared outside of any function, usually at the top of the file before any functions. They are created once when the program is loaded and are destroyed when the program ends. They are also stored in the data section of the program

Register variables: Register variables are similar to automatic variables, but they are stored in the CPU registers instead of memory, this is used to improve performance by reducing the memory access time. The register keyword can be used to request that a variable be stored in a register. The compiler will choose which variable to store in a register, but it will not guarantee that the variable will be stored in a register. It is recommended to use automatic and registered variables for local variables because they have faster access time and limited scope, which reduces the risk of unexpected changes. Using variables of appropriate storage class, as it helps to manage memory and improve performance. Global variables should be used with caution, as they can cause conflicts or unexpected changes in the program if not handled properly.

a table that lists the different types of variables in C and their characteristics:

Type of Variable	**Description**	**Example**
Automatic variables	Variables are declared inside a function and are created and initialized every time the function is called, and their values are destroyed when the function returns. They are stored in the stack memory	int x;
Static variables	Variables that are declared inside a function but whose value persists between function calls, are only created once when the program is loaded,	static int y;

	and are destroyed when the program ends. They are stored in the data section of the program	
External variables	Variables that are declared outside of any function, usually at the top of the file before any functions. They are created once when the program is loaded and are destroyed when the program ends. They are also stored in the data section of the program	int global_var;
Register variables	Variables that are stored in the CPU registers instead of memory, this is used to improve performance by reducing memory access time	register int z;

C LANGUAGE: CONSTANT AND LITERAL

In C language, a constant is a variable whose value cannot be modified after it is assigned. Constants are often used to represent values that will not change during the execution of a program, such as the value of pi or the maximum value that a variable can take.

There are several ways to define a constant in C:

- The **const** keyword: A variable can be defined as a constant by using the **const** keyword. For example, **const int MAX_AGE = 100;** declares a constant named MAX_AGE with a value of 100.
- The **#define** preprocessor directive: The **#define** preprocessor directive can be used to define a constant. For example, **#define MAX_AGE 100** defines a constant named MAX_AGE with a value of 100.
- Enumerations: Enumerations (or enums) are a way to define a set of named integer constants. For example, **enum days {SUN, MON, TUE, WED, THU, FRI, SAT};** creates an enumeration type named "days" with named constants SUN, MON, TUE, WED, THU, FRI, and SAT.

A **literal** is a value that is written directly in the source code of a program. Literals are used to initialize variables or to provide values for other parts of the program. C language supports several types of literals:

- Numeric literals: These are numbers written directly in the source code. For example, **50**, **3.14**, **0xFF**.
- Character literals: These are characters written directly in the source code. For example, **'A'**, **'5'**, **'\n'**.
- String literals: These are sequences of characters written directly in the source code. For example, **"Hello, world!"**, **"This is a string"**.
- Boolean literals: These are the constants **true** and **false** which represent the logical values.

That constants and literals are similar but not the same, constants are variables that cannot be modified and literals are values written directly in the source code.

The ‘C’ LANGUAGE CONSTANT WITH EXAMPLES

In the C programming language, a constant is a value that cannot be changed during the execution of the program. Constants are typically used to store fixed values such as pi or to define symbolic names for values that are used throughout the program.

There are two ways to define a constant in C:

Using the #define preprocessor directive:

#define PI 3.14

This creates a constant named PI with a value of 3.14.

Using the const keyword:

Example

const double PI = 3.14;

This creates a constant named PI with a value of 3.14 and a data type of double.

Both ways have the same meaning and usage, but using const makes the constant visible to the compiler, so it can check for the validity of the use of the constant, such as in cases where the constant is assigned a value.

For example:

#define MAX_LENGTH 50

const int MIN_LENGTH = 5;

```
int main()
{
  int array[MAX_LENGTH];
  for(int i=0; i<MIN_LENGTH; i++)
    array[i] = i;
  return 0;
}
```

In this example, MAX_LENGTH and MIN_LENGTH are constants that are used to define the size of an array and a loop. Because they are constants, their values cannot be changed during the execution of the program. It is a best practice to use constants where appropriate because it makes the code more readable, and easier to maintain and modify.

TYPES OF CONSTANTS USED IN C LANGUAGE

In C, constants can be grouped into several categories based on their type and value:

Numeric constants: These are constants that represent numbers, such as integers (e.g. 12, -7) or floating-point numbers (e.g. 3.14, -2.5). They can be represented in decimal, octal or hexadecimal notation, and can have a suffix indicating their type (e.g. 12L for long int, 12U for unsigned int).

Character constants: These are constants that represent single characters, such as letters, digits, or symbols. They are enclosed in single quotes, e.g. 'A', '3', '$'.

String constants: These are constants that represent a sequence of characters, such as a word or a sentence. They are enclosed in double quotes, e.g. "hello", and "goodbye".

Enumeration constants: These are constants that are defined using the enum keyword and are represented by a set of named values. They are used to define a distinct type consisting of a set of named values.

enum color {red, green, blue};

enum color c = red;

Boolean constants: These are constants that have either a true or false value and are used to represent boolean logic. They are not specified with any keywords but true and false are keywords.

bool is_valid = true;

Symbolic constants: These are constants that are defined using the #define preprocessor directive or the const keyword, and are represented by a name that stands for a specific value. They are typically used to define fixed values such as pi or to define symbolic names for values that are used throughout the program.

#define PI 3.14

const double E = 2.71828;

It is important to keep in mind that even though these constants cannot change during runtime, but the value of some constants like those defined using the #define preprocessor directive are replaced by the pre-processor in the source code before it is passed to the compiler.

a table that lists the different types of constants in C and their characteristics:

Type of Constant	**Description**	**Example**
Numeric constants	Constants that represent numbers	12, -7, 3.14, -2.5
Character constants	Constants that represent single characters	'A', '3', '$'
String constants	Constants that represent a sequence of characters	"hello", "goodbye"
Enumeration constants	Constants that are defined using the **enum** keyword and are represented by a set of named values	enum color {red, green, blue};

Boolean constants	Constants that have either a **true** or **false** value and are used to represent boolean logic	true, false
Symbolic constants	Constants that are defined using the **#define** preprocessor directive or the **const** keyword, and are represented by a name that stands for a specific value	**#define PI 3.14**, const double E = 2.71828;

Keep in mind that constants defined with the const keyword are visible to the compiler, and the program can take advantage of this fact, with the compiler doing things like range checking, or type checking on these constants, which is not possible when using #define

C LANGUAGE DATATYPES

In the C programming language, data types are used to define the type of a variable or a value. They tell the compiler the size of the variable and how it will be used in the program.

The following is a table of the most common data types in C, along with their sizes and ranges.

Data Type	**Size (bytes)**	**Range**
char	1	-128 to 127 or 0 to 255
short	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647
long long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	1.2 x 10^-38 to 3.4 x 10^38 (6 to 7 decimal digits)
double	8	2.3 x 10^-308 to 1.7 x 10^308 (15 decimal digits)

It is important to keep in mind that the size of data types may vary depending on the system or platform, so these sizes should be used as a general guide rather than an absolute rule. Also, not all architectures and compilers support all the aforementioned datatype, in particular, 'long long' which is a c99 feature.

int: This data type is used to store integers or whole numbers.
float: This data type is used to store decimal numbers, with a certain precision.
double: Similar to float, but with greater precision.
char: This data type is used to store individual characters, such as letters, numbers, and symbols.
void: This data type is used to represent the absence of a type or value.
In addition to these basic data types, C also has several other data types, including: short, long, and long are used for integers

float and double are used for floating-point numbers.

bool data type to represent boolean values (true or false)

enum to create an enumeration, a distinct type consisting of a set of named values.

struct which can be used to define a custom data type that can hold multiple variables of different types.

typedef which can be used to give a type a new name.

When defining a variable in C, you have to specify its data type and give it a name. For example:

int age;

This creates a variable called "age" that can hold an integer value.

It is important to choose the appropriate data type when declaring a variable, this will ensure that the variable has enough memory to store the values and that the operations on it are the expected ones.

Some Question and Answers

Simple C basic Language: Short Question and Answer

Q: What is a variable in C? A: A variable in C is a named location in memory that can store a value of a specific data type, such as an integer or a floating-point number. Variables are used to store values that can change during the execution of a program.
Q: How do you declare a variable in C? A: Variables in C are declared using the keywords "int", "float", "double", "char" etc, followed by the variable name, and an assignment operator (=) to assign a value to the variable. For example, to declare an integer variable named "x" and assign it the value 5, you would use the following statement: "int x = 5;".
Q: What is a function in C? A: A function in C is a block of code that performs a specific task and can be reused throughout a program. Functions are defined using the keyword "function" followed by the function name, a set of parentheses () that may contain parameters, and a block of code enclosed in curly braces {}. For example, the following is the syntax for a function named "example": "void example() { // code to be executed }".
Q: What is the main function in C? A: The main function in C is the starting point of a C program. It is a special function that is called when the program is executed. The main function is defined using the keyword "int" followed by the function name "main", a set of parentheses (), and a block of code enclosed in curly braces {}. For example, the following is the syntax for the main function: "int main() { // code to be executed }". The execution of the program starts from the main function and ends with the last statement of the main function.
Simple C basic Language: Short Question and Answer
2 / 2
Q: What is the difference between a variable and a constant in C?
A: In C, a variable is a location in memory that can hold a value that can be changed during program execution. A constant, on the other hand, is a value that cannot be changed once it has been assigned. Constants are typically declared using the keyword "const" before the data type, for example: "const int x = 5;". Variables do not have the keyword "const" before the data type, for example: "int x = 5;".
Q: What are the basic data types in C?
A: The basic data types in C are:

1. int (integer) - stores whole numbers (e.g. -1, 0, 1, 2)
2. float (floating point number) - stores decimal numbers (e.g. 3.14, 0.01, -2.5)

3. double (double precision floating point number) - stores decimal numbers with more precision than float
4. char (character) - stores a single character (e.g. 'a, 'b', '0', '$')
5. void - used to indicate that a function does not return a value

Additionally, C also has derived data types such as arrays, pointers, and structures.

1. What is C programming language?
2. Who created C programming language?
3. What are the main features of C programming language?
4. What are the basic data types in C?
5. How do you declare variables in C?
6. What are the basic control structures in C?
7. How do you create functions in C?
8. What is the purpose of the main function in C?
9. How do you input and output data in C?
10. What are pointers in C and how do they work?
11. How do you use arrays in C?
12. How do you use structures in C?
13. What are the basic operators in C?
14. How do you debug C programs?
15. How does C compare to other programming languages?