***Overview of C***

**C programming language** was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.

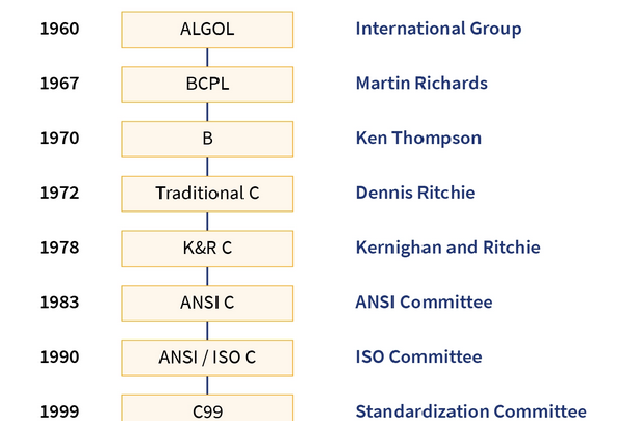
It was developed to overcome the problems of previous languages such as B, BCPL, etc.

It was mainly developed as a system programming language to write an operating system. The main features of the C language include low-level memory access, a simple set of keywords, and a clean style, these features make C language suitable for system programming like an operating system or compiler development.

**Dennis Ritchie** is known as the **founder of the c language**.

***History of c language***

To learn about the history of C language, let's first start with its root and early developments. The root of all modern languages is ALGOL (Algorithmic Language). ALGOL was the first computer programming language to use a block structure, and it was introduced in 1960. In 1967, Martin Richards developed a language called BCPL (Basic Combined Programming Language). BCPL was derived from ALGOL. In 1970, Ken Thompson created a language using BCPL called B. Both BCPL and B programming languages were typeless. After that, C was developed using BCPL and B by Dennis Ritchie at the Bell lab in 1972. So, in terms of history of C language, it was used in mainly academic environments, but at long last with the release of many C compilers for commercial use and the increasing popularity of UNIX, it began to gain extensive support among professionals.

******

***Structure of a C program***

Most programming languages have a structure, including the C language. A 'C' program

is divided into six sections: Documentation, Link, Definition, Global Declaration, main()

Function, Subprograms.

While the main section is compulsory, the rest are optional in the structure of the C

program.

#include <stdio.h>

int main() {

printf("Hello World!");

return 0;

}

* The structure of a C program can be mainly divided into six parts, each having its

purpose. It makes the program easy to read, easy to modify, easy to document, and makes

it consistent in format.

🡪 Example explained

Line 1: #include <stdio.h> is a header file library that lets us work with input and

output functions, such as printf() (used in line 4). Header files add functionality to C

programs.

Line 2: A blank line. C ignores white space. But we use it to make the code more readable.

Line 3: Another thing that always appears in a C program, is main(). This is called a function. Any code inside its

curly brackets {} will be executed.

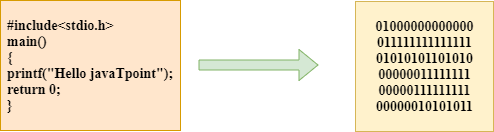
Line 4: printf() is a function used to output/print text to the screen. In our example it will output "Hello World".

Line 5: return 0 ends the main() function.

Line 6: Do not forget to add the closing curly bracket } to actually end the main function.

**What is a compilation?**

The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.



The c compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.

The preprocessor takes the source code as an input, and it removes all the comments from the source code. The preprocessor takes the preprocessor directive and interprets it. For example, if **<stdio.h>,** the directive is available in the program, then the preprocessor interprets the directive and replace this directive with the content of the **'stdio.h'** file.

# printf() and scanf() in C

The printf() and scanf() functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

### ***printf() function***

The **printf() function** is used for output. It prints the given statement to the console.

The syntax of printf() function is given below:

1. printf("format string",argument\_list);
2. The **format string** can be %d (integer), %c (character), %s (string), %f (float) etc.

### ***scanf() function***

The **scanf() function** is used for input. It reads the input data from the console.

1. scanf("format string",argument\_list);

### Program to print cube of given number

Let's see a simple example of c language that gets input from the user and prints the cube of the given number.

1. #include<stdio.h>
2. int main(){
3. int number;
4. printf("enter a number:");
5. scanf("%d",&number);
6. printf("cube of number is:%d ",number\*number\*number);
7. return 0;
8. }

**Output**

enter a number:5

cube of number is:125

The **scanf("%d",&number)** statement reads integer number from the console and stores the given value in number variable.

The **printf("cube of number is:%d ",number\*number\*number)** statement prints the cube of number on the console.

### Program to print sum of 2 numbers

Let's see a simple example of input and output in C language that prints addition of 2 numbers.

1. #include<stdio.h>
2. int main(){
3. int x=0,y=0,result=0;
4. printf("enter first number:");
5. scanf("%d",&x);
6. printf("enter second number:");
7. scanf("%d",&y);
8. result=x+y;
9. printf("sum of 2 numbers:%d ",result);
10. return 0;
11. }

**Output**

enter first number:9

enter second number:9

sum of 2 numbers:18

# Variables in C

A **variable** is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

1. int a;
2. float b;
3. char c;

Here, a, b, c are variables. The int, float, char are the data types.

We can also provide values while declaring the variables as given below:

1. int a=10,b=20;//declaring 2 variable of integer type
2. float f=20.8;
3. char c='A';

## Rules for defining variables

* A variable can have alphabets, digits, and underscore.
* A variable name can start with the alphabet, and underscore only. It can't start with a digit.
* No whitespace is allowed within the variable name.
* A variable name must not be any reserved word or keyword, e.g. int, float, etc.

**Valid variable names:**

1. int a;
2. int \_ab;
3. int a30;

**Invalid variable names:**

1. int 2;
2. int a b;
3. int long;

## Types of Variables in C

There are many types of variables in c:

1. local variable
2. global variable
3. static variable
4. automatic variable
5. external variable

### Local Variable

A variable that is declared inside the function or block is called a local variable.

It must be declared at the start of the block.

1. void function1(){
2. int x=10;//local variable
3. }

You must have to initialize the local variable before it is used.

### Global Variable

A variable that is declared outside the function or block is called a global variable. Any function can change the value of the global variable. It is available to all the functions.

It must be declared at the start of the block.

1. int value=20;//global variable
2. void function1(){
3. int x=10;//local variable
4. }

### Static Variable

A variable that is declared with the static keyword is called static variable.

It retains its value between multiple function calls.

1. void function1(){
2. int x=10;//local variable
3. static int y=10;//static variable
4. x=x+1;
5. y=y+1;
6. printf("%d,%d",x,y);
7. }

If you call this function many times, the **local variable will print the same value** for each function call, e.g, 11,11,11 and so on. But the **static variable will print the incremented value** in each function call, e.g. 11, 12, 13 and so on.

### Automatic Variable

All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using **auto keyword**.

1. void main(){
2. int x=10;//local variable (also automatic)
3. auto int y=20;//automatic variable
4. }

### External Variable

We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use **extern keyword**.

myfile.h

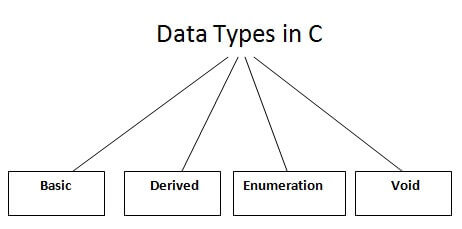
1. extern int x=10;//external variable (also global)

program1.c

1. #include "myfile.h"
2. #include <stdio.h>
3. void printValue(){
4. printf("Global variable: %d", global\_variable);
5. }

# Data Types in C

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.



There are the following data types in C language.

|  |  |
| --- | --- |
| **Types** | **Data Types** |
| Basic Data Type | int, char, float, double |
| Derived Data Type | array, pointer, structure, union |
| Enumeration Data Type | enum |
| Void Data Type | void |

## Basic Data Types

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.

The memory size of the basic data types may change according to 32 or 64-bit operating system.

Let's see the basic data types. Its size is given **according to 32-bit architecture**.

|  |  |  |
| --- | --- | --- |
| **Data Types** | **Memory Size** | **Range** |
| **char** | 1 byte | −128 to 127 |
| signed char | 1 byte | −128 to 127 |
| unsigned char | 1 byte | 0 to 255 |
| **short** | 2 byte | −32,768 to 32,767 |
| signed short | 2 byte | −32,768 to 32,767 |
| unsigned short | 2 byte | 0 to 65,535 |
| **Int** | 4 byte | -2,147,483,648 to 2,147,483,647 |
| signed int | 4 byte | -2,147,483,648 to 2,147,483,647 |
| unsigned int | 2 byte | 0 to 4,294,967,295 |
| **short int** | 2 byte | −32,768 to 32,767 |
| signed short int | 2 byte | −32,768 to 32,767 |
| unsigned short int | 2 byte | 0 to 65,535 |
| **long int** | 8 byte | |  | | --- | | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | |  |  |  | |
| signed long int | 8 byte | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| unsigned long int | 8 byte | 0 to 18,446,744,073,709,551,615 |
| **float** | 4 byte | **1.2E-38 to 3.4E+38** |
| **double** | 8 byte | 2.3E-308 to 1.7E+308 |
| **long double** | 10 byte | 3.4E-4932 to 1.1E+4932 |

## Keywords in C

A keyword is a reserved word. You cannot use it as a variable name, constant name, etc. There are only 32 reserved words (keywords) in the C language.

A list of 32 keywords in the c language is given below:

auto break case char const continue default

do double else enum extern float for

goto if int long register return short signed

sizeof static struct switch typedef union unsigned

volatile while void

We will learn about all the C language keywords later.

# C Identifiers

C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc. An identifier can be composed of letters such as uppercase, lowercase letters, underscore, digits, but the starting letter should be either an alphabet or an underscore. If the identifier is not used in the external linkage, then it is called as an internal identifier. If the identifier is used in the external linkage, then it is called as an external identifier.

We can say that an identifier is a collection of alphanumeric characters that begins either with an alphabetical character or an underscore, which are used to represent various programming elements such as variables, functions, arrays, structures, unions, labels, etc. There are 52 alphabetical characters (uppercase and lowercase), underscore character, and ten numerical digits (0-9) that represent the identifiers. There is a total of 63 alphanumerical characters that represent the identifiers.

## Rules for constructing C identifiers

* The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore.
* It should not begin with any numerical digit.
* In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
* Commas or blank spaces cannot be specified within an identifier.
* Keywords cannot be represented as an identifier.
* The length of the identifiers should not be more than 31 characters.
* Identifiers should be written in such a way that it is meaningful, short, and easy to read.

**Example of valid identifiers**

1. total, sum, average, \_m \_, sum\_1, etc.

**Example of invalid identifiers**

1. 2sum (starts with a numerical digit)
2. int (reserved word)
3. char (reserved word)
4. m+n (special character, i.e., '+')

## Types of identifiers

* Internal identifier
* External identifier

**Internal Identifier**

If the identifier is not used in the external linkage, then it is known as an internal identifier. The internal identifiers can be local variables.

**External Identifier**

If the identifier is used in the external linkage, then it is known as an external identifier. The external identifiers can be function names, global variables.

## Differences between Keyword and Identifier

|  |  |
| --- | --- |
| **Keyword** | **Identifier** |
| Keyword is a pre-defined word. | The identifier is a user-defined word |
| It must be written in a lowercase letter. | It can be written in both lowercase and uppercase letters. |
| Its meaning is pre-defined in the c compiler. | Its meaning is not defined in the c compiler. |
| It is a combination of alphabetical characters. | It is a combination of alphanumeric characters. |
| It does not contain the underscore character. | It can contain the underscore character. |

**Let's understand through an example.**

1. int main()
2. {
3. int a=10;
4. int A=20;
5. printf("Value of a is : %d",a);
6. printf("\nValue of A is :%d",A);
7. return 0;
8. }

**Output**

Value of a is : 10

Value of A is :20

The above output shows that the values of both the variables, 'a' and 'A' are different. Therefore, we conclude that the identifiers are case sensitive.