**UNIT - I CONCEPTS, CONSTANTS, VARIABLES AND DATA TYPE**

**1.1 Concepts of programming methodology : Flowchart Algorithm**

**1.2 Character set**

**1.3 ‘ C’ tokens**

**1.4 Keywords , Identifiers**

**1.5 Constants & Variables**

**1.6 Declaration of variables**

**1.7 Assigning values to variables**

**1.8 Data types**

**1.9 Storage class**

* 1. **Concepts of programming methodology : Flowchart Algorithm**

**Algorithm :**

1] In programming, algorithm are the set of well defined instruction in sequence to solve a

program. An algorithm should always have a clear stopping point.

[2] Qualities of a good algorithm :

Inputs and outputs should be defined precisely.

Each steps in algorithm should be clear and unambiguous.

Algorithm should be most effective among many different ways to solve a problem.

An algorithm shouldn't have computer code. Instead, the algorithm should be written in such a way that, it can be used in similar programming languages.

**An algorithm to add two numbers entered by user.**

Step 1: Start

Step 2: Declare variables num1, num2 and sum.

Step 3: Read values num1 and num2.

Step 4: Add num1 and num2 and assign the result to sum.

sum←num1+num2

Step 5: Display sum

Step 6: Stop

**Flowchart :**

**Flowcharts are diagrams that visually present the process of solving problems.**



**Features of  C Language:**

1. C is a structured programming language with fundamental flow control construction.
2. C is simple and versatile language
3. Programs written in C are efficient and fast.
4. C has only 32 keywords.
5. C is highly portable programming language. The programs written for one computer can be run on another with or without any modifications
6. C has rich set of operators.
7. C permits all data conversions and mixed mode operations
8. Dynamic memory allocation(DMA) is possible in C.
9. Extensive varieties of data types such as arrays, pointers, structures and unions are available in C.
10. C improves by itself. It has several predefine functions.
11. C easily manipulates bits, bytes and addresses.
12. Recursive function calls for algorithmic approach is possible in C.
13. Mainly we are using C language to implement system softwares. These are compilers ,editors, drivers ,databases and operating systems.
14. C compiler combines the capability of an assembly level language with the features of high level language.  So it is called as middle level language.

**Structure of C language :**

Include header file section

Global declaration section

main()

{

Declaration part

Executable part

}

User-defined functions

{

Statements

}

**1.2 Character set**

|  |  |
| --- | --- |
| **Symbol**  | **Meaning**  |
|  ~  |  Tilde |
|  !  | Exclamation mark  |
|  #  | Number sign  |
|  $  | Dollar sign  |
| %  | Percent sign   |
|  ^  | Caret |
|  &  | Ampersand  |
|   \*  | Asterisk  |
| (   | Lest parenthesis  |
|  )  | Right parenthesis  |
| \_  | Underscore   |
|  +  | Plus sign  |
|  |  |  Vertical bar |
|   \  |  Backslash |
|  ` |  Apostrophe |
|  -  |  Minus sign |
|  =  |  Equal to sign |
|   {  |  Left brace |
|   }  |  Right brace |
|  [  |  Left bracket |
| ]  |  Right bracket |
|  : |   Colon |
|  "  |  Quotation mark |
|  ;  |  Semicolon |
|  <  |  Opening angle bracket |
|  >  |  Closing angle bracket |
|   ?  |  Question mark |
|  ,  |  Comma |
|  .  | Period |
|   /  |  Slash |

**1.3 ‘ C’ tokens**

**C Tokens :**



| **No** | **Token Type** | **Example 1** | **Example 2** |
| --- | --- | --- | --- |
| 1 | Keyword | do | while |
| 2 | Constants | number | sum |
| 3 | Identifier | -76 | 89 |
| 4 | String | “HTF” | “PRIT” |
| 5 | Special Symbol | \* | @ |
| 6 | Operators | ++ | / |

The smallest individual elements or units in a program are called as Tokens. C has following tokens.

* Identifiers
* Keywords
* Constants
* Operators
* Special characters

**Identifiers :**

Identifiers in c refer to the name of the variables, functions, arrays,etc. created by the  programmer, using the combination of following characters.

* Alphabets :     A to Z     (or)    a to z
* Digits :     0 to 9
* Underscore :      \_

**Note :**

* . The first character of an identifier must be an alphabet or underscore ,we canot use digit.
* . Default identifier length is 32 characters.

**Keywords :**

Keywords are the words whose meaning has been already explained by the compiler. That means at the time of designing a language, some words are reserved to do  specific tasks. Such words are called as keywords (or) reserved words. All C compilers support 32 keywords.

**Constants :**

Constants define fixed values, that do not change during the execution of a program. C supports the following constants.

* Integer constants
* Character constants
* Real or floating constants
* String constants

Constants are explained [here](http://cprogrammingexpert.com/c-tutorial/c-constants/)

**Operators:**

Operator  is a symbol which performs particular operation. C supports a rich set of operators. C operators can be classified into No of categories. They include arithmetic operators, logical operators, bitwise operators, etc.

Operators are explained briefly [here](http://cprogrammingexpert.com/c-tutorial/c-operators/)

**Special characters:**

All characters other than alphabets and digits  are treated as special characters.

**Eg:**   \* , % , $ , {  ,etc.

**1.4 Keywords , Identifiers**

**Keywords :**

Keywords are the words whose meaning has been already explained by the compiler. That means at the time of designing a language, some words are reserved to do  specific tasks. Such words are called as keywords (or) reserved words. All C compilers support 32 keywords.

| **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 |

 **Identifiers :**

Identifiers in c refer to the name of the variables, functions, arrays,etc. created by the  programmer, using the combination of following characters.

* Alphabets :     A to Z     (or)    a to z
* Digits :     0 to 9
* Underscore :      \_

**Note :**

* . The first character of an identifier must be an alphabet or underscore ,we canot use digit.
* . Default identifier length is 32 characters.

| **No** | **Token Type** | **Example 1** | **Example 2** |
| --- | --- | --- | --- |
| 1 | Keyword | do | while |
| 2 | Constants | number | sum |
| 3 | Identifier | -76 | 89 |
| 4 | String | “HTF” | “PRIT” |
| 5 | Special Symbol | \* | @ |
| 6 | Operators | ++ | / |

| **Token** | **Meaning** |
| --- | --- |
| **Keyword** | A variable is a meaningful name of data storage location in computer memory. When using a variable you refer to memory address of computer |
| **Constant** | Constants are expressions with a fixed value |
| **Identifier** | The term identifier is usually used for variable names |
| **String** | Sequence of characters |
| **Special Symbol** | Symbols other than the Alphabets and Digits and white-spaces |
| **Operators** | A symbol that represent a specific mathematical or non mathematical action |

**1.5 Constants & Variables**

**Variables**

In programming, a variable is a container (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name ([identifier](https://www.programiz.com/c-programming/c-keywords-identifier)). Variable names are just the symbolic representation of a memory location. For example:

int playerScore = 95;

Here, playerScore is a variable of integer type. The variable is assigned value: 95.

The value of a variable can be changed, hence the name 'variable'.

In C programming, you have to declare a variable before you can use it.

### Rules for naming a variable in C

1. A variable name can have letters (both uppercase and lowercase letters), digits and underscore only.
2. The first letter of a variable should be either a letter or an underscore. However, it is discouraged to start variable name with an underscore. It is because variable name that starts with an underscore can conflict with system name and may cause error.
3. There is no rule on how long a variable can be. However, only the first 31 characters of a variable are checked by the compiler. So, the first 31 letters of two variables in a program should be different

**Constants/Literals :**

A constant is a value or an identifier whose value cannot be altered in a program. For example: 1, 2.5, "C programming is easy", etc.

As mentioned, an identifier also can be defined as a constant.

const double PI = 3.14

Here, PI is a constant. Basically what it means is that, PI and 3.14 is same for this program.

**1. Integer constants**

An integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:

decimal constant(base 10)

octal constant(base 8)

hexadecimal constant(base 16)

**For example:**

Decimal constants: 0, -9, 22 etc

Octal constants: 021, 077, 033 etc

Hexadecimal constants: 0x7f, 0x2a, 0x521 etc

### 2. Floating-point constants

A floating point constant is a numeric constant that has either a fractional form or an exponent form. For example:

-2.0

0.0000234

-0.22E-5

**Note:**E-5 = 10

### 3. Character constants

A character constant is a constant which uses single quotation around characters. For example: 'a', 'l', 'm', 'F'

### 4. Escape Sequences

Sometimes, it is necessary to use characters which cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc. In order to use these characters, escape sequence is used.

For example: \n is used for newline. The backslash ( \ ) causes "escape" from the normal way the characters are interpreted by the compiler.

| Escape Sequences |
| --- |
| Escape Sequences | Character |
| \b | Backspace |
| \f | Form feed |
| \n | Newline |
| \r | Return |
| \t | Horizontal tab |
| \v | Vertical tab |
| \\ | Backslash |
| \' | Single quotation mark |
| \" | Double quotation mark |
| \? | Question mark |
| \0 | Null character |

**5. String constants**

String constants are the constants which are enclosed in a pair of double-quote marks. For example:

"good" //string constant

"" //null string constant

" " //string constant of six white space

"x" //string constant having single character.

"Earth is round\n" //prints string with newline

### 6. Enumeration constants

Keyword enum is used to define enumeration types. For example:

enum color {yellow, green, black, white};

Here, color is a variable and yellow, green, black and white are the enumeration constants having value 0, 1, 2 and 3 respectively.

**1.6 Declaration of variables**

**int**

variables can represent negative and positive integer values (whole numbers). There is a limit on the size of value that can be represented, which depends on the number of bytes of storage allocated to an int variable by the computer system and compiler being used. On a PC most compilers allocate two bytes for each int which gives a range of -32768 to +32767. On workstations, four bytes are usually allocated, giving a range of -2147483648 to 2147483647. It is important to note that **integers are represented exactly** in computer memory.

**float**

variables can represent any real numeric value, that is both whole numbers and numbers that require digits after the decimal point. The accuracy and the range of numbers represented is dependent on the computer system. Usually four bytes are allocated for float variables, this gives an accuracy of about six significant figures and a range of about to . It is important to note that **float values are only represented approximately**.

**bool**

variables can only hold the values **true** or **false**. These variables are known as *boolean* variables in honour of George Boole, an Irish mathematician who invented boolean algebra.

**char**

variables represent a single character -- a letter, a digit or a punctuation character. They usually occupy one byte, giving 256 different possible characters. The bit patterns for characters usually conform to the American Standard Code for Information Interchange (ASCII).

Examples of values for such variables are:

int 123 -56 0 5645

float 16.315 -0.67 31.567

char '+' 'A' 'a' '\*' '7'

**A typical set of variable declarations that might appear at the beginning of a program could be as follows:**

int i, j, count;

float sum, product;

char ch;

bool passed\_exam;

which declares integer variables i, j and count, real variables sum and product, a character variable ch, and a boolean variable pass\_exam.

**A variable declaration has the form:**

*type identifier-list*;

*type* specifies the type of the variables being declared. The *identifier-list* is a list of the identifiers of the variables being declared, separated by commas.

Variables may be initialised at the time of declaration by assigning a value to them as in the following example:

int i, j, count = 0;

float sum = 0.0, product;

char ch = '7';

bool passed\_exam = false;

which assigns the value 0 to the integer variable count and the value 0.0 to the real variable sum. The character variable ch is initialised with the character 7. i, j, and product have no initial value specified, so the program should make *no* assumption about their contents.

**1.7 Assigning values to variables**

#include <stdio.h>

int main(void) {

// Declare variable

float pay\_rate;

// Assign Value

pay\_rate = 100.75;

}

The next option is to simply give the pay\_rate a value right off the bat:

float pay\_rate = 100.75;

**1.8 Data types**

Data types in C Language Data types specify how we enter data into our programs and what type of data we enter. C language has some predefined set of data types to handle various kinds of data that we use in our program. These datatypes have different storage capacities. C language supports 2 different type of data types,

Primary data types These are fundamental data types in C namely integer(int), floating(float), character(char) and void.

Derived data types Derived data types are like array, function, stucture, union and pointer

Primary Data Types(Fundamental Data Types)

All C compiler support five type of fundamental data type

1. Integer int 2,768 to 32,768

2. Character char -128 to 127

3. Floating Point float3.4e-38 to 3.4e+38

4. Double Precision Floating Point double 1.7e-308 to 1.7e+308

 5. Void Data Type void(used for function when no value is to be return) Integer

There are five basic data types associated with variables:

**int** - integer: a whole number.

**float** - floating point value: ie a number with a fractional part.

**double** - a double-precision floating point value.

**char** - a single character.

**void** - valueless special purpose

**Derived Data Type**

Those data types which are derived from fundamental data types are called derived data types. There are basically three derived data types .

1. Array: A finit collection of data of same types or homogenous data type.

2. String: An array of character type.

 3. Structure: A collection of related variables of the same or different data types. note:

Details of Array, String and Structure is available separately in this site

**1. /\*...........Array...........\*/**

**2. int roll\_ no[40]; //Array to contain roll number of 40 students**

**3. /\*...........String...........\*/**

**4. char name[20] = "Deepak Kumar"; //string of MAX 20 chars**

 **5. /\*........... Structure...........\*/**

**6. struct employee**

**7. {**

**8. char name[20];**

**9. int age;**

**10. float salary;**

**11. }; // Collection of different data types**

**12. struct employee emp = {"Shakshi", 28, 450000.00};**

**1.9 Storage class**

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify. We have four different storage classes in a C program −

* auto
* register
* static
* extern

The auto Storage Class

The **auto** storage class is the default storage class for all local variables.

{

 int mount;

 auto int month;

}

The example above defines two variables with in the same storage class. 'auto' can only be used within functions, i.e., local variables.

The register Storage Class

The **register** storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

{

 register int miles;

}

The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

The static Storage Class

The **static** storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

In C programming, when **static** is used on a global variable, it causes only one copy of that member to be shared by all the objects of its class.