C PROGRAMMING

- C is a procedural programming language.
- It was initially developed by Dennis Ritchie in the year 1972, at the Bell Telephone Laboratories to develop the UNIX operating system.
- C programming is considered as the base for the other programming languages, that's why it is known as mother language.
- C language is the system programming language because it can be used to do low-level programming.
- It is generally used to create hardware devices, Operating system, drivers, kernels etc.
- The main features of the C language includes low-level memory access, a simple set of keywords & a clean style.
- It is used in lot of scientific computing like AI/ML, weather forecasting etc.
- C/C++ code is very faster than a code written in java, python.

C-TOKENS

- C tokens are the basic building blocks in C lang.
- C token are the smallest individuals units.

● Keywords ● Identifiers ● Constants
● Operators ● Special Symbols

Keyword

Keywords are predefined & reserve words used in programming that have special meanings to compiler.

Eg. int money; Here, int is keyword and money is variable.

Keywords are auto, break, char, continue, default etc.

There are keywords in C language.

Identifiers

Identifirs refers to name given to entities such as variables, functions, structures etc.

Identifiers must be unique, as they are created to give a unique name to an entity to identify it during the execution of program.

Identifires names must be different from keywords.

Eg. int money; Here money & accountbalance are identi-fiers.
double accountbalance;

Constants

A constant is a value that can't be changed in the program.

Eg. Decimal constant → 10, 20, 450 etc.
Character constant → 'a', 'b' etc; Octal constant → 021, 033, 016 etc.

Operators

An operator is simply a symbol that is used to perform operations.

There can be many types of operations like arithmetic, logical, bitwise etc.

Eg. Arithmetic Operators [+, -, *, /]
Relational Operators [< , <= , > , >= , == , !=]
Logical Operators [&& , ||, !]

Special Symbols

In programming language, the special symbols have some special meaning and they can not be used for other purposes.

Eg. [ ], ( ), { }, ; , * , =, # etc.

!	Exclamation mark	_	Underscore	:	colon
#	Number sign	+	plus sign	;	semicolon
%	Percent sign	,	comma	"	Quotation mark
&	Ampersand	/	slash	?	Question mark
*	Asterisk	=	Equal to sign	.	Period.

C-PROGRAM

A C program basically consists of the following parts -

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

Hello, World program

```
#include<stdio.h>
 int main( ) {
 /*first program*/
  printf("Hello , World!\n");
  return 0; }
```

O/P: Hello, World!

Let us take a look at the various parts of the above 'Hello, World!' program -

The first line of the program #include<stdio.h> is a pre-processor command which tells a c compiler to include stdio.h file before going to actual compilation.

The next line int main( ) is the main function where the program execution begins.

The next line /*-------*/ will be ignored by the compiler, so such lines are called comments in the program.

The next line printf(---) is another function which causes the message "Hello, World!" to be displayed on screen.

The next line return 0; terminates the main function & returns the value 0.

DATA TYPE

Data types refers to an extensive system used for declaring variables or functions of different types.

The type of a variable determines how much space it occupies in storage.

Different data types have different ranges to store nos.

Basically data types are of different types -

- Primary Data Type (char , float , Int)
- User defined Data Type (Enum , Typedef)
- Derived Data Type (Pointers , Arrays , structures , Union)

Primary Data Type

- Primary data types are also known as fundamental data types because they are pre-defined in C language.
- Primary data types in C are of 4 types : int , char, float & double.

S. N.	Data Type	Memory (bytes)	Range	Format Speifier
1.	int	4	-2^{31} to $2^{31}-1$	%d
2	char	1	-128 to 127	%c
3	float	4	1.2E-38 to 3.4E+38	%f
4	double	8	2.3E-308 to 1.7E+308	%lf

INTEGER Data Type

- The int data type is used to store integer values.
- It can be signed or un signed.
- It has generally 32 bits (4 bytes).
- By default integer is signed.

Range:

signed int – -2^{N-1} to $2^{N-1}-1$

unsigned int – 0 to $2^{N}-1$

CHARACTER Data Type

- The char data type is used to store the characters.
- Characters must be inside single quotes.
- It is usually 1 byte. Range: -2^{N-1} to $2^{N-1}-1$
- Whenever we enter a character type variable, the character is stored as integer value in the address location.

1 nibble = 4 bits

1 byte = 8 bits

FLOAT Data Type

- The float data type is used to store the floating point numbers.
- The numbers that have a fractional part are called floating point numbers. Range: 1.2E-38 to 3.4E+38
- It has generally 4 bytes.
- These can represent a much larger and wider range of digits as compared to integer data type.

DOUBLE Data Type

- The double data type is used to store floating numbers.
- It occupies twice as much memory as float data type.
- It has generally 8 bytes. Range: 2.3E-308 to 1.7E+308

Eg. int – 10, 20, 30, 100, 1000, 586 etc.

char – 'a', 'D', 'L', 'm', 's', 'x', 'N', 'K', 'o', 'Q', 'R' etc.

float – 0.7, 0.33, 8.21, 6.43, 1000.29, 562.01 etc.

double – 32.334000, 194.4698300 etc.

Enumeration Data Type

Enumeration (enum) is a user defined data type.

- It is mainly used to assign names to integral constants.
- Value assigned must be in range of signed int.

Syntax: enum enumname{ };
enum enumname;

Eg. Enum data type

```
#include <stdio.h>
 int main() {
  enum letter {a,b,d,f};
  enum letter y=d;
 printf("%d", y);
 return 0; }
```

O/P: 2

Conversion Specifier

- The conversion specifier character specifies whether to interpret the corresponding argument as a character, a string, a pointer, an integer or a floating number.

%d	signed int	%ld	long int	%p	Pointer
%u	unsigned int	%lld	long long int	%s	String
%hi	short signed int	%lu	unsigned long int	%x	Hexa decimal
%hu	short unsigned int	%llu	unsigned long long int	%o	Octa integer

Variables

- A variable is a name given to storage area that our programs can manipulate.
- Two main Concepts of variable-
  - Declaration
  - Initialization & Usage
- Variable name starts with lower case or underscore (_).

Eg. variables

```
void main() {
int a;
 a=10;
 a=5; }
```

NOTE Here, value is changing from 10 to 5, value is varied, so it is a variable.

- int a,b,c,d; → declaring multiple variables in single line.
- int a=5; int b=5; → declaring & initializing variables.

Constants

- A constant is a value that can't be changed in the program.
- Types → char, string, integer real valued.
  - String → "GATE", "BOOK", "India is my Country" "Newton Desk"
  - Char → '#', 'g', '&', 'a' etc.

'abc' → X Not a char

' ' → X empty can't be char

a → X Not valid, must be in single quotes.

OPERATORS

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions.

ARITHMETIC OPERATORS

operators	Description
+	Adds two operands.
−	subtracts second operand from first.
*	Multiplies both operands.
/	Divides numerator by de-numr.
%	Modulus operator and remainder of after an integer division.
++	Increment operator increases the integer value by one.
−−	Decrement operator decreases the integer value by one.

Example

```
int main() {
  int a=17 , b=4;
  printf("Sum = %d\n", a+b);
  printf(" Diff. = %d \n", a-b);
  prinf (" Quotient = %d \n", a/b);
  printf(" Remainder = %d \n", a%b);
  printf(" a = %d \n", ++a);
  return 0;
  }
```

O/P:
```
Sum = 21
Diff. = 13
Quotient = 4
Remainder = 1
a = 18
```

RELATIONAL OPERATORS

oprs.	Description
==	Checks if the values of two operands are equal or not. If yes then the condition becomes true.
!=	checks if the values of two operands are equal or not. If No, then the condition becomes true.
>	The value of left operand is greater than the value of right operand.
<	The value of left operand is less than the value of right operand.
>=	The value of left operand is greater than or equal to the value of right operand.
<=	The value of left operand is less than or equal to the value of right operand.

Example

```
int main() {
 int a=12, b=4;
 if (a>b)
 printf("%d isgreaterthan %d\n", a,b);
 if (a==b)
 printf("%d is equal to %d\n",a,b);
 if(a!=b)
 printf("%d is not equal to
          %d \n", a, b);
 return 0;
  }
```

O/P:
```
12 is greater than4.

12 is not equal
  to4.
```

LOGICAL OPERATORS

oprs.	Description
&&	Logical AND→ If both the operands are non zero then conditions becomes true.
||	Logical OR → If any of the two operands is non zero, then cond.^{n} becomes true.
!	Logical NOT → It is used to reverse the logical state of its operand.

NOTE // True→1 ; False→0

- Short circuit incase of '&&'→ if there is a condition anywhere in expression that return false, then rest of the cond.^{n} after that will not evaluated
- Short circuit in case of '||'→ if there is a cond.^{n} anywhere in expression that return true, then rest of the conditions after that will not be evaluated.

BITWISE OPERATORS

oprs.	Description
&	Binary AND→ It copies a bit to the result if it exits in both operands.
|	Binary OR → It copies a bit if it exists in either operands.
^	Binary XOR→ It copies the bit if it is set in only one operand.
~	Binary One's complement→ It has the effect of flipping bits.
<<	Binary Left shift → The value is moved left by the numberof bits specified by right operand.
>>	Binary Right shift → The left operand value is moved right by the no. of bits specified.

TRUTH TABLE

p	q	p&q	p|q	p^q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

NOTE

- In bitwise left/Right shift the trailing and leading pos.^{n} are filled with zeros.
- Left shifting is equivalent to multiplication by 2nd right operand.
- Right shifting is equivalent to division by 2nd right operand.

ASSIGNMENT OPERATORS

oprs.	Description	oprs.	Description
=	Assignment op^r → Assigns value from right to left operand.	+=	ADD & assignment op^r→ It adds right operands & assign to left operand.
-=	Subtract and assignment op^r → It substracts the right operand and assign to left operand.	*=	Multiply and assignment op^r→ It multiplies the rights operand & assign to left.

/=	Divide and Assignment opr.→ It divides the left operand with the right operand & assigns to left operand.	%=	Modulus and Assignment opr.→ It takes modulus using two operands & assigns the result to left operand.
<<=	Left shift and assignment op.	>>=	Right shift and assignment opr.
&=	Bitwise and assignment opr.	^=	Bitwise EX-OR & assignment opr.
|=	Bitwise OR & assignment opr.		

MISCELLANEOUS OPERATOR

sizeof()	Returns the size of variable.	&	Returns the address of variable.
*	Pointer to a variable.	?:	Conditional expression.

Size of OPERATOR [size of ()]

- sizeof() operator computes the size of variable (byte).
- It is a unary operator.
- For float values use like this i.e. 10.2f otherwise it will be treated as double (10.2) or use typecasting for float and double.
- For character data type i.e. char values use typecasting or directly use datatype or variable.
- Operands can be variable, constant data type.

Eg. size of (x); , sizeof (int); , sizeof (5);

TYPE CASTING

- It refers to changing an variable of one data type into another.
- The compiler will automatically change one type of data into another if it make sense.
- Types → IMPLICIT TYPE

• When the type conversion is performed automatically by the compiler without programmer's intervention such type of conversion is known as implicit type conversion or type promotion.

Eg.

```
int x;
for (x=97; x<=122; x++)
{
 printf ("%c," x);
}
```

ONLY CONVERT
int → float
float → double
char → int

EXPLICIT TYPE

• The type conversion performed by the programmer by posing the data type of expression of specific type is known as explicit type conversion.

Eg.

```
int x;
for (x=97; x<=122; x++)
{
printf ("%c", (char) x);
}
```