Introduction to C

UNIT-1

Features of C and its Basic Structure, &
Simple C programs.
Constants, Integers.
Integer Constants.
Real Constants.
Character Constants.
String constants.
Backslash Character Constants.
Concept of an Integer and a variable.
Rules for naming Variables and
Assigning Values to Variables.

- C programming language was developed by Dennis Ritchie in 1972.
- We will be learning C99 standard (ISO C)

CONSTANTS

A constant is an entity that doesn't change
They are often called as literals.

TYPES OF CONSTANTS

PRIMARY
- Integer Constants
- Variable Constants
- Character Constants

SECONDARY
- Pointer
- Array
- String
- Structure
- Union
- Enum

PRIMARY CONSTANTS

1. Integer Constants

A integer constant is a value a decimal, octal or Hexadecimal number that represents an integral value.

RDC [Integer Constants]

- Must contain at least 1 digit.
- must not have a decimal point.
- can be 0 +ive or -ive.
- Commas or blanks are not allowed.
- Range :→ -2147483648 to $+2147483647$

Ex → 426, +782, -8000, -7605

2. Real Constants

A constant with the combination of +ive, 0, +ive followed by a decimal point and the fractional part

ROC a Real Constant

- A real constant must contain at least one digit
- must contain a decimal point
- can be +ive, -ive or zero
- commas or blanks are not allowed

Ex → +325.54, 426.0, -3276, -48.5792

3 Character Constant

A character constant is a single alphabet, digit or special symbol enclosed within single inverted commas.

ROC a character constant

- must have one alphabet, number or special symbol.
- should be enclosed in single inverted commas.

Ex → 'A', 'I', '5', '='

→

SECONDARY CONSTANTS

1. STRING CONSTANTS:
A string constant is a sequence of a sequencing of characters enclosed in double quotes

ROC of a String Constant

- A string constant may consist of any number of alphabets, numbers or special characters.
- It must be enclosed in double quotes.
-

Difference b/w a character constant and string constant.

Character Constant	String Constant
A single character string has an equivalent integer value.	A single character string constant does not have an equivalent integer value.
enclosed within single inverted commas.	enclosed within double inverted commas.
Max length of a char constant can be one character	A string constant can be of any length.
A single character constant occupies one byte.	A single string constant occupies two bytes. "A" & \0.
Ex → 'A', '5', '3'	"she is 10 years old." "A5" "361=1"

Backslash Character Constants

There are some types of characters that have a special type of meaning in the C language. These must be preceded by a backslash symbol so that the program can use the special function in them.

Meaning of Character	Backslash Character
Backspace	\b
New line	\n
Form feed	\f
Horizontal tab	\t
Carriage return	\r
Single quote	\'
Double quotes	\"
Vertical tab	\v
Backslash	\\
Question mark	\?
Alert or Bell	\a
Hexadecimal constant	\XN
Octal constant	\N

Variables

A variable is a name assigned to a memory space that may be used to store a data value.

Rules for naming variables

#Rule No. 1

A variable name may include alphabets, numbers and '_'. Special characters are not allowed.

#Rule No. 2

A variable name must not begin with a digit.

#Rule No. 3

A variable name may begin with a underscore, but try not to.

#Rule No. 4

C language in case sensitive.

So, Name, NAME, NaMe, nAME ; all are different.

#Rule No. 5

Blanks or white spaces are not allowed.

Do not use long names for a variable.

#Rule No. 6

Don't use keywords to name your variable like print, switch, if, while, elf .etc.

Assigning value to a variable

To assign a value to a variable, we use an assignment expression.

Assignment expressions assign a value to the left operand.

Assignment Operators

Simple (=)

gives the value of the right operand to the left operand.

Ex → name = "Shreya";
var = 5;

Compound (+=, -=, *=, /=)

performs an operation on both operands and give the result of that operation to the left operand.

Ex → inde += 2

Types of compound assignment operators:

1.) += First addition then assignment.
2.) -= First substraction then assignment.
3.) *= First multiplication then assignment.
4.) /= First division then assignment.
5.) %= First modulus then assignment.
6.) <<= First bitwise left shift then assignment.
7.) >>= First bitwise right shift then assignment.
8.) &= First bitwise AND then assignment.
9.) |= First bitwise OR then assignment.
10.) ^= First bitwise XOR then assignment.

```
int main() {
    var = 5;
    var += 2;

    printf("%d", var);
    return 0;
}
```

ANS → 7.

Scope of Variables

Scope: a block or a region where a variable is declared, defined and used and when a block or area ends, variable is automatically destroyed.

Types of Variables

Local

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

Syntax

```
main() {
    void fun() {
        int a = 10
        pf("%d", a)
    }
```

Global

A variable that is declared outside a function or a block. It is accessible to all the functions.

Syntax of local variable

```
main() {
    void fun() {
        int a = 5;
        printf("%d", a)
    }
    printf("%d", a)
}
```

Since a is inside the region of the void fun() { function, the second printf statement cannot access it. So this will produce an error.

a = 5 → local variable accessible only to void fun() function.

Operators

TYPES OF OPERATORS

- Arithmetic Operators
- Unary Operators
- Relational and logical Operators
- Bitwise Operators
- Increment Decrement
- Conditional Operator
- Assignment operators

1. ARITHMETIC OPERATORS

i) + Addition
ii) − Subtraction
iii) * Multiplication
iv) / Division
v) % Modulus (Remainder)

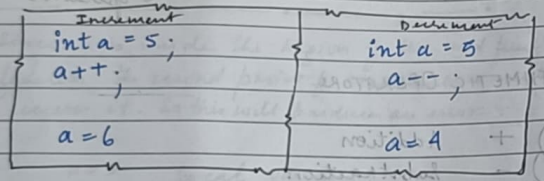
OPERATOR PRECEDENCE & ASSOCIATIVITY

Precedence	Operator	Associativity
Highest	+, −, *, /, %	Left to right
Lowest	+, −	Left to right.

2. Increment Operator and decrement Operators

Increment Operator: used to increment the value of a variable by 1.

Decrement Operator: used to decrement the value of a variable by 1.



3. Relational and Logical Operators

Relational Operators: Used for comparing values.

- They all return True or False.

i) == equal to
ii) != not equal to
iii) <= less than or equal to
iv) >= greater than or equal to
v) < less than
vi) > greater than

```
int a = 300, b = 2090;

if (a >= b)
  { printf ("Bingo!"); }

else
  { printf ("Alas!"); }
```

NOTE: since (a) is smaller than (b)
so, else statement will be printed.

→ Alas!

Logical Operators: evaluates the conditions and returns a result.

i) && AND → returns TRUE only when all cond. are T.
ii) || OR → returns TRUE if any one cond. is True.
iii) ! NOT → returns True whe cond. is False and vice versa

4. Bitwise Operators

Performs bitwise manipulation.

i) & AND
ii) | OR
iii) ~ NOT
iv) << Left Shift
v) >> Right Shift
vi) ^ XOR

i) & AND : It takes two bits at a time and performs AND operation.

Ex → 0111 → 7
0100 4
0100 4

7 & 4 = 4

ii) | OR : It takes two bits at a time and performs OR operation.

Ex → 0111 → 7
0100 → 4
0111 → 7

7 | 4 = 7

iii) ~ NOT : Unary operator: complements each bit one by one.

Ex → 0111
→ 1000

~7 → 8

iv) Left shift << : It shifts the bit one by one to the left and trailing positions are filled with zeroes.

```
char var = 3;
var << 1;
```

0011 (3)
→ 0110 (6)

→ 6

→ n x 2^degree of shift

Ex → var: 60
var << 4
→ 60 x 2^4 = 60 x 16 = 960

v) Right shift >> : It shifts the bits one by one to the right and trailing leading positions are filled with zeroes.

```
char var = 3;
var >> 1;
```

0011 (3)
0001 (1)

→ 1

n / 2^degree of shift

Ex → var = 32
var >> 4
→ 32 / 2^4 = 32 / 16 = 2

vi) ^ XOR → When both are 1 or 0 then the result is 0. When both are differnt then the result is zero.
Bitwise XOR operator perform XOR functions bit by bit.

```
0111  → 7
0100  → 4
0011  → 3
```

7 ^ 4 → 3

5. Conditional Operator [?]

→ takes three operands
→ condition? expression1 : Expression2;

```
char result;
int marks = 50

result = (marks > 33) ? 'p' : 'f';
```

→ p

6 COMMA OPERATOR (,)

→ can be used as a separator

int a = 3, b = 4, c = 8;

→ comma operator returns the right most operand in the expression and it simply evaluates the rest of the operands and finally rejects them.

```
int var = (3, 4, 8);
printf("%d", a);
```

→ 8

* comma operator evaluates all operands but assign the value to the var variable of the last or rightmost operand.

```
int var = (printf("%d %d %s", "Hello!"), 5);
printf("%d", var);
```

→ Hello!
5

7. sizeof() determines the no. of bytes required to store the data.

→ int x;
sizeof x ; 4
Sizeof 5 : 2
Size of char: 1
Size of float: 4
Size of double; 8
Size of long double; 10 - 12

Operators in C based on no. of operands

Unary (1 operand)
- Unary minus -
- Increment ++
- Decrement --
- Logical Not !
- sizeof()

ternary (3 operands)
conditional operator (?)

Binary (2 operands)
- Assignment
- Arithmetic
- Relational
- Bitwise operator

TYPES OF DATA TYPES

1) Char

char variable_name = 'N';
range - Signed → -128 to +127
Unsigned 0 to 255

size of (char) = 1 byte

%c

2) float

→ float variable_name = value;

printf("%.16f\n", variable_name)

float can print correct values till 6 decimal points.

size of (float) → 4 bytes

%f

3) double :

double variable_name = value;

double → 16 digits → size of (double) → 8
long double → 19 digits size of (long double) → 12

int v;

int var = 4/9

var = 0

size of (int) = 2 bytes

%d

INTEGER & FLOAT CONVERSIONS

1. int ± */ int = int
2. float ± */ float = float
3. int ± */ float = float
   - if int is promoted to float then the operation is performed.

```
float a, b, c; int s;
s = a*b*c/100 + 32/4 - 3*1.1;
```

Here the expression has various float and int terms. So, first the int terms will be promoted to float. Then the operations will be performed.
Then the result will be converted to int since variable assigned to the result is of int data type.

MIXED MODE EXPRESSION

Such an expression in which the two operands are not of the same data type is called a mixed mode expression.

TYPE CASTING

Converting of a data type of one variable to some other data type.

1) Implicit casting → done by compiler
2) Explicit casting → done by programmer.

Explicit castings.

Case I

```
int a = 10, b = 3;
float c = a/b;
```

Output → 10/3 → 3

cast type operator

Case II

```
int a = 10, b = 3;
float c = (float) a/b;
```

(float) parenthesis before operand expression

Output → 10/3 → 3.33
10.0/3.0

Case III

```
int a = 10, b = 3;
float c = (float) (a/b);
```

Type Cast Operator

The type cast operator converts the data type of the expression to the type specified by the type-name.

KEYWORDS

The keywords are words whose meaning has already been explained to the C compiler. 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

Character Input & Output

getchar()

To read a single character as input we use [getchar()] function.

```
char ch;
printf("Enter a character.");
ch = getchar();
printf("Entered character: %c\n", ch);
return 0.
```

Output → Enter any character: A
Entered character: A

putchar()

To output a single character we use the [putchar()] function.

```
char ch;
printf("Enter any character.");
ch = getchar()
printf("Entered character: ");
putchar(ch);
return 0;
```

Output → Enter any character: A
Entered character: A

Formatted Input and Output

1) printf() used for displaying a single or multiple values in the form of output for the user end at the console.

2) scanf() : this is used for reading a single or multiple values in the form of input from the user end at the console.

3) sprintf() :

Scanf() - Syntax

scanf("%d" & variable name)

int var;

scanf("%d" & var);

Format Specifiers

\t	tab space
\n	new line
%d	signed integer
%c	character value
%f	floating point value of a float
%lf	floating point value of a double
%Lf	" " long double
%s	string

gets() : used for reading a string from a user.

puts() : prints a string taken from the user input on the console.

i) & AND : It takes two bits at a time and performs AND operation.

Ex → 0111 → 7
0100 4
0100 4

7 & 4 = 4

ii) | OR : It takes two bits at a time and performs OR operation.

Ex → 0111 → 7
0100 → 4
0111 → 7

7 | 4 = 7

iii) ~ NOT : Unary operator: complements each bit one by one

Ex → 0111
→ 1000

~7 → 8

iv) Left shift << : It shifts the bit one by one to the left and trailing positions are filled with zeroes.

```
char var = 3;
var << 1;
```

→ 6

0011 (3)
→ 0110 (6)

→ n x 2^degree of shift

Ex → var: 60
var << 4
→ 60 x 2^4 = 60 x 16 = 960

v) Right shift >> : If shifts the bits one by one to the right and trailing leading positions are filled with zeroes.

```
char var = 3;
var >> 1;
```

→ 1

0011 (3)
0001 (1)

n / 2^degree of shift

Ex → var = 32
var >> 4
→ 32 / 2^4 = 32 / 16 = 2

vi) ^ XOR → When both are 1 or 0 then the result is 0. When both are differ then the result is zero.
Bitwise XOR operator performs XOR function bit by bit.

```
0111  → 7
0100  → 4
0011  → 3
```

7 ^ 4 → 3

5. Conditional Operator [?]

→ takes three operands
→ condition? expression1 : Expression2;

```
char result;
int marks = 50

result = (marks > 33) ? 'p' : 'f';
```

→ p

6 COMMA OPERATOR (,)

→ can be used as a separator

int a = 3, b = 4, c = 8;

→ comma operator returns the right most operand in the expression and it simply evaluates the rest of the operands and finally rejects them.

```
int var = (3, 4, 8);
printf("%d", a);
```

→ 8

* comma operator evaluates all operands but assigns the value to the var variable of the last or rightmost operand.

```
int var = (printf("%d %d\n", "Hello!"), 5);
printf("%d", var);
```

→ Hello!
5

7. sizeof() determines the no. of bytes required to store the ~~bt~~ data.

→ int x;
sizeof x ; 2
Size of 5 ; 2
Size of char ; 1
Size of float ; 4
Size of double ; 8
Size of long double ; 10 - 12

Operators in C
based on no. of operands

Unary (1 operand)
- Unary minus -
- Increment ++
- Decrement --
- Logical Not !
- sizeof()

ternary (3 operands)
conditional operator (?)

Binary (2 o/a
- Assignments
- Arithmetic
- Relational
- Bitwise ope.

TYPES OF DATA TYPES

) Char

char variable_name = 'N';
range → Signed → -128 to +127
Unsigned → 0 to 255

size of (char) = 1 byte

%c

2) float

→ float variable_name = 'value';

printf("%.16f\n", variable_name)

float can print correct values till 6 decimal points.

size of (float) → 4 bytes

%f

) double :

double variable_name = 'value'

double → 16 digits → size of (double) → 8
long double → 19 digits → size of (long double) → 12

int v;

int var = 4/9

var = 0

sizeof(int) = 2 bytes

%d

INTEGER & FLOAT CONVERSIONS

1. int ± */ int = int
2. float ± */ float = float
3. int ± */ float = float
   ↳ int is promoted to float then the operation is performed.

```
float a, b, c; int s;
s = a*b*c/100 + 32/4 - 3*1.1;
```

Here the expression has various float and int terms. So, first the int terms will be promoted to float. Then the operations will be performed.
Then the result will be converted to int since variable assigned to the result is of int data type.

MIXED MODE EXPRESSION

Such an expression in which the two operands are not of the same data type is called a mixed mode expression.

TYPE CASTING

Converting a data type of one variable to some other data type.

1) Implicit casting → done by compiler
2) Explicit casting → done by programmer.

Explicit castings.

Case I

```
int a = 10, b = 3;
float c = a/b;
```

Output → 10/3 → 3

cast type operator

Case II

```
int a = 10, b = 3;
float c = (float) a/b;
```

(float) — parenthesis before algebraic expression

Output → 10/3 → 3.33
10.0/3.0

Case III

```
int a = 10, b = 3;
float c = (float)(a/b);
```

Output → 10/3 → 3 → 3.0

Type Cast Operator

The type cast operator converts the data type of the expression to the type specified by the type-name.

KEYWORDS

The keywords are words whose meaning has already been explained to the C compiler. 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

Character Input & Output

getchar()

To read a single character as input we use [getchar()] function.

```
char ch;
printf("Enter a character.");
ch = getchar();
printf("Entered character: %c\n", ch);
return 0.
```

Output → Enter any character: A
Entered character: A

putchar()

To output a single character we use the [putchar()] function.

```
char ch;
printf("Enter any character.");
ch = getchar()
printf("Entered character: ");
putchar(ch);
return 0;
```

Output → Enter any character: A
Entered character: A

Formatted Input and Output

1) printf() used for displaying a single or multiple values in the form of output for the user end at the console.

2) scanf() : this is used for reading a single or multiple values in the form of input from the user end at the console.

3) sprintf():

---

scanf() - syntax

```
scanf("%d" & variable name)

int var;

scanf("%d" & var);
```

Format Specifiers

\t	tab space
\n	new line
%d	signed integer
%c	character value
%f	floating point value of a float
%lf	floating point value of a double
%Lf	" " long double
%s	string

gets() : used for reading a string from a user.
puts() : prints a string taken from the user input on the console.

If - Else Statements

Syntax

```
if (condition)
{
    task 1
    task 2
    .
    .
}
else
{
    task 1
    task 2
    .
    .
}
```

to give more options than 2

```
int x = 0;
if (x == 0)
    pr- - - -
else if (cond. 2)
    pr- - - -
else if (cond. 3)
    pr- - - -
else
    pr- - - - -
```

SWITCH

A control instruction that allows us to make a decision from the number of choices.

Syntax

```
switch(x)
{
    case 1: printf(" . . . ");
            break;
    case 2: printf(" . . . ");
            break;
    case 3: printf(" . . . ");
            break;
    default: printf(" . . . ");
             break;
}
```

For Loops

Syntax

```
for(intialization; condition; increment/decrement)
{
    statement;
}
```

Example :

```
for (i=3; i>0; i--)
{
    print(i);
}
```

While loops

Syntax

```
while(expression)
{ statement;
  statement;
    .
    .
}
```

Example

```
int i=3
while (i>0)
{
    print(1);
    i--;
}
```

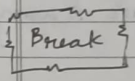
Do while loops

Syntax

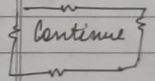
```
do
{
    print(" .... ");
    i--;
} while(i>0);
```

The difference between while and do while

We use do while, we want the body of the condition to execute once first, then check the condition and then start a loop.



Used to terminate from the loop.



It forces to execute the next iteration of the loop.

Functions

Function is basically a set of statements that takes inputs, perform some computation and produces output.

Syntax

```
Return_type function_name(set_of_inputs);
```

Why do we need functions?

1. Reusability: Once the function is defined, it can be reused over and over again.
2. Abstraction: If you are just using the functions in your program then you don't have to worry about how it works inside!

Function Prototype: declaring the properties of a function to the compiler.

Eg: `int fun(int, char);`

Properties of a Function

- Name of the function: 'fun'
- Return type of function: 'int'
- Number of parameters: 2
- Types of parameter 1 & 2: int and char