KSU CET

S1 & S2 Notes

2019 Scheme

Manifest the leadership in you

DIVYA CHRISTOPHER

Module 2

History of C

C was developed in the 1970s by Dennis Ritchie at Bell Telephone Laboratories.

C was used at Bell Laboratories until 1978, when Brian Kernighan and Dennis Ritchie published a description of the C language.

ANSI (American National Standards Institute) committee has standardized C language in 1999.

Now, all compilers and interpreters adhere to ANSI standard.

C Character Set -

C uses upper case letters A to Z, lower case letters a to z, digits 0 to 9, white spaces and certain special characters as building

blocks to form basic program elements.

C is a case-sensitive language.

The small individual units in a program are called tokens.

Tokens include :

(a) Identifiers

(b) Keywords

(c) Constants

(d) Datatypes

(e) Variables

(f) Operators

Terminators in C

;	Statement termination
[]	Array termination
()	Function termination
{}	Scope of a block of statements represented by curly braces
//	Single comment statement
/*... ...*/	A Group of comment statements

(a) Identifiers -

A valid identifier is a sequence of 1 or more letters, digits and underscore (_) character.

Identifiers refers to names of variables, functions and arrays.

Rules for naming identifiers are as follows:-

- Space character is not allowed.
- Every identifier should begin with a letter.
- Name cannot start with a digit.
- Upper case and Lower case letters are distinct, since C is a case-sensitive language.
- A Keyword cannot be used as a variable name.

In ANSI C only first 32 characters in a name are recognized.

(b) Keywords -

They are reserved words that has a particular predefined meaning assigned by the compiler designer of a programming language.

Eg:- void int float for case
main char double switch goto

* All Keywords are lowers-case letters.

(c) Constants –

They refer to fixed values that do not change its value during program execution.

(i) Integer constants –

It is an integer-valued number, consisting of a sequence of digits.

Three different number systems are used:

- decimal (base 10)
- octal (base 8)
- hexadecimal (base 16)

A decimal integer constant consists of a combination of digits from 0 to 9.

Eg:- 32767, 5280.

An octal integer constant consists of any combination of digits from 0 to 7.

The first digit must be a zero.

Eg:- 0743, 077777.

Octal numbers take up values from 0 to 7.

Decimal digit	Binary value
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

A hexadecimal integer constant begins with 0X. It consists of a combination of digits from 0 to 9 and A through F (upper-case or lowercase) They represent quantities 10 through 15, respectively.

Eg:- 0X7FFF, 0xabcd.

An unsigned integer constant is appended by a letter u at the end of constant. Long integer constant is appended by a letter L at the end of the constant. Eg:-1234567891

An unsigned long integer is specified by appending the letters UL at the end of constant. Eg:- 12345678uL.

Hexadecimal numbers take up values from 0 to 15.

Decimal digit	Binary value	Decimal digit	Binary value	Decimal digit	Binary value
0	0000	5	0101	10 (A)	1010
1	0001	6	0110	11 (B)	1011
2	0010	7	0111	12 (C)	1100
3	0011	8	1000	13 (D)	1101
4	0100	9	1001	14 (E)	1110
				15 (F)	1111

(ii) Floating-point constants -

It is a Base-ten number that contains an integer part, decimal point, fractional part and exponential part.

Eg:- +5.67, 12.7

3.14159 e-20

const float PI = 3.14;

Syntax of constant variable

const datatype variable = value;

(iii) Characters constants -

These are values / escape sequences / plain characters represented in single quotes, ' '.

'A' Ascii Value 65

'$' ASCII Value 36

Escape sequences are non-printable characters comprising of backslash (\) followed by other character.

Backslash causes an Escape from the normal way, characters are interpreted by the compiler. They are special strings used to control the output seen on

the monitor or any output device.

\zero or \0 - null character

\n - Neco line Character

\r - carriage Return

\b - Backspace

\t - Horizontal tab

\v - Vertical tab

C could hold $2^7 = 128$ different characters, where each character has a 7-bit ASCII code.

Most of the computers make use of ASCII character set, where each individual character is encoded with its own 7-bit ASCII combination.

(iv) String constants –

"LOURDES MATHA"

These are values, plain characters and escape sequences represented in " " double quotes.

(d) Data Types –

It enables storage of different types of values for better manipulation of data items.

C++ data items are classified into 3 categories:-

(a) Built-in datatype

(i) Integral type

- int Integer
- char character

(ii) Floating type

- float Floating point
- double Double Precision

(iii) void

(b) User - defined datatype

- Structure
- Union
- Enumeration

(c) Derived datatype

- Array
- Function
- Pointer
- Reference

Data types	Specifiers	Memory Requirement	Range of values
int	signed int	2 bytes = 16 bits 1 bit - sign Remaining 15 bits - number	-2^{15} to $+2^{15}-1$ (-32768 to +32767)
	unsigned int	16 bits	0 to $+2^{16}-1$ (0 to 65535)
	short int	1 byte = 8 bits	-2^{7} to $+2^{7}-1$ -128 to +127 (signed) 0 to $2^{8}-1$ 0 to 255 (unsigned)

long int	4 bytes = 32 bits	-2^{31} to $+2^{31}-1$ (signed) 0 to $2^{32}-1$ (unsigned)
char	1 byte = 8 bits	-2^{7} to $+2^{7}-1$ -128 to +127 (signed) 0 to $2^{8}-1$ 0 to 255 (unsigned)
float	4 bytes = 32 bits (1 word)	3.4 e +/- 38 (7 digits precision)
double	8 bytes = 64 bits (2 words)	1.7 e +/- 308 (15 digits precision)

If memory organization is n bits, then an unsigned number will take a range of values from 0 to $2^n - 1$ and signed number coill take a range of values from -2^{n-1} to $+2^{n-1}-1$, where 1 bit is used to store sign and remaining n-1 bits for number.

(e) Variables - It is an identifier used to represent a single data item.

They are temporary place holders for values in a program, by providing a storage facility for better manipulation of data items. All variables must be declared, before they can appear in executable statements.

Every expression statement has an assigment operator (=) to assign R.H.S expression to L.H.S identifier.

A variable is a valid identifier used for naming and declaring arrays, functions and pointers. All variable declarations should end with a semicolon (;)

Syntax for variable declaration -

datatype variablename;

or

datatype variablename = value;

The datatypes can be int, char, float, double.

Eg: int a = 3, b = 4; // Variables storing // numerical quantity.

int z = a + b; // variables storing // expressions.

The value of a+b is assigned to z using assignment operator '='.

Eg:- Integer variable declaration

```
int a;
```

Character variable declaration

```
char c;
```

Floating-point variable declaration

```
float f;
```

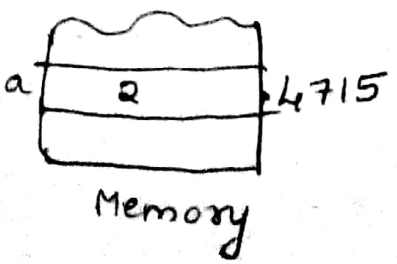
Double-precision variable declaration

```
double d;
```

Semicolon ; indicates statement termination in C++.

Compound statements are a group of statements enclosed in curly braces. { }

```
int a = 2; // This initialization associates a with a value 2 by storing it in physical address 4715.
char c = 'x';
float f = 1.2;
double d = 3.14159 e-20;
```



(ii) Expressions –

An expression represents a single data item such as a number or character.

An expression may consist of a single entity, a constant, a variable, an array element or a reference to a function. An expression is a meaningful combination of operands and operators.

These entities are interconnected by one or more operators.

In C, the conditions true and false are represented by the integer values 1 and 0 respectively.

Egs of few expressions are given below -

$a + b$	Sum of variables a and b
$x = y$	Assigning value of y (variable) to x (variable).
$c = a+b$	Assigning the value of a+b (expression) to the variable c.
$x <= y$	x is less than or equal to y Condition True indicates 1 & False indicates 0.
$x == y$	The value of x is equal to value of y. Expression checks for Equality.
$++i$	Unary operator for incrementing the value of i. i.e Prefix Increment operator
$d = a+b+c$	The value of expression a+b+c is assigned to d, using the assignment operator '='

(iii) Statements -

A Statement causes the computer to carry out some action. There are 3 different classes of statements in C.

(a) Expression statement -

This statement consists of an expression followed by semicolon (;). Every expression statement contains 1 or more operands and operators signifying a meaning to the statement. The execution of an expression statement causes the expression to be evaluated.

Eg:- c = a+b;

(b) A compound statement consists of several individual statements enclosed within a pair of braces, { }.

(c) Control statements are used to create special program features like logical tests, loops and branches. Control statements controls the flow of execution in a program.

(v) Symbolic constants –

It is a name that substitutes a sequence of characters.

Characters may represent a numeric constant, character constant or a string constant.

#define directive is used to define macros. When a program is compiled, each occurence of a symbolic constant is replaced by its corresponding character sequence.

A symbolic constant is defined by writing,

#define symbolic name constant

where #define is a C preprocessor directive. Symbolic constants does not end with a semicolon (;)

#define PI 3.141593

#define TRUE 1

#define FALSE 0

Eg of a C Program using Symbolic constants

```
#include <stdio.h>
#include <conio.h>
 #define PI 3.14
void main()
 {
  float area, radius;
printf("Enter the radius of a circle");
scanf("%f", &radius);
area = PI * radius * radius;
printf("Area of circle = %d", area);
getch();
}
```

Suppose a program contains the statement,

```
area = PI * radius * radius;
```

During compilation process, each occurrence of a symbolic name is replaced by its corresponding constant.
∴ Each occurence of PI is replaced with the value 3.141593.

(f) Operators in C programming language -

The delimiters in C are the following.

//	Double Slash	Single Comment statement
/* ::::*/		Commenting a group of statements
;	Semicolon	Statement Termination
()	Parantheses	Function Termination
[]	Subscript	Array Termination
{ }	Curly braces	Scope of a block of statements
#	Pound sign (Hash sign)	Preprocessor directive / Header File
,	Comma Operator	Variable Separator (used for left → Right evaluation of comma separated expressions)

Basic operators in C -

1. Unary operators
2. Arithmetic operators
3. Relational operators
4. Logical operators
5. Bitwise operators
6. Conditional operators
7. Assignment operators

① Unary operators -

These are operators that acts upon a single operand to produce a new value.

(i) ++ Increment operator -

Increment operator is equivalent to A=A+1, which increments the value of operand by 1.

There are two classifications -

(a) Prefix Increment operator

++A

It first increments the value of A and then only prints the value of A.

(b) Postfix increment operator

A++

It first prints the value of A, and then only increments the value of A.

(ii) Decrement operator --

Decrement operator is equivalent to A=A-1, which decrements the value of operand by 1. There are two classifications:-

(a) Prefix decrement operator

--A

It first decrements the value of A and then only prints the value of A.

(b) Postfix decrement operator

A--

It first prints the value of A and then only decrements the value of A.

(iii) Unary + (Unary Plus Operator)

(iv) Unary - (Unary Minus operator)

Unary - operator preceeds a single operand.

(v) 1's complement operator ~

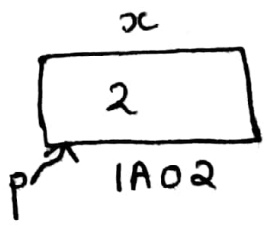
(vi) Logical NOT operator !

(vii) size of (type-name) returns the size of the operand in bytes.

(viii) Address of operator (&)

Used to return address of a variable.

```
int x=2;
int *p;
p=&x;
```



only pointer variables can hold address values. Address of x is 1A02

i.e p is 1A02.

(ix) Value at operator (*)

The value at pointer p can be retrieved as, * p (value at p)

which will return the value 2.

(X) Type cast operator -

C permits explicit type conversion of variables or expressions using type cast operator,

(type).

C supports implicit casting as well.

i.e If i is an integer variable, then the expression i = 3.3 will evaluate to the integer value 3. (Implicit Type Casting)

Syntax of explicit type casting in C -

(type) expression ; // C notation

Eg:- average = sum / float(i); // C notation

Unary Operators - Example

i = 2
j = 3

i++ = 2
j++ = 3

After processing,
i = 3, j = 4

K = (i++) + (j++)

= 2 + 3

= 5

i = 3, j = 4

++i = 4

++j = 5

After processing,
i = 4 and j = 5

l = (++i) + (++j)

= 4 + 5

= 9

i = 4 and j = 5

--i = 3

--j = 4

After processing,
i = 3 and j = 4

$m = (i--) * (j--)$

$= 4 * 5$

$= 20$

$n = (--i) * (--j)$

$= 2 * 3$

$= 6$

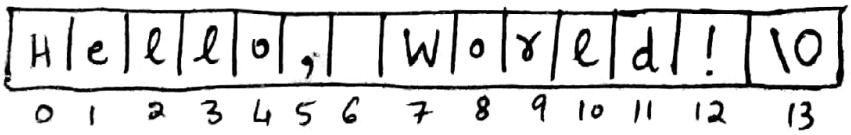
Eg ② :- sizeof(char) = 1 byte

char s[] = "Hello, World !";

sizeof s returns 14 bytes, including the NULL character

or

sizeof(s) returns 14 bytes, including the NULL character.



\0 indicates the end of string / NULL characters

Eg :- (in C)

printf ("float: %d \n", sizeof x);

O/p

float: 4

② Arithmetic operators - If A = 10 and B = 20

		Result
A + B	Addition	30
A - B	Subtraction	-10
A * B	Multiplication	200
A / B	Division	0.5
A % B	Modulus operation	10

* / %
Higher Precedence
+ -
Lower Precedence

③ Relational operators - If A = 10 and B = 20

A > B	False (0)	The Condition A>B checks whether left operand A is greater than right operand B
A < B	True (1)	The condition A<B checks whether left operand A is less than right operand B.
A >= B	False (0)	The condition A>=B checks whether A is greater than or equal to B.
A <= B	False (0)	The condition A<=B checks whether A is less than or equal to B

Equality Operators

A == B	False (0)	Checks whether the expressions A and B are equivalent.
A != B	True (1)	Checks whether A and B are not equivalent.

$i = 8, j = 15, K = 4$

$2 * ((i \% 5) * (K + (j - 3) / (K + 2)))$

Evaluating the arithmetic expression,

$= 2 * ((8 \% 5) * (4 + (15 - 3) / (4 + 2)))$

$= 2 * 3 * (4 + (12/6)) \Rightarrow 2 * 3 * 6 = \underline{36}$

④ Logical Operators –

a) Logical AND &&

Eg:- A && B

If both the operands A and B are true, then A && B becomes true, indicating a value 1.

b) Logical OR ||

Eg:- A || B

If either of the operands A or B are true, then A || B becomes true, indicating a value 1.

c) Logical NOT !

This unary operator negates the value of a logical expression.

If a condition is TRUE, Logical NOT operator will make the condition false.

If (A && B) is True,

!(A && B) becomes False indicating a value 0.

⑤ Bitwise operators –

These operators works on bits and performs bit-by-bit operation.

Bitwise AND operator (&)

It copies a bit 0 or 1 to the result, if it exists in both operands.

AND Truth Table

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

```
A   = 0011 1100 (60)
B   = 0000 1101 (13)
-----------------
A&B   0000 1100 (12)
```

2 | 60
2 | 30 - 0
2 | 15 - 0
2 | 7 - 1
2 | 3 - 1
2 | 1 - 1
0 - 1

Bitwise OR operator (|)

It copies a bit if it exists in either of the operands.

OR Truth Table

A	B	A|B
0	0	0
0	1	1
1	0	1
1	1	1

```
A = 0011 1100 (60)
B = 0000 1101 (13)
-----------------
A|B = 0011 1101 (61)
```

Bitwise XOR operator (^)

It copies the bit if it is set in 1 operand but not both.

XOR Truth Table

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

```
A = 0011 1100 (60)
B = 0000 1101 (13)
-----------------
    0011 0001 (49)
```

Binary 1s complement operator (~) It flips the bits..

A = 0011 1100 (60)

1's complement ~A = 1100 0011 (195)

1 +

2's complement = 1100 0100 (-61)

Bitwise Left-Shift operator (<<)

A 0011 1100 (60)

Eg:- A<<2 1111 0000 (240)

Bitwise Right-shift operator (>>)

A = 0011 1100 (60)

A>>2 0000 1111 (15)

⑥ Conditional operator (Ternary operator ?:)

This operator ?: takes three operands as input and evaluates either second or third expression, based on the condition given by conditional expression.

```
int i=1, j=2;
cout << (i>j) ? i : j << "is greater";
```

Output

2 is greater

⑦ Assignment operator - (Right to Left Associativity)

(a) Simple Assignment operator '='

Syntax:-

identifier = expression ;

The above syntax assigns the values computed from R.H.S operands to L.H.S operand.

The identifier can be a variable, whereas an expression can include constants, variables and other expressions.

The expression C = A + B will assign the value of A+B to C.
(expression, variable)

The variable initialization statement,

Eg① - x = 1;

assigns the value of 1 to x, (variable)
using assignment operator =
where x is the L-value and
1 is the R-value.

char c = 'x'

Eg ② - int i = 'x'; // i will have the ASCII value 120.

The above initialization assigns only the ASCII value of x in i. Each ASCII value has a 7-bit ASCII code.

Eg ③ - i = j = 5;

The above expression is a multiple assignment expression, which assigns value 5 to variable j and assigns variable j to variable i.

Operator Precedence Table

Operator Category	Operators	Associativity
Unary operators	- ++ -- ! ~ sizeof(type)	R → L
Arithmetic Multiplication, Division, Modulus operators	* / %	L → R
Arithmetic Addition, Subtraction	+ -	L → R
• Bitwise Left Shift and Right Shift operators	<< >>	L → R
Relational operators	< <= > >=	L → R
Equality operators	== !=	L → R
• Bitwise AND, OR, XOR	& | ^	L → R
Logical AND	&&	L → R

Logical NOT (!) is meant for boolean values and Bitwise NOT (~) for integers.

Operator Category	Operators	Associativity
Logical OR	||	L → R
Conditional operator	?:	R → L
Assignment operators	=, +=, -=, *=, /=, %=	R → L
Comma operator	,	L → R

The compound assignment operators are

a += 2;	meaning	a = a + 2;
b -= 5;	meaning	b = b - 5;
c *= 20;	meaning	c = c * 20;
d /= 10;	meaning	d = d / 10;
e %= 3;	meaning	e = e % 3;
f <<= 5;	meaning	f = f << 5;
g >>= 1;	meaning	g = g >> 1;

DIVYA CHRISTOPHER

Data Input and Output -

C language is accompanied by a collection of library functions including input/output functions like getchar, putchar, scanf, printf, gets and puts. These six functions permits transfer of information between the computer and standard input/output devices (Keyboard and a TV Monitor).

- First two functions, getchar() and putchar() allows single characters to be transferred into and out of the computer.
- scanf() and printf() permits transfer of single characters, numeric values and strings, into and out of the computer.
- gets() and puts() facilitates input and output of strings.

All the functions, getchar(), putchar(), scanf(), printf(), gets() and puts(). are written as function name followed by a list of arguments in parantheses.

C includes a set of header files for each of the input/output library functions. All these functions are included in the header file, stdio.h.

Header files are inserted into the program via #include statement at the beginning of the program.

① Single Character Input - getchar() function

Single characters can be entered into the computer using getchar().

getchar() function returns a single character from a standard input device like Keyboard. This function do not accept any arguments.

```
char c;
c = getchar();
```

② Single Character Output - putchar() function

Single character can be displayed using putchar() function.

putchar() function transmits a single character to a standard output device like monitor.

Eg:-
```
#include<stdio.h>
#include<ctype.h>
char c;
putchar(c);
```

Eg-
```
char letter[15];
letter = "LOURDES MATHA";
```

Another method

```
for(i=0; i<10; i++)
{ putchar(toupper(letter[i]);
}
```

(3) Entering Input data using scanf() function

Input data can be entered into the computer via a standard input device, like Keyboard.

Syntax -

```
scanf(control string, arg1, arg2, ...., argN);
```

Control String consists of individual character group. Each character group begins with a percent sign (%) followed by a conversion character.

Each argument read as variables should be preceeded by ampersand symbol (&) because each variable represents a memory address.

Conversion character	Meaning
%c	Single character
%d	decimal integer
%f	floating point value
%h	short integer
%i	decimal, octal, hexadecimal integer
%u	unsigned decimal integer
%x	hexadecimal integer

Eg:-

```
#include <stdio.h>
void main()
{
 char item[30];
  int partno;
   float cost;
scanf("%s %d %f", item, &partno, &cost);
}
```

Eg :- scanf("%[^\n]", line);

Using the above declaration, any ASCII character except the newline character can be entered by the user, from the standard input device, keyboard. For this, a circumflex character ^ should precede the new line character within square brackets [ ]. (i.e [^\n]).

Any Field width can be specified for each data item as follows:

scanf("%3d %3d %3d", &a, &b, &c);

Field width is placed within the (percent sign) control string between % sign and conversion character.

④ Writing output data using printf() function -

Output data can be written from the computer onto a standard output device, Monitor using library function, printf.

Syntax -

printf (control string, arg1, arg2, ..., argn);

where control string refers to a string that contains formatting information arg1, arg2, ..., argn are the arguments that represents individual output data items.

Eg:-

```
#include <stdio.h>
void main()
 {
  char item[20];
  int partno;
  float cost;
```

```
printf("%s %d %f", item, partno, cost);
. . . . .
}
```

In the above example, arguments in a printf function do not represent memory addresses and therefore are not preceded by ampersands (&).

It is also possible to specify the maximum number of decimal places for a floating-point value. This specification is known as precision.

```
eg:- #include <stdio.h>
     void main()
     {
      float X = 123.456;
      printf("%f  %.3f  %.1f\n\n",X,X,X);
     }
```

output

123.456000 123.456 123.5

(5) gets() and puts() function –

These functions facilitates transfer of strings between the computer and standard input/output device.

The input to gets() function represents a string and output from puts() function also represent a string.

After entering input data, the string will be terminated with a newline character.

Eg:-

```
#include <stdio.h>
#include <conio.h>
void main()
{
 char line[80];
 gets(line);
 puts(line);
 getch();
}
```

Output

```
LMCST
LMCST
```

Interactive Programming

Many modern computer programs are designed to create an interactive dialog between computer and the user using program.

Eg :- Average Marks computation – C program

```
#include <stdio.h>
void main()
{
char name[20];
float score1, score2, score3, avg;
printf("Enter your name:");
scanf("%[^\n]", name);   //A string including white space character can be read
printf("First score:");
scanf("%f", &score1);
```

```
printf("Second Score:");
  scanf("%f", &score2);
printf("Third Score:");
 scanf("%f", &score3);
 avg = (score1 + score2 + score3)/3;
printf("\n Name: %s\n", name);
 printf("Score 1: %f\n", score1);
printf("Score 2 :%f\n", score2);
printf("Score 3: %f\n", score3);
printf("Average: %f\n", avg);
}
```

output

```
Enter your name : Malavika
First Score : 80
Second Score : 60
Third Score: 70
Name: Malavika
Score 1 : 80
Score 2 : 60
Score 3 : 70
Average: 70
```

Other Library functions in C

C language is accompanied by a number of library functions that carry out certain operations.

Purpose of functions:-

→ Some functions return a data item.

→ Other functions indicates whether a condition is true or false, by returning a value 1 or 0 respectively.

Types of functions:-

→ Functions to carry out standard input/output operations for reading/writing files, opening and closing files and to test for end of file (EOF)

→ Functions that performs operations on characters like getchar(), putchar(), toupper() and tolower().
(lower case to upper case) (upper case to lower case)

→ Functions that performs operations on strings like strcmp(), strcpy(), strlen()
(string compare) (string copy)
strcat(), etc
(string concatenation)

→ Functions to carry out calculations, like abs(), sqrt(), ceil(), sin(), cos(), log() etc.

A library function is accessed by simply writing the function name followed by a list of arguments passed to the function.

Examples of other library functions - C

Function	Type	Purpose
abs(i)	int	Returns the absolute value of i.
ceil(d)	double	Rounds up a number to the next integer value.
cos(d)	double	Returns cosine of d
sin(d)	double	Returns sine of d
floor(d)	double	Rounds
tan(d)	double	Returns tangent of d.
rand()	int	Returns a random positive integer.
sqrt(d)	double	Returns the square root of d
pow(d1, d2)	double	Returns d1 raised to d2 power.
getchar()	char	Gets a character from the standard input device
putchar()	chars	Puts a character to the standard output device.

.....

void main()

Function	Type	Purpose
printf("control string", arguments);	int	Send numbers, characters or strings to standard output device.
scanf("control string", arguments);	int	Reads numbers, characters or strings from standard input device.
toascii(c)	int	Converts value of argument to ASCII.
tolower(c)	int	Converts letter to lower-case.
toupper(c)	int	Converts letter to upper-case.

c - character type argument

i - integer argument

d - double-precision argument

DIVYA CHRISTOPHER

Introduction to C

C is a general-purpose structured programming language augmented by ENGLISH-LIKE KEYWORDS.

BASIC STRUCTURE OF A C PROGRAM

Header Files (Include Files)

Main function

Begin

Variable Declaration statements

Input statements

Processing statements

Output statements

End

```
#include<stdio.h>
void main()
{
printf("Hello World");
}
```

The Header files stdio.h and conio.h are included using #include directive.

stdio.h contains all standard input/output library functions in C

conio.h contains all console input/output functions in C

```
clrscr();   // Clear screen function
getch();    // holds the output screen
            // until a key in the keyboard
            // is pressed.
```

Main function is represented as

```
void main()
```

The function is terminated by Parantheses.

Begin and end are denoted by a pair of curly braces { }

It should be noted that variables should be declared before use in the program.

Input statements in C are written using scanf() statements, which assigns values to address of variables.

After acquiring values, processing is applied to the variables holding values using expression statements for processing.

After processing, results are printed on the output screen (Monitor) using printf() statement.

/* Eg of a program to calculate the area of a circle */

C program in LINUX

```
#include <stdio.h>          // Header Files
int main()                  // Main Function
{
    float radius, area;     // Variable declarations
    printf("Enter the radius of a circle"); // O/P statement
    scanf("%f", &radius);   // Input statement
    area = 3.14 * radius * radius; // Processing statement
    printf("Area = %f", area);     // Output statement
    return 0;
}
```

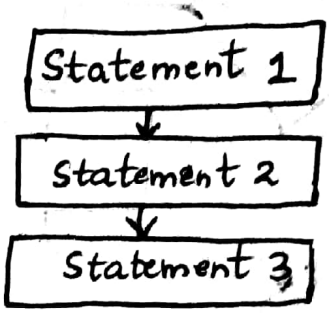
DIVYA CHRISTOPHER

Control Flow Statements in C -

Control Flow statements controls the flow of execution in a program.

(a) Sequential Control Structure -

It executes a set of statements in a sequential manner until } (closing brace) is encountered. It follows a straight line execution.



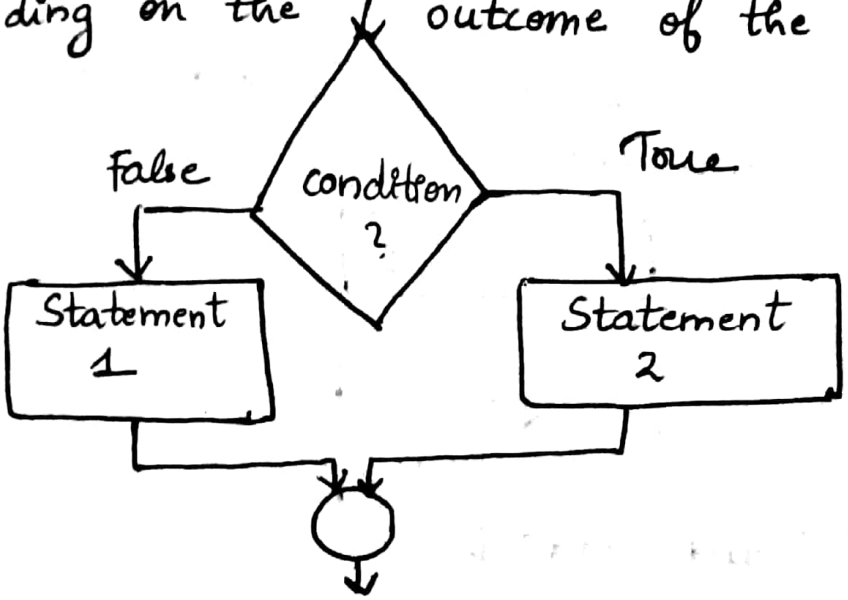

```
#include <stdio.h>
int main()
{
int a, b, sum;
printf("Enter 2 numbers");
scanf("%d %d", &a, &b);
sum = a+b;
printf("\n Sum=%d", sum);
return 0;
}
```

O/P

```
Enter 2 numbers
3 4
Sum = 7
```

(b) Selection Control Structure - (Branching)

It performs a logical test on a condition to check whether the condition is True or False. A decision is taken to execute one group of statements, among several available groups, depending on the outcome of the test (condition).



(i) if statement -

If expression (condition) is true (has a value 1), then statement 1 will be executed or else control is transferred to a sequence of statements following the curly braces { }.

Syntax -

```
if(expression)
{
statement 1;
}
sequence of statements;
```

Eg:- ①

```
if (x<0)
    printf ("%d", x);
```

Eg ② :-

```
if ((balance <1000)||(status == 'R'))
    printf("%f", balance);
```

(ii) if-else statement -

Syntax-

```
if (expression)
 {
  statement 1;
 }
else
 {
  statement 2;
 }
statement 3;
```

If condition is true (value 1), the statements within if block is executed.

If condition is false (value 0), the statements within else block is executed.

Eg① -

```
if (a > b)
    printf("%d", a);
else
    printf("%d", b);
```

(iii) if-elseif-else statement -

Multiple conditions could be embedded in if-elseif-else statement. But, control is transferred to one group of statements, among several available groups.

Syntax-

```
if (condition 1)
{
sequence of statements 1;
}
else if (condition 2)
{
sequence of statements 2;
}
else
{
sequence of statements 3;
}
sequence of statements 4;
```

(iv) switch statement -

It is a multiple branching statement in which control is transferred to one of the many possible cases, whose case label matches with the value in the switch expression. Expression can be int or char.

Syntax -

```
switch (expression)
{
case constant-expression 1:
   {
    statement 1;
   }
case constant-expression 2:
   {
    statement 2;
   }
   .
   .
   .
case constant-expression n:
   {
    statement n;
   }
   .
   .
   .
default:
    {
     statement ;
    }
    statements;
```

Multiple Case Labels in a switch expression

Case group 1

Case group 2

Case group n

default case group. If none of the case groups within case labels, matches with the value of expression in switch statement, the group of statements labeled as default will be executed

Eg:- ①

```
switch (choice)
{
 case '+':
    {
     c = a + b;
     break;
    }
Case '-':
   {
    c = a - b;
    break;
   }
case '*':
  {
  c = a * b;
  break;
  }
case '/':
   {
   c = a / b;
   break;
   }
default:
    cout << "Invalid choice";
    break;
}
```

Sum will be computed, if choice is +,
Difference will be computed, if choice is -.
of 2 numbers

Eg :- ②

```
switch (choice = getchar())
 {
 case 'r':
 case 'R':
   printf ("Red");
   break;
case 'w':
case 'W':
   printf("White);
   break;
case 'b':
case 'B':
   printf("Blue");
}
```

Red will be displayed, if the choice is either r or R, White will be displayed if the choice is either w or W and Blue will be displayed if the choice is either b or B.

Jumping statements

a) break;

Break statement is used to terminate from a loop (for, while or do-while) or switch statement.

b) continue;

Continue statement is used to bypass the remainder of the current pass through a loop. The loop does not terminate, when a continue statement is encountered, and the computation proceeds directly to the next pass through the loop, by skipping the remaining loop statements.

c) goto statement -

goto statement is used to transfer control from one part of the program to another part, where label is used as an identifier to mark the target statement.

Syntax:

```
goto label;
```

label: statement

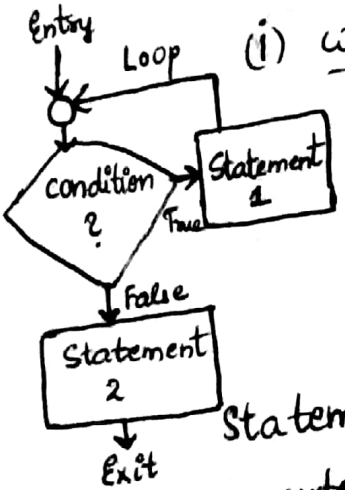
The target statement will appear as shown above

Each labeled statement in the program must have a unique label.

Eg:- goto 100;
.....
100: cout <<" Welcome to LMCST\n";

(C) Looping Structure -

It is used to repeatedly execute a group of statements until a certain condition is satisfied.



(i) while statement -

Syntax:- while (condition) //entry controlled loop
{
statement 1;
statement n;
}

Statement 1 in while expression will be executed repeatedly, as long as the expression is true, in the while condition.

In a while statement, the number of passes are not known in advance. A group of statements are executed repeatedly until the condition is true. When the condition becomes false, loop is terminated.

Eg:- Program to display digits from 0 to 9.

```
#include <stdio.h>
int main()
{
 int dig = 0;
 while (dig <= 9)
  printf ("%d \n", dig);
  ++dig;
  return 0;
 }
```

O/P

```
0
1
2
3
4
5
6
7
8
9
```

The program can also be written as,

```
#include <stdio.h>
int main()
{
 int dig = 0;
  while (dig <= 9)
    printf ("%d \n", dig++);
  return 0;
}
```

O/P

```
0
1
2
3
4
5
6
7
8
9
```

(ii) do-while statement -

Syntax:-

```
do                          // exit controlled loop
{
 statement 1;
 .....
 statement n;
} while (condition);
```

A do-while statement will be executed repeatedly, as long as the value of the (condition) expression is true (value 1).

Note that a do-while statement will be executed atleast once, when condition becomes false since the test for repetition does not occur until the end of first pass through the loop.

Eg:-

```
#include <stdio.h>
int main()
{
int dig=0;
do
{
printf("%d \n", dig++);
}
while (dig <=9);
return 0;
}
```

O/P

```
0
1
2
3
4
5
6
7
8
9
```

(iii) for statement -

In a for statement, the number of passes are known in advance.

Syntax-

```
for (expression 1 for initialization ; expression 2 for condition ; expression 3 for increment/decrement)
{
statement 1;
statement 2;
.....
statement n;
}
```

The expression 1 indicates initial value of loop index.
The condition in for loop is a logical expression.
(expression 2)

The expression 3 is a unary expression used to either increment / decrement the loop, using increment / decrement operators.
(unary)

For statement is executed as long as condition in expression 2 is true.

The condition in expression 2 is evaluated and tested at the beginning of each pass through the loop and expression 3 is evaluated at the end of each pass.

Looping action coill continue, until the value of expression 2 is true.

When condition becomes false, for loop terminates.

Syntax:-

```
int n=100;   // Increment loop
             // increments from 0 to 99.
for (i=0 ; i<n ; i++)      // This for loop
    {                      // terminates
    printf ("%d ", i);     // only when
    }                      // i=100
```

Syntax 2 -

```
for (i=n ; i>0 ; i--)
  {
   printf ("%d", i);
  }
```

// Decrement loop decrements from 100 to 1.
// This for loop terminates when i = 1

Eg:- To display digits from 0 to 9.

```
#include <stdio.h>
int main()
{
 int dig;
 for (dig=0; dig<=9 ; ++dig)
 printf ("%d\n", dig);
 return 0;
}
```

O/p

0
1
2
3
4
5
6
7
8
9

Eg:-
```
for(ch='a' ; ch<='z' ; ch++)
 {
  cout<<ch;
 }
```

Eg:-
```
char ch='a';
while(ch<='z')
  {
   cout<<ch;
   ch++;
  }
```

These 2 expressions 1a and 1b initializes 2 separate indices that would be used within for loop.

Usage of Comma operator in for loop

```
for (expression 1a,
     expression 1b;
     expression 2 ;
     expression 3)
  {
   Statement;
  }
```

expression 1a and 1b are two expressions separated by comma operator.

Qn) Write a C program to print the sum of 2 numbers?

```
#include <stdio.h>
int main()
{
int a, b, c;
printf("Enter the 2 numbers");
scanf("%d%d", &a, &b);

c = a + b;
printf("Sum = %d", c);
}
```

Qn) Write a C program to print the Fibonacci Series?

```
#include <stdio.h>
int main()
{
int N;
int A, B, C, count;
printf("Enter the no: of terms to be
                    generated");

scanf("%d", &N);

A = 0;
B = 1;
printf("%d %d\n", A, B);
```

```
for (count = 2 ; count < N ; count++)
  {
    C = A + B;
    printf ("%d \n", C);
    A = B;
    B = C;
  }
}
```

Qn) write a C program to find the reverse of a given number?

```
#include <stdio.h>
int main()
{
 long int num, rem, rnum;
 printf ("Enter the number");
 scanf ("%d", &num);
 rnum = 0;
 do
 {
  rem = num % 10;
 rnum = (rnum * 10) + rem;
  num = num / 10;
 }
  while (num > 0);
printf (" The reverse is %d ", rnum);
 }
```

Qn) Write a C program to convert the binary number to decimal number?

```
#include <stdio.h>
int main()
{
 long BinNum;
 int Binbit;
 int DecNum, i;
printf(" Enter the binary number");
scanf ("%d", &BinNum);
 DecNum = 0;
 i = 0;
while (BinNum > 0)
 {
  Binbit = BinNum%10;
DecNum = DecNum + Binbit * pow(2, i);
 BinNum = BinNum/10;
  i++;
 }
printf("The decimal number is %od", decnum);
}
```

Qn) Write a C program to find the sum of sequence, $\frac{2}{9} - \frac{5}{13} + \frac{8}{17} + \ldots$?

```
#include <stdio.h>
int main()
{
int count, i;
float num, den, sum;
  num = 2;
  den = 9;
  sum = 0;
 printf(" Enter the no: of terms");
 scanf("%d", &count);
for(i=0; i<count; i++)
 {
sum = sum + (((num/den) * pow(-1, i)));
num = num + 3;
den = den + 4;
}
printf(" Sum of the series is %d", sum);
}
```

Q) write a C program to find the sum of digits in a number & the no: of digits in a number?

```
#include <stdio.h>
int main()
 {
 int num, sum, digit = 0;
printf("Enter the number");
scanf("%d", &num);
while (num > 0) || (sum > 9))
 {    if(num == 0) { num = sum;
                     sum = 0; }
 num = num % 10;

 sum = sum + num;
  digit ++;
 num = num / 10;
 }
printf("Sum = %d", sum);
printf("No: of digits in the integer no: %d",
                                    digit);
}
```

Qn) Write a C program to find whether a number is an Armstrong or Not?

```
#include <stdio.h>
int main()
{
 int b, n, s = 0;
 printf("Enter the number ");
 scanf("%d ", &n);
  b = n;
 while (n > 0)
 {
  a = n % 10;
  s = s + (a * a * a);
  n = n / 10;
 }

if(b == s)
   printf(" The no: is an armstrong no");
else
   printf("The no: is not an armstrong no:");
}
```

Qn) Write a program to find the largest of 3 numbers?

```
#include <stdio.h>
int main()
{
printf("Enter 3 numbers");
scanf("%d %d %d", &a, &b, &c);
if((a>b) && (a>c))
 printf("%d", a, "is the largest");
 else if (b>c)
  printf("%d", b, "is the largest");
 else
  printf("%d", c, "is the largest");
}
```

Qn) write a program to find the largest of 2 numbers?

```
#include <stdio.h>
int main()
{
 int a, b;
  printf("Enter 2 numbers");
  scanf("%d %d", &a, &b);
   if(a>b)
   printf("%d", a, "is the largest");
}else printf("%d", b, "is the largest");
```

Qn) Write a C program to check whether a number is prime or not?

```
#include <stdio.h>
int main()
{
    int i, n, flag = 1;
printf("Enter the number");
scanf("%d", &n);
for(i = 2; i < n/2; i++)
  {
   if(n % i == 0)
    {
     flag = 1;
     break;
    }
  }
if(flag == 1)
printf("Number is not prime");
else
   printf("Number is prime");
}
```

Qn) Write a C program to read an English alphabet through Key Board and display whether the given alphabet is lower case or upper case?

```
#include <stdio.h>
int main()
{
char ch;
printf("Enter the character");
scanf ("%c", &ch);
if(ch >= 'A' && ch <= 'Z')
  printf ("Character is upper-case");
else
   printf ("Character is lower-case");
}
```

}

Qn) Write a C program to find whether a number is odd or even using bitwise operator?

```
#include <stdio.h>
int main()
{
int n;
printf("Input an Integer\n");
scanf("%d", &n);
if(n & 1 == 1)
  printf("Odd \n");
else
  printf("Even \n");
}
```

Qn) Write a C program to exchange the contents of 2 integers?

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

void main()
{
 int a, b, temp;
 printf("Enter the 2 numbers");
 scanf("%d %d", &a, &b);
 temp = a;
  a = b;
  b = temp;
printf("The exchanged values are");
printf("%d \t %d \t", a, b);
getch();
}
```

Qn) Write a C program to find whether the given year is a leap year or not?

```
#include <stdio.h>
#include <conio.h>
void main()
{
 int lyear;
 printf("Enter the leap year");
 scanf("%d", &lyear);
if ((lyear % 4 == 0) && (lyear % 100 != 0)
    || (lyear % 400 == 0))
 printf ("Year is a leap year");

else
 printf ("Year is not a leap year");

getch();
}
```

Qn) Write a C program to print the ASCII value of a given symbol?

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

void main()
{
int num;
char ch;
printf("Enter the symbol");
scanf("%c", &ch);

 num = ch;
 printf("Ascii value of %c is %d",
                        ch, num);
 getch();
}
```

Qn) Write a C program to print the largest of two numbers?

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a, b;
printf("Enter 2 numbers");
scanf("%d %d", &a, &b);
if(a > b)
printf("%d is greater", a);
else
printf("%d is greater", b);
getch();
}
```

(p) write a C program to print the largest of three numbers?

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a, b, c;
printf("Enter 3 numbers");
scanf ("%d %d %d", &a, &b, &c);
if((a>b) &&(a>c))
printf ("%d is the largest", a);
else if(b>c)
printf("%d is the largest", b);
else
printf ("%d is the largest", c);
getch();
}
```

Qn) Write a C program to print alphabets from a to z?

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

void main()
{
  char ch='a';
  while (ch<='z')
  {
    printf("%c", ch);
    ch++;
  }
  getch();
}
```

```
#include <stdio.h>
#include <conio.h>
  void main()
 {
char ch='a';
do
 {
 printf("%c", ch);
 ch++;
} while(ch<='z');
getch();
}
```

Qn) Write a C program to find the roots of a quadratic equation?

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
float a, b, c, X1, X2, r1, r2;
printf ("Enter the three coefficients");
Scanf ("%f %f %f", &a, &b, &c);
X1 = b * b - 4 * a * c;
X2 = 2 * a;
if (X1 == 0)
{
r1 = (-1 * b) / X2;
r2 = r1;
```

```
printf("The roots are real and equal");
printf("\n Root1: %d", r1);
printf("\n Root 2: %d", r2);
}
else if (x1 > 0)
{
x1 = sqrt(x1);
r1 = (-1 * b + x1) / x2;
r2 = (-1 * b - x1) / x2;
printf("The roots are real and unequal");
printf("\n Root 1: %d", r1);
printf("\n Root 2: %d", r2);
}
else
printf("Roots are imaginary");
getch();
}
```

Qn) Write a C program to print the first 50 prime numbers?

```
#include <stdio.h>
#include <conio.h>
void main()
{
int N, i=0, j, isPrime;
printf("Enter the limit");
scanf("%d", &N);
for(i=2; i<=N; i++)
{
isPrime = 0;
for(j=2; j<=i/2; j++)
{
if(i%j==0)
{
isPrime = 1;
break;
}
}
}
```

```
if (isPrime == 0 & N != 1)
    printf("%d \t", i);
  }
 }
 getch();
}
```

Qn) Write a C program to find the factorial of a given number?

```
#include <stdio.h>
#include <conio.h>
void main()
{
 long int num, fact;
 printf("Enter the number");
 scanf("%d", &num);
 fact= 1;
```

```
do
{
  fact = fact * num;

  num--;
}
while (num > 0)
printf ("Factorial=%d", fact);
getch();
}
```

Qn) Write a program to print the factorial of a number using for loop?

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i, n, fact = 1;
printf("Enter the number");
scanf ("%d", &n);
for (i=1; i<=fact; i++)
  fact = fact * i;
printf ("Factorial = %d", fact);
} getch();
```

```
do
{
  fact = fact * num;

  num--;
}
while (num > 0)
printf ("Factorial=%d", fact);
getch();
}
```

Qn) Write a program to print the factorial of a number using for loop?

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i, n, fact = 1;
printf("Enter the number");
scanf("%od", &n);
for(i=1; i<=fact; i++)
  fact = fact * i;
printf("Factorial = %d", fact);
} getch();
```