unit 1 notes

What is Programming Language(PL)?

- Formal computer language or constructed language designed to communicate instructions to a machine, particularly a computer
- A way of telling a machine what operations to perform
- Can be used to create programs to control the behaviour of a machine or to express algorithms

formatted output function-printf()

- Output is controlled by the first argument
- Has the capability to evaluate an expression
- On success, it returns the number of characters successfully written on the output. On failure, a negative number is returned.
- Arguments to printf can be expressions.
- On success, it returns the number of characters successfully written on the output. On failure, a negative number is returned.
  example 1

```
#include <stdio.h>

int main()

{ printf("Let us C");

return 0;

}
```

output:

```
Let us C
```

example 2

```
#include <stdio.h>

int main()

{

printf("hello Friends");

return 0;

}
```

output:

```
hello Friends
```

The first argument to printf is a string. The output depends on this string. String literals in ‘C’ are enclosed in double quotes.

example 3

```
warning: data argument not used by format string [-Wformat-extra-args]
printf("hello ", "friends"); // hello ;
       ~~~~~~~~  ^
1 warning generated.
hello
```

Output depends on the first string. As there is no interpretation of the second parameter, it would not appear in the output.

example 4

```
#include <stdio.h>

int main()

{

printf("hello %s\n", "friends\n"); // hello friends
```

```
return 0;

}
```

output:

```
hello friends
```

The presence of %s in the first string makes the function printf look for the next parameter and interpret it as a string.

example5

```
#include <stdio.h>

int main()

{

printf("%s %s %s %s\n", "one", "two", "three", "four");

return 0;

}
```

output:

```
one two three four
```

The function printf takes variable number of arguments. All arguments are interpreted as the number of %s matches the number of strings following the first string.

example 6

```
#include <stdio.h>

int main()

{
```

```
printf("%s %s %s %s\n", "one", "two", "three");

return 0;

}
```

output

```
warning: more '%' conversions than data arguments [-Wformat-insufficient-args]
printf("%s %s %s %s\n", "one", "two", "three");
                ~^
1 warning generated.
one two three (null)
```

NO! we are in for trouble. There is no argument for the 4th%s in the format string. So, printf tries to interpret as a string. If we are lucky, the program will crash. We have an “undefined behaviour”. C does no checking at runtime!

example 7

```
#include <stdio.h>

int main()

{

printf("%5d and %5d is %6d\n", 20, 30, 20 + 30); // Discussed in detail: Format String

return 0;

}
```

output:

```
 20 and    30 is     50
```

Arguments to printf can be expressions – Then the expressions are evaluated and their values are passed as arguments.
%d : indicates to printf to interpret the next parameter as an integer. %5d : tells printf to display the integer using 5 character width.

example 8:

```
#include <stdio.h>

int main()

{

int a = 20;

int n = printf("%d\n",a);// printf executes and returns # of characters successfully printed on the terminal

printf("n is %d",n); //n=3: 3 characters are printed on monitor

return 0;

}
```

output:

```
20
n is 3
```

example 9:

```
#include <stdio.h>

int main()

{

printf("what : %d\n", 2.5);

return 0;
}
```

output:

```
undefined behaviour
```

example 10:

```
#include <stdio.h>

int main()

{

int num1; float num2;

printf ("Input an integer value:");

scanf ("%d",&num1);

printf ("Input a float value:");

scanf ("%f", &num2);

printf ("The value of num1 is %d\n",num1);

printf ("The value of num2 is %f",num2);

return 0;

}
```

output:

```
Input an integer value:13
Input a float value:34.5
The value of num1 is 13
The value of num2 is 34.500000
```

format string

- Flags -optional argument which specifies output justification such as numerical sign, trailing zeros or octal, decimal, or hexadecimal prefixes
- field_width -optional minimum width of the output field
- Precision -optional argument which specifies the maximum number of characters to print.
- Conversion character indicates the type of the argument that is being formatted or parsed.
- Ex- printf("%8.1f\n",num) ;

escape sequence

- \a
  Alarm or Beep
  It is used to generate a bell sound
- \b
  Backspace
  It is used to move the cursor one place backward.
- \
  Backlash
  Use to insert backslash character.
- \n
  New Line
  It moves the cursor to the start of the next line.
- \r
  Carriage Return
  It moves the cursor to the start of the current line.
- \t
  Horizontal Tab
  It inserts some whitespace to the left of the cursor and moves the cursor accordingly.
- \’
  Single Quote
  It is used to display a single quotation mark.
- \”
  Double Quote
  It is used to display double quotation marks.
  example:

```
#include<stdio.h>

int main()

{

int a,b;

a = 16;

b = 78;

float c = 2.5;

// sizeof operator returns the size of the variable in bytes

printf("%zu %zu\n", sizeof(a), sizeof(short int));

// Printing a float with width 4 and precision 2

printf("%4.2f\n", c);

// Printing a float with width 2 and precision 1

printf("%2.1f\n", c);

// Left justification with width 5 for the first integer and normal for the second

printf("%-5d %d\n", 16, b);

// Using \r to move the cursor to the beginning of the line and overwrite characters

printf("ravan\rram\n");

// Using \n to move to the next line

printf("ram\nlakhan\n");
// Using \t for tabulation

printf("ram\tbheem\n");

// To include quotes in the output, you should use double quotes

printf("\"Mahabharath\"");



return 0;

}
```

output:

```
4 2
2.50
2.5
16    78
raman
ram
lakhan
ram     bheem
"Mahabharath"
```

Note:
sizeof(short int) <= sizeof(int) <= sizeof(float) <= sizeof(double)
size of a type depends on the system of implementation. We can know the exact size by using sizeof() operator.
Ex. sizeof(int),sizeof(float) etc.

formatted input function-scanf():

Syntax:

- int scanf(const char *format, list of arguments...)
- It’s a predefined function declared in stdio.h
- & - address operator is compulsory in scanf for all primary types

- Reads formatted input using stdin. By default it is keyboard
- This function returns the following value
  >0 — The number of items converted and assigned successfully
  0 — No item was assigned.
  <0 — Read error encountered or end-of-file (EOF) reached before any assignment was made

example 1:

```
#include <stdio.h>

int main()

{

int a,b,ret_Value_scanf,ret_Value_printf;

printf("Enter the value of a and b");

ret_Value_scanf=scanf("%d%d",&a,&b);

ret_Value_printf=printf("a : %d b : %d\n", a, b);

printf("The return value of scanf is %d\n",ret_Value_scanf);

printf("The return value of printf is %d",ret_Value_printf);

return 0;

}
```

output:

```
Enter the value of a and b5,8
a : 5 b : 1
The return value of scanf is 1
The return value of printf is 12
```

example 2:

```
#include <stdio.h>



int main() {

int var1;

int var2;

printf("enter the values:");

scanf("%d %d", &var1, &var2);

printf("%d %d", var1, var2);

return 0;

}
```

output:

```
10 20
```

Example 3:

```
#include <stdio.h>



int main() {

int var1;

int var2;

printf("enter the values:");

scanf("%d,%d", &var1, &var2);
printf("%d,%d", var1, var2);

return 0;

}
```

output:

```
10,20
```

scanf has a comma between two format specifiers, the input should have a comma between a pair of integers.

Example 4:

```
#include <stdio.h>



int main() {

int var1;

int var2;

printf("enter the values:");

scanf("%d%d/n", &var1, &var2);

printf("%d%d", var1, var2);

return 0;

}
```

output:

```
It is not a good practice to have any kind of escape sequence in scanf. In
the above code, it expects two integers. Ex: 20 30. Then if you press enter
```

```
key, scanf does not terminate. You need to  give some character other than
the white space.
```

unformatted output functions

These functions are not capable of controlling the format that is involved in writing the available data. Hence these functions constitute the most basic forms of output. The display of output isn’t allowed in the user format, hence we call these functions as unformatted functions. The unformatted output functions further have two categories: Character functions and string functions
Character functions: putchar(ch); where ch is the character variable
example:

```
#include<stdio.h>
int main()
{
char c;
printf("enter the character\n");
c = getchar();
printf("you entered %c\n",c);
printf("you entered "); putchar(c);
return 0;
}
```

output:

```
enter the character
h
you entered h
you entered h
```

unformatted input functions

A character in programming refers to a single symbol. A character in ‘C’ is like a very small integer having one of the 256 possible values. It occupies a single byte. We code English alphabets A as 65, B as 66, Z as 90, a as 97 and so on in a coding scheme called ASCII. To read a character from the keyboard, we could use

scanf(“%c”,&x); We could also use x = getchar();
We prefer the second as it is developed only for handling a single char and therefore more efficient even though it is not generic.
The unformatted input functions further have two categories: Character functions and string functions
Character functions: getchar(ch), getche(), getch() where ch is the character variable
example:

```
#include<stdio.h>

int main() {
    char ch = 'p';

    // Uncommented lines
    // printf("Ch is %c\n",ch);// p

    // This line will result in an error because 'pqrs' is a string, not a single character.
    // char ch1 = 'pqrs';// TERRIBLECODE
    // printf("Ch is %c\n",ch1);// s

    char x;
    char y;

    // These lines are commented out
    /*
    scanf("%c", &x);
    scanf("%c", &y);
    printf("x:%c y:%c\n", x, y);
    */

    // Read two characters using getchar
    x = getchar();
    y = getchar();

    // Print the characters stored in x and y
    putchar(x);
    putchar(y);

    // Print the ASCII value of y
    printf("%d", y);

    return 0;
}
```

output:

```
ab
ab98
```

tokens

- A token or a lexical unit is the smallest individual unit of a program.
- Token can be a keyword, identifier, variable, constant, a string ,literal, or a symbol.

identifiers

- It is a name used to identify a variable, function, or any other user-defined item.
- Rules for naming a identifier
  i.
  Starts with a letter A to Z, a to z, or an underscore ‘_’ followed by zero or more letters, underscore, and digits (0 to 9)
  ii.
  An identifier cannot be keyword
  iii.
  The first character cannot be a number
  Ex. of Legal Identifiers: Myname50, _temp, j, a23b9, retVal Ex. of Illegal Identifiers: 50ar, $name, auto, my-name

literals

Values assigned to Variables such as numbers, strings, etc. Ex: Integer literal , Float literal , Character literal

Operators

Symbols that perform some operation on the operands.

Punctuators

Symbols that are accepted by the compiler and is legal to be used in the program.

Constants

Constant is basically a named memory location in a program that holds a single value throughout the execution of that program

Variable declaration

- Variable is a name given to a storage area that a code can manipulate ● Has a name, location, type, life, scope and qualifiers
- An uninitialised variable will have some undefined value.
- Ex. : int a; , int b=10
- Note: C is statically typed language means once a variable is declared as a integer in C it cannot be assigned any float value.

Keywords

- are identifiers which have special meaning in C.
- Cannot be used as constants or variables ● Ex: auto, else, long, switch, break, enum, case, extern, return char, float, for, void, sizeof etc.
- int auto = 10; // Error
- There are 32 keywords in C programming Language

Data Types

- Data types specify the type and range of values that can be stored in variable, operations that can be performed on it, amount of space taken by the variable.
- Sizeof (short int)<= sizeof(int) <= sizeof(long int) <=sizeof(long long int)

- There are mainly 3 types of Data Types in C
  I.
  Primitive Data Types(int,char,float)
  II. Derived Data Types(Arrays)
  III. User Defined Data Types (struct,union)

int-%d
float-%f
char-%c
double-%lf
long double-%llf
string-%s
pointers-%p
octal number-%o
Hexadecimal number-%x or %X

L-value and R-values (Locator value and read value)

- We talk about l-value and R-value with respect to assignment operator.
- R-value refers to whatever that could occur to the right of assignment operator and L-value refers to whatever that could occur to the left of assignment operator.
- A constant has only R-value. An initialised variable has both l and r value.
- An expression of the form a + b is only a R-value.

example from notes:

```
#include<stdio.h>

#include<limits.h>

#include<float.h>

int main()

{

int a = 8;
```

```
char p = 'c';

double d = 89.7;

int e = 0113; // octal literal

/*printf("%d\n",e);

printf("%o\n",e);

printf("%X\n",e);

printf("%x\n",e);

*/

//int h = 0xRG; // error

int h = 0xA;

//printf("%d",h);

int i = 0b111;

//printf("%d",i);

//printf("%d %d %d %d\n",sizeof(int),sizeof(short
int),sizeof(p),sizeof(long));

//int g = 787878787888888787;

int g = 2147483649;

/*printf("%d\n",INT_MAX);

printf("%d\n",INT_MIN);

printf("%d\n",INT_MIN);

printf("%d\n",INT_MIN);*/
printf("%d\n",CHAR_MAX);

printf("%g\n",DBL_MAX);

printf("%g\n",FLT_MAX);

printf("%d\n",g);

return 0;

}
```

output:

```
127
1.79769e+308
3.40282e+38
-2147483647
```

getchar():

- `getchar()` reads a single character from the standard input stream (usually the keyboard).
- It reads the next available character from the input buffer and returns its ASCII value as an `int`.
- If successful, `getchar()` advances the file position indicator to the next character.
- If an error occurs or if the end of the file is reached, `getchar()` returns the special value `EOF` (End of File), which is usually defined as `-1`.
- `getchar()` is often used inside a loop to read characters until encountering a newline (`'\n'`) or another specific termination condition.
  example:

```
#include <stdio.h>

int main() {
    char c;
    printf("Enter a character: ");
    c = getchar(); // Read a single character from the standard input

    printf("You entered: ");
    putchar(c); // Print the character just read

    return 0;
}
```

output:

```
Enter a character: A
You entered: A
```

putchar(ch);

- This function accepts a mandatory parameter ch which is the character to be written to stdout.
- It also returns EOF when some error occurs.

example:

```
#include <stdio.h>

int main() {
    char ch;
    ch = getchar();   // Read a single character from standard input
    putchar(ch);      // Print the character to standard output
    return 0;
}
```

output:

```
hello
h
```

getc and putc

example:

```
#include <stdio.h>

int main() {

char c;

printf("Enter character: ");

c = getc(stdin);

printf("Character entered: ");

putc(c, __stdoutp);

return 0;

}
```

output:

```
Enter character: Preksha
Character entered: P
```

Non- standard Input Output Functions:

getch() , getche() and putch() functions are called as non-standard functions since they are not part of standard library and ISO C(certified c programming language). It is available in conio.h.

getch():

getch() also reads a single character from the keyboard.
Syntax: int getch();

- getch() method pauses the Output Console until a key is pressed.
- The entered character is immediately returned without waiting for the enter key.
- The entered character does not show up on the console.

getche():

getche() function reads a single character from the keyboard and displays immediately on the output screen without waiting for enter key.
Syntax: int getche();
Ex: ch=getche();

putch():

Putch() function outputs a character stored in the variable on the output screen.
Syntax:
putch(ch);

example:

```
#include <stdio.h>

int main() {
    char ch;
    printf("Enter a character: ");
    ch = getchar();
    putchar(ch);
    return 0;
}
```

output:

```
Enter a character: preksha
p
```

Operator and its Classification

● Operator is a symbol used for calculations or evaluations
● Has rank, precedence and Associativity
Classification:

1. Based on the operation:
   Arithmetic – Increment(++) & Decrement(--)

Relational - >, <, <=, >=, == ,!=

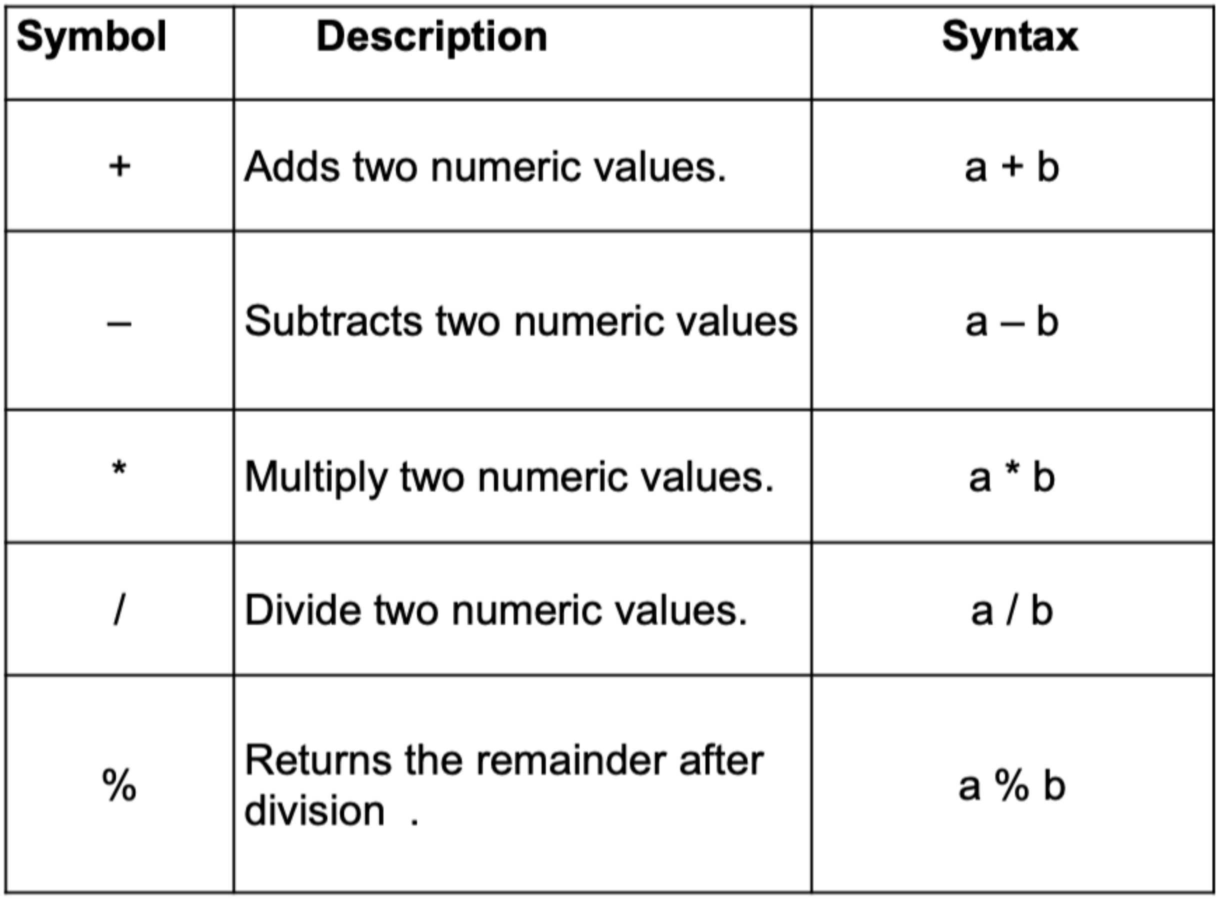
Logical(short circuit evaluation) - &&, ||, !

Bitwise - &, |, ~, ^, <<, >>

Address - &

Dereferencing Operator - *

2. Based on the number of operands: Unary, Binary, Ternary



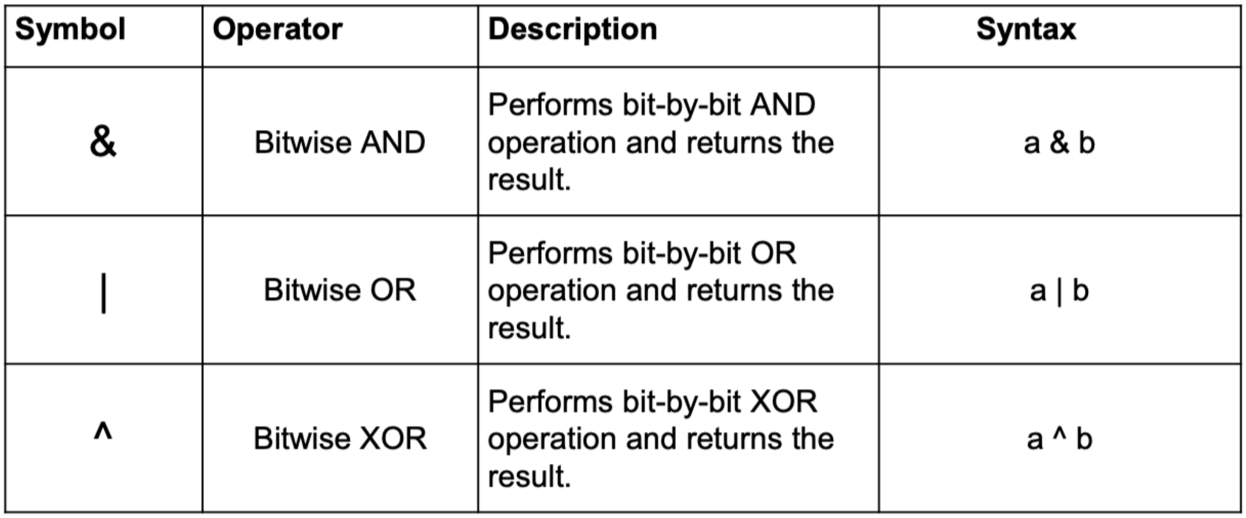
Symbol	Description	Syntax
+	Adds two numeric values.	a + b
–	Subtracts two numeric values	a – b
*	Multiply two numeric values.	a * b
/	Divide two numeric values.	a / b
%	Returns the remainder after division .	a % b

Symbol	Operator	Description	Syntax
+	Unary Plus	Used to specify the positive values.	+a
–	Unary Minus	Flips the sign of the value.	-a
++	Increment	Increases the value of the operand by 1.	a++ or ++a
--	Decrement	Decreases the value of the operand by 1.	a-- or --a

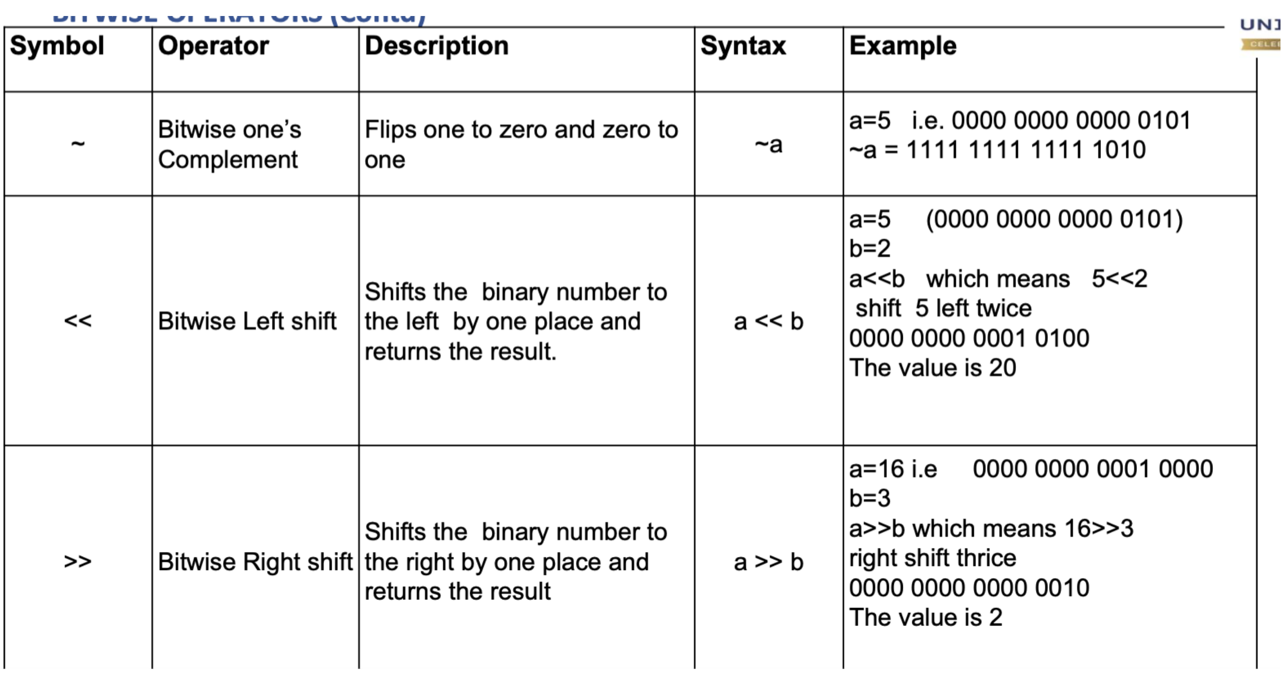
Symbol	Operator	Description	Syntax
<	Less than	Returns true if the left operand is less than the right operand else returns false	a < b
>	Greater than	Returns true if the left operand is greater than the right operand else returns false	a > b
<=	Less than or equal to	Returns true if the left operand is less than or equal to the right operand else returns false	a <= b

Symbol	Operator	Description	Syntax
>=	Greater than or equal to	Returns true if the left operand is greater than or equal to right operand. else returns false	a >= b
==	Equal to	Returns true if both the operands are equal.	a == b
!=	Not equal to	Returns true if both the operands are NOT equal.	a != b

Symbol	Operator	Description	Syntax
&&	Logical AND	Returns true if both the operands are true.	a && b
||	Logical OR	Returns true if both or any of the operand is true.	a || b
!	Logical NOT	Returns true if the operand is false.	!a



Symbol	Operator	Description	Syntax
&	Bitwise AND	Performs bit-by-bit AND operation and returns the result.	a & b
|	Bitwise OR	Performs bit-by-bit OR operation and returns the result.	a | b
^	Bitwise XOR	Performs bit-by-bit XOR operation and returns the result.	a ^ b



BITWISE OPERATORS (Contd)

Symbol	Operator	Description	Syntax	Example
~	Bitwise one's Complement	Flips one to zero and zero to one	~a	a=5 i.e. 0000 0000 0000 0101 ~a = 1111 1111 1111 1010
<<	Bitwise Left shift	Shifts the binary number to the left by one place and returns the result.	a << b	a=5 (0000 0000 0000 0101) b=2 a<<b which means 5<<2 shift 5 left twice 0000 0000 0001 0100 The value is 20
>>	Bitwise Right shift	Shifts the binary number to the right by one place and returns the result	a >> b	a=16 i.e 0000 0000 0001 0000 b=3 a>>b which means 16>>3 right shift thrice 0000 0000 0000 0010 The value is 2

Few inputs on carrying out arithmetic operation using bit wise operators:

1. n << 1 can be used to multiply variable n by 2

2. n & 1 can be used to check whether n is odd

3.To swap two variables a and b, the following code can be used

```
a = a ^ b
b = a ^ b
a = a ^ b
```

4. To check whether i^{th} bit in a variable n is 1

n & (1 << i) : 0 implies not set and non 0 implies set

5.To set the i^{th} bit in n

```
n = n | 1 << i
```

6.To clear the i^{th} bit in n

```
n = n & ~(1 << i)
```

Symbol	Operator	Description	Syntax
=	Simple Assignment	Assign the value of the right operand to the left operand.	a = b
+=	Plus and assign	Add the right operand and left operand and assign this value to the left operand.	a += b
-=	Minus and assign	Subtract the right operand and left operand and assign this value to the left operand.	a -= b

Symbol	Operator	Description	Syntax
*=	Multiply and assign	Multiply the right operand and left operand and assign this value to the left operand.	a *= b
/=	Divide and assign	Divide the left operand with the right operand and assign this value to the left operand.	a /= b
%=	Modulus and assign	Division of left operand by the right operand and assign the remainder to the left operand.	a %= b

Symbol	Operator	Description	Syntax

sizeof	sizeof	A compile-time unary operator which can be used to compute the size of its operand	sizeof(variable)
			sizeof(type)

,	Comma Operator	Used to string together several expression. The result of the rightmost expression becomes the value of the total comma-separated expression.	b= (a=3,a+1); First assigns 3 to a and assigns 4 to b.

Symbol	Operator	Description	Example

? :	Ternary	**Syntax:** (Expression1) ? Expression2 : Expression3
		If Expression1 evaluates to true then Expression 2 is returned else Expression3 is returned
			Example: (usage)
			int a= 10; int b = 20;
			printf("%d\n",(a>b)?a:b);
			Outputs 20

Expression

- An expression consists of Operands and Operators Eg: 6+2, a+b, a+b*c
- Evaluation of Operands: Order is not defined
- The precedence of operators is the order in which the operators are evaluated in the expression.
- The operator precedence determines the priority to be given to a particular operator.
- Operator associativity is the order in which operators are evaluated when operators have same precedence.
  associativity and precedence

Sequence point Operation

- It is any point in the execution of a program which guarantees or ensures that all the side effects of the previous evaluation of the program's code are successfully performed.
- The term "side effects" is nothing but the changes done by any sort of function or expression where the state of something gets changed.
- Sequence point ensures that none of the alterations or side effects of the subsequent evaluations is yet to be performed at all.
  Some of the basic sequence points available in C are: Logical AND, OR, conditional operator, comma operator etc.
  Example of a sequence point (&&)
  a++ == 10 && a == 11 // will be true.
  Initially the value a++ is 10 and it becomes 11 before the evaluation moves across the sequence point &&.
  example:

```
#include<stdio.h>

int main()

{

int a = 11;

printf("%d",a++==1||a==12); // outputs 1

printf("\n%d",a); // outputs 12
return 0;

}
```

output:

```
1
12
```

selection structures

if

example:

```
#include <stdio.h>

int main()

{

int x = 20;

int y = 18;

if (x > y)

printf("x is greater than y");

return 0;

}
```

output:

```
x is greater than y
```

if else

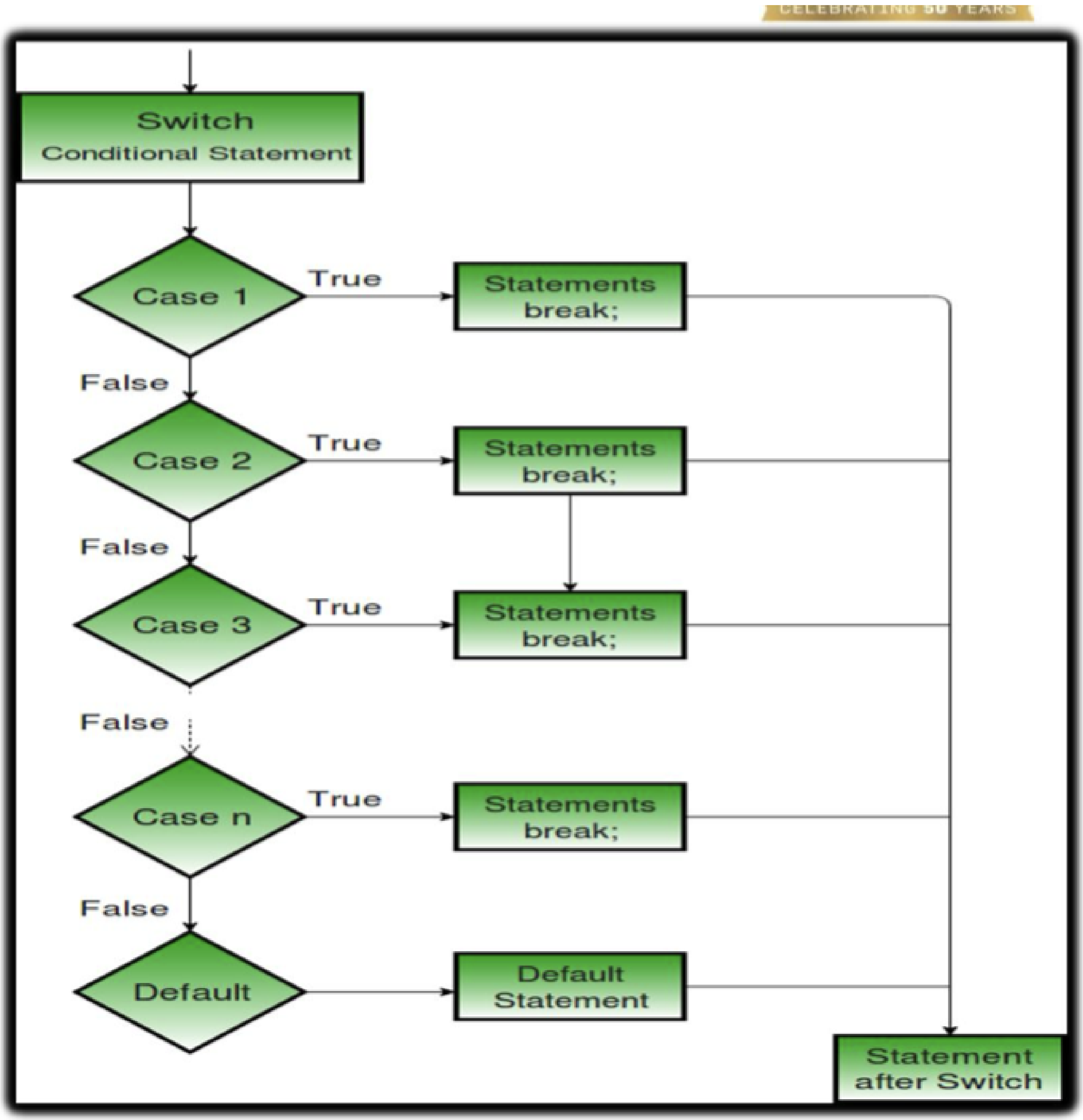
example:

```
#include<stdio.h>

int main()

{

int time = 22;

if (time<10) {

printf("Good morning.");

}

else if (time < 20) {

printf("Good afternoon");

}

else {

printf( "Good evening.");

}

return 0;

}
```

output:

```
Good evening.
```

switch case



example1:

```
#include<stdio.h>

int main()

{

int day = 4;
```

```
switch (day){

case 1:

printf ("Monday") ;

break;

case 2:

printf("Tuesday");

break;

case 3:

printf ("Wednesday");

break;

case 4:

printf("Thursday");

break;

case 5:

printf("Friday");

break;

case 6:

printf ("Saturday");

break;

case 7:
```

```
printf ("Sunday");

break;

}

return 0;

}
```

output:

```
Thursday
```

do while loop

language specifications

programs on control structures

Coding Example_1: Display all the numbers from n to 0.
version 1:Results in infinite loop as n remains to be 5 –>is always true

```
#include <stdio.h>

int main()

{

int n = 5;

while(n)

printf("%d ", n);

n= n –1;//control will never execute this statement

return 0;
}
```

version 2: Displays 5 4 3 2 1. But the program layout does not indicate the logic structure of the program. Always indent the code to indicate the logic structure

```
#include <stdio.h>

int main()

{

int n = 5; while(n){

printf("%d ", n);

           n = n -1;}

return 0;

}
```

version 3:

```
#include <stdio.h>

int main()

{

int n = 5;

while(n)

{

printf("%d ", n); // output : 5 4 3 2 1.

n = n -1;

}
return 0;
}
```

Find the sum of numbers from 1 to n.

code:

```
#include <stdio.h>

int main() {
int n;
printf("Enter a positive integer: ");
scanf("%d", &n); // Read n from the user

int i = 1; // Initialization
int sum = 0;

while (i <= n) { // Condition
sum += i++; // Equivalent to sum = sum + i
// Modification
}
printf("Sum of first %d natural numbers = %d\n", n, sum);



return 0;

}
```

output:

```
Enter a positive integer: 5
Sum of first 5 natural numbers = 15
```

Classify triangles given the 3 sides as equilateral, isosceles or scalene. This is one possible solution.

code:

```
#include<stdio.h>

int main(){

int a,b,c;

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

if(((a+b)>c)||(a+c)>b||(b+c)>a){

if((a==b)&&(b==c)){

printf("equilateral");}

else if((a!=b)&&(b!=c)&&(c!=a)){

printf("scalene");}

else{
```

```
printf("isosceles");}

}

else{

printf("triangle not possible");

}

return 0;

}
```

output:

```
3 6 7
scalene
4 4 4
equilateral
5 6 5
isosceles
```

using switch case:

```
#include<stdio.h>
int main(){

printf("enter three sides of a triangle\n");

int a,b,c;

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

int count = 0;

if(a == b) count++;

if(b == c) count++;
if(c == a) count++;

switch(count) {

case 1:printf("iso\n");

break;

case 3:printf("equi\n");

break;

case 0:printf("scalene\n");

break;

default:printf("not a valid count\n4");

break;

}

printf("switch ended\n");

return 0;

}
```

Find whether the entered character is a vowel or not.

my code:

```
#include<stdio.h>
int main(){

char a[5]={'a','e','i','o','u'};

int count=0;
char b;

scanf("%c",&b);

for(int i=0;i<5;i++){

if(a[i]==b){

count++;

}

break;

}

if(count>0){

printf("the given character is a vowel");

}

else{

printf("not an vowel");

}

return 0;

}
```

alternate code:

```
#include<stdio.h>

char lowercase(char ch){

if (ch >= 'A' && ch <= 'Z') {
```

```
// If the character is uppercase, convert it to lowercase

return ch + ('a' – 'A');

} else {

// If the character is already lowercase or not a letter, return it
unchanged

return ch;}

}

int main()

{

printf("enter the character\n");

char ch;

ch = getchar();

ch=lowercase(ch);

//scanf("%c",&ch);

// solution using if

/*if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {

printf("you entered an vowel\n");

}

else{

printf("you did not enter the vowel\n");
```

```
}*/

// Solution using switch case

switch(ch) {

//case 'a' || 'A': printf("hello 1"); break; // output of expression will be the case label case 'a':

case 'a': printf("you entered an vowel\n");

break;

case 'e': printf("you entered an vowel\n");

break;

case 'i':printf("you entered an vowel\n");

break;

case 'o': printf("you entered an vowel\n");

break;

case 'u': printf("you entered an vowel\n");

break;

default: printf("you did not enter the vowel\n");

}

// if(ch == 'a' || 'e'|| ) // Logical error return 0;

}
```

Generate all Armstrong numbers between 0 and 999.

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

int main() {
int a, b;
scanf("%d %d", &a, &b);

for (int j = a; j < b + 1; j++) {
int i = j;
int result = 0; // Reset result for each number

while (i != 0) {
int remainder = i % 10;
result += pow(remainder, 3);
i /= 10;
}

if (result == j) {
printf("%d ", j); // Print the Armstrong number
}
}



return 0;

}
```

output:

```
0
1000
0 1 153 370 371 407
```

alternate code:

```
#include<stdio.h>

int main()

{

int h,t,u,hc,tc,uc;

int number;

for(h = 0; h<10;h++)

{

for(t = 0; t< 10;t++)

{

for(u = 0;u< 10;u++)

{

hc = h*h*h;
tc = t*t*t;

uc = u*u*u;

int sum = hc+tc+uc;

number = h*100 + t*10 + u*1;

if(number == sum)

printf("%d\n",number);

}

}

}

return 0;

}
```

You are transporting some boxes through a tunnel whose height is only 41 feet. Given the length, width, and height of the box, calculate the volume of those boxes and check if they pass through the tunnel.

Ex: Input = 5, 5, 5 Output = 125
Input = 1, 2, 40 Output = 80
Input = 10, 5, 41 Output = Can’t Pass!

```
#include<stdio.h>

int main(){

int l,b,h;

scanf("%d%d%d",&l,&b,&h);
if(h<41){

printf("%d",l*b*h);

}

else{

printf("cant pass");

}

return 0;

}
```

(imp) Given the number of rows, print a hollow diamond using star symbol: Ex: Input = 5

“Screenshot 2024-03-09 at 8.56.17 PM.png” could not be found.

code:

```
#include<stdio.h>

int main() {
    int n;
    printf("Enter number of rows: ");
    scanf("%d", &n);

    // Upper half of the diamond
    for(int i = 1; i <= n; i++) {
        for(int j = i; j <= n; j++)
            printf(" ");

        for(int k = 1; k <= 2 * i - 1; k++) {
            if(k == 1 || k == (2 * i - 1))
                printf("*");
            else
                printf(" ");
        }

        printf("\n");
    }

    // Lower half of the diamond
    for(int i = n - 1; i >= 1; i--) {
        for(int j = n; j >= i; j--)
            printf(" ");

        for(int k = 1; k <= 2 * i - 1; k++) {
            if(k == 1 || k == 2 * i - 1)
                printf("*");
            else
                printf(" ");
        }

        printf("\n");
    }

    return 0;
}
```

output:

```
Enter number of rows: 5
     *
    * *
   *   *
  *     *
 *       *
  *     *
   *   *
    * *
     *
```

Check whether the given number is divisible by the sum of its digits. Display appropriate message (Divisible or Not Divisible)

```
#include<stdio.h>

int main(){

int n;

scanf("%d",&n);

int m=n;

int result=0;

while(n!=0){

int remainder=n%10;

result=result+remainder;

n=n/10;

}

if (m%result==0){

printf("divisible");

}

else{

printf("not divisible");

}

return 0;
}
```

output:

```
20
divisible
```

Write a C program to find the eligibility of admission for a professional course based on the following criteria: Eligibility Criteria : Marks in Maths >=65 and Marks in Physics >=50 and Marks in Chemistry>=55 and Total in all three subjects >=190 or Total in Maths and Physics >=140

```
#include<stdio.h>

int main(){

int math,phy,chem;

scanf("%d%d%d",&math,&phy,&chem);

int total=math+phy+chem;

if((math>=65&&phy>=50&&chem>=55&&total>=190)||((math+phy>=140)))

printf("eligible");

else

printf("not eligible");

return 0;

}
```

output:

```
100
100
100
eligible
```

Write a C program to print the prime numbers within the given input range. Ex. if user gives 10 and 50 as the input range, then the program must display all the prime numbers between the range 10 and 50 (i.e. 11 13 17 19 23 29 31 37 41 43 47)

code:

```
#include<stdio.h>
#include<math.h>
int main(){
    int a,b;
    scanf("%d%d",&a,&b);
    for(int i=a;i<=b;i++){
        int temp1=ceil(sqrt(i));
        int p=i;
        int is_prime = 1; // Assume i is prime initially
        for(int j=2;j<=temp1;j++){
            if(p%j==0){
                is_prime = 0; // Set is_prime to 0 if i is divisible by j
                break;
            }
        }
        if(is_prime && p > 1){
            printf("%d ",p);
        }
    }
    return 0;
}
```

alternate code:

```
#include <stdio.h>

int main() {
    int low, high, i, flag;
    printf("Enter two numbers(intervals): ");
    scanf("%d %d", &low, &high);
    printf("Prime numbers between %d and %d are: ", low, high);
    while (low < high) {
        flag = 0;
        for(i = 2; i <= low/2; ++i) {
            if(low % i == 0) {
                flag = 1;
                break;
            }
        }
        if (flag == 0)
            printf("%d ", low);
        ++low;
    }
    return 0;
}
```

(imp)Construct a menu-based calculator to perform arithmetic operations (Add, Subtract, Multiply,Divide) on complex numbers

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

int main() {
    int choice, a, b, c, d;
    do {
        printf("1. Addition\n");
        printf("2. Subtraction\n");
        printf("3. Multiplication\n");
        printf("4. Divide\n");
        printf("Enter your choice\n");
        scanf("%d", &choice);
        if (choice > 4 || choice < 1)
            exit(0); // terminates the program

        printf("Enter a and b where a + ib is the first complex number.");
        scanf("%d %d", &a, &b);
        printf("Enter c and d where c + id is the second complex number.");
        scanf("%d %d", &c, &d);

        int real, img;
        switch(choice) {
            case 1:
                real = a+c;
                img = b+d;
                if (img >= 0)
                    printf("Sum = %d + %di\n", real, img);
                else
                    printf("Sum = %d%di\n", real, img);
                break;
            case 2:
                real = a-c;
                img = b-d;
                if (img >= 0)
                    printf("Difference = %d + %di\n", real, img);
                else
                    printf("Difference = %d %di\n", real, img);
                break;
            case 3:
                real = a*c - b*d;
                img = b*c + a*d;
                if (img >= 0)
                    printf("Product = %d + %di\n", real, img);
                else
                    printf("Product = %d %di\n", real, img);
                break;
            case 4:
                if (c == 0 && d == 0)
                    printf("Division by 0 + 0i isn't allowed.");
                else {
                    int x = a*c + b*d;
                    int y = b*c - a*d;
                    int z = c*c + d*d;
                    if (x%z == 0 && y%z == 0) {
                        if (y/z >= 0)
                            printf("Division of the complex numbers = %d +
%di", x/z, y/z);
                        else
                            printf("Division of the complex numbers = %d
%di", x/z, y/z);
                    } else if (x%z == 0 && y%z != 0) {
                        if (y/z >= 0)
                            printf("Division of two complex numbers = %d +
%d/%di", x/z, y, z);
                        else
                            printf("Division of two complex numbers = %d
%d/%di", x/z, y, z);
                    } else if (x%z != 0 && y%z == 0) {
                        if (y/z >= 0)
                            printf("Division  = %d/%d + %di", x, z, y/z);
                        else
                            printf("Division = %d %d/%di", x, z, y/z);
                    } else {
                        if (y/z >= 0)
                            printf("Division = %d/%d + %d/%di",x, z, y, z);
                        else
                            printf("Division = %d/%d %d/%di", x, z, y, z);
                    }
                }
                break;
        }
    } while(choice <= 4);

    return 0;
}
```

(imp)prime numbers

```
#include<stdio.h>

int main(){

int i,n,f=0;

scanf("%d",&n);

for (i=2;i<=n/2;i++){

if(n%i==0){

f=1;

break;

}

}

if(f==0){

printf("prime");

}

else{

printf("not prime");

}

return 0;

}
```

do

Write a program to print all odd numbers between the two limits “m” and “n” (both m and n is given as an integer inputs by the user and m<n) and print all odd numbers excluding those which are divisible by 3 and 5.

Implement a converter program to obtain an integer from a hexadecimal byte.

Given a number N, check whether the nth bit of a number, N is set to 1 or not. Input the N value from the user.

to print pattern

```
#include<stdio.h>



int main()



{

char star='*';

int i,j;

int r=5,c=5;

for(i=0;i<r;i++){

   for(j=0;j<i+1;j++){

       printf("%c",star);

      }

     printf("\n");

}



return 0;



}
```

output

```
*
**
***
****
*****
```

Given a number N, check whether it is a palindrome numbers or not. Ex. 121 , 3443 are palindromes whereas 123,4531 are not plaindromes

```
#include<stdio.h>

int main(){

int n,i;

scanf("%d",&n);

i=n;

int result=0;

while(n!=0){

int remainder=n%10;
result=result*10+remainder;

n=n/10;

}

if(result==i){

printf("palindrome");

}

else{

printf("not an palindrome");

}

return 0;



}
```

to print a pattern

```
#include<stdio.h>

int main(){

int i,j;

int r=5;

for(i=1;i<=5;i++){

for(j=1;j<i+1;j++){
```

```
printf("%d ",j);

}

printf("\n");

}

return 0;



}
```

output

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

gcd of two numbers

```
#include<stdio.h>

int gcd(int a,int b);

int main(){

int a,b;

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

int ans=gcd(a,b);

printf("%d",ans);

return 0;

}

int gcd(int a,int b){

if(b==0){

return a;

}

else{

return gcd(a,b-1);

}

}
```

to find the freq of occurence of a particular character snd display the count

```
#include<stdio.h>

int main(){

char s[100];

scanf("%s\n",s);

int i;

char search;

search=getchar();//scanf("%c",&search);

int count=0;
for (i=0;s[i]!='\0';i++){

if(s[i]==search){

count++;

}

}

if(count==0){

printf("no occurence");

}

else{

printf("occurence is %d times",count);

}

return 0;

}
```

to count character,words,lines{imp}

```
#include <stdio.h>

int main() {
    char s[1000];
    printf("Enter text (press Enter after input):\n");
    fgets(s, sizeof(s), stdin);

    int i = 0;
    int character = 0;
    int words = 0;
    int lines = 0;
    int inWord = 0;  // Flag to track whether currently in a word

    while (s[i] != '\0') {
        // Increment character count
        if (s[i] != '\n') {
            character++;
        }

        // Check for word boundary
        if (s[i] == ' ' || s[i] == '\n' || s[i] == '\t') {
            if (inWord) {
                inWord = 0;  // Exit word
                words++;
            }
        } else {
            inWord = 1;  // Enter word
        }

        // Increment line count
        if (s[i] == '\n') {
            lines++;
        }

        i++;
    }

    // If the last character is not a newline, increment line count
    if (s[i - 1] != '\n') {
        lines++;
    }

    // Increment word count by 1 if the last character is not a space,
newline, or tab
    if (inWord) {
        words++;
    }

    printf("Characters: %d\n", character);
    printf("Words: %d\n", words);
    printf("Lines: %d\n", lines);

    return 0;
}
```

binary to decimal

```
#include<stdio.h>

#include<math.h>

int main(){

int n;

scanf("%d",&n);

int i=0;

int res=0;

while(n!=0){

int r=n%10;

res=res+r*pow(2,i);

n=n/10;

i++;

}

printf("%d",res);

return 0;
}
```