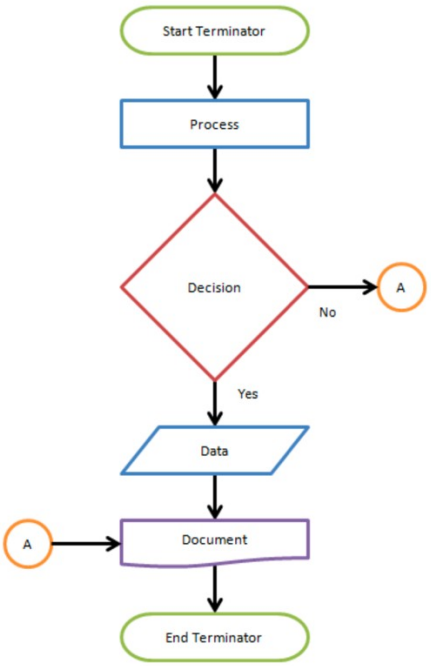
C NOTES (By Varun Sivanesan, First of His Name ;)

Flowchart

Terminator = Start/End (Oval)
Data = Input/ Output (Parallelogram)
Process = Functions (Rectangle)
Decision = if/else or switch (Diamond)



Von Neumann Architecture

CPU (Central Processing Unit) = includes Arithmetic Logic Unit & Control Unit

1) Arithmetic Logical Unit performs Mathematical and Logical Operations
2) Control Unit fetches & interprets instructions to main memory, control transfer of data to and from main memory, control input and output devices, overall supervision of system.

Advantages

1) Control Unit retrieves data and instruction in the same manner from one memory.
2) Design and development of the Control Unit is simplified, cheaper and faster.
3) Data from input / output devices and from memory are retrieved in the same manner.
4) Organization of memory is done by programmers which allow them to utilize the memory's whole capacity.

Disadvantages
1) Parallel Implementation of program is not allowed due to sequential instruction processing
2) Von Neumann bottleneck – instruction can only be carried out one at a time and in sequence
3)Risk of instruction being rewritten due to error in a program

Memory

measured in bits(0 or 1), bytes(8bits) and word(16/32/64bits)

Main Memory = Data and Instruction currently being executed are stored & get erased when the power goes off, is high Speed and measured in mega/gigabytes

Primary Storage RAM (Random Access Memory) = Information typed by user is stored here as Read and Write Memory, Any memory location can be accessed directly without sequential scanning as its random access memory but it is all erased after power failure hence volatile memory

Primary Storage ROM (Read Only Memory) = Permanent & Non Volatile memory and stores programs and basic input-output system programs.

Secondary Memory = Non Volatile memory that is made up of magnetic material and stores information for large time but is low speed.

Cache Memory = High Speed Memory placed between CPU and Main Memory which stored data and instructions that need to be executed. It is costly but less in capacity and can't be accessed by users.

Operating System

1) It is the Interface between Man and Machine

2) It is a integrated collection of programs that it executes which make the computer operational

3) It manages the system resources like memory, processors, I/O devices

Computer Languages

Machine Language = Low Level Language that Consists of 0's and 1's and can be understood by computer easily however it is difficult to debug, understand, not portable and has complex maintenance

Assembly Language = Symbols or Mnemonics are used to represent instructions and is hardware specific

High Level Language = it can be easily interpreted by programmer, easy to debug, understand, easily portable from one location to another and has simple maintenance
Ex – C, C++, C#

Language Translator

Compilers = program that translate entire high level language problem into machine language one at a time Ex – C/C++ Compilers

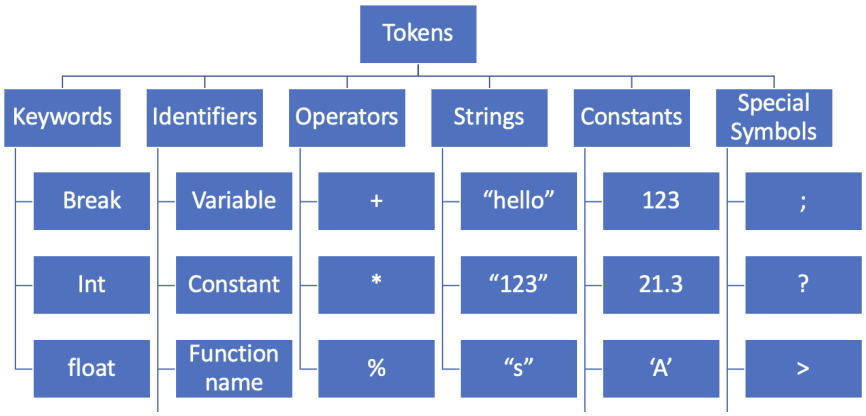
Interpreter = Program which translates one statement at a high-level language program into machine language and executes it Ex – Basic Interpreter, Java Interpreter

Assembler = Program which translates an assembly language into machine language
Ex – Turbo Assembler(TASM), Macro Assembler(MASM)

Compiling Process

1) Creating 2) Preprocessing 3) Compiling 4) Linking 5) Loading 6) Execution

Tokens in C



(Literals are the values that are given to identifiers or variables)
(**** & **\”** can be used to **printf** **** & **“** in code)

Keywords in C

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

```
#include <stdio.h>
Int main(void){                    /*entry point of program ensuring int is returned
though not needed coz here we have void in () which omits return 0; statement*/
Int num;                           /*declaring variable w/ datatype before operating on it*/
scanf(“%d”,&num);                          /*is used to input any value from user*/
printf(“_____%d“,num);                              /*is used to print any text */
}
```

as you can see both printf and scanf aren’t keywords and are formatted I/O functions of **stdio.h library** as they can be used for any datatype chosen by programmer.

whereas getch(), getchar(), puts(), putchar() are unformatted (I/O) functions of **stdio.h library** as they only work on character or string/arrays.

gets() = it is a function that takes a string or character from user as input even if there are spaces in the string for ex: varun sivanesan as input in scanf would only be varun as **scanf() stops at spaces** but gets() would record the blank space as a string and take the whole name under the string.

puts() = prints the string with the newline code(\n) inputted after the string to ensure next output will print on the next line.

(\t for tab space & \n is newline in printf function)

getchar() = it is a function that takes only one character as input and stores it in a variable after user presses enter, even if multiple characters are inputted it will only store first character.

```
char ch;
ch = getchar();          /*here ch will be equal to 2 even if 22 is inputted or can be any alphabet as long as its inputted*/
```

putchar() = it is a function that printing exactly one character.

```
putchar(ch);                              /*prints whatever value ch had*/
```

getc() = it is a function that takes one character but can take that both from keyboard or from a file

```
ch = getc(stdin);                         /*keyboard input*/
ch = getc(fileptr);                       /*input from file*/
```

putc() = it is a function that prints one character & can output to screen or file

```
putc(ch,stdout);                          /*output to screen*/
putc(ch,outfileptr);                      /*output to file*/
```

getch() = it is a function that takes one character as input from user immediately without pressing enter but **doesn't display it on screen** and just records it internally for the program. It is declared in conio.h library header file.

getche() = it is a function that takes one character as input from user immediately without pressing enter & **displays it on screen** after it is inputted. It is also declared in conio.h library header file.

putch() = it is a function that prints exactly one character at the current cursor location. It is also declared in conio.h library header file.

#define = is a preprocessor that declares **defined constant** variables outside of int main()
#define PI 3.14 or #define X 69 are methods to declare global constants.

const = is a datatype dependent method to declare **memory constants** inside int main()
const double Cpi = 3.14;

Size and Range of Data Types (16Bit)

char	1 byte	-128 to +127
signed char	1 byte	-128 to +127
unsigned char	1 byte	0 to +255
int	2 bytes / 4 bytes (64)	-32768 to +32767
signed int	2 bytes / 4 bytes (64)	-32768 to +32767
signed short int	2 bytes / 4 bytes (64)	-32768 to +32767
short	2 bytes	-32768 to +32767
short int	2 bytes	-32768 to +32767
signed short	2 bytes	-32768 to +32767
unsigned int	2 bytes / 4 bytes (64)	0 to +65535
unsigned short	2 bytes	0 to +65535
unsigned short int	2 bytes	0 to +65535
long int	4 bytes / 8 bytes (64)	-2,147,483,648 to +2,147,483,647
signed long int	4 bytes / 8 bytes (64)	-2,147,483,648 to +2,147,483,647
unsigned long int	4 bytes / 8 bytes (64)	0 to +4,294,967,295
float	4 bytes	loses precision after 7 digits for before & after decimal point)
double	8 bytes	loses precision after 15 digits
long double	10 bytes / 16 bytes (64)	

1) char, variations of short, float, double have same storage space from 16bit to 64bit computer systems.

2) float and double are used for decimal numbers instead of integers and always print 6 digits after decimals for %f or %lf unless changed with %.xf even if they are imprecise.

3) sizeof() returns the size of bytes of the datatype of the variable in which it was declared so char and double will always be 1 & 8 bytes respectively.

sizeof(a+b) where a is a int and b is a double is undefined but if **b is long int** then answer would be **8 bytes** as int is converted into long int

4) each set of byte storage for a data type has specific range for its values so when int has 4 bytes in 64 bit compiler we just have to correspond the new range for int to that of 4 bytes.

Format Specifiers (64 bit)

%hd = short signed integer (2bytes)
%d = signed integer (4bytes)

%i = same as %d but if the input is as 0x____ & 0____ it will store data as hexadecimal or octal number respectively

for example
0x70 in **scanf("%i",&a)** will **printf("%d",a)** as 112 (70 is 112 in hexadecimal system) and similarly 070 will give 56 (70 is 56 in octal system)

%u = unsigned integer (4bytes)
%ld = long signed integer (8bytes)

%o = octal unsigned integer

%x = hexadecimal unsigned but alphabetical characters are lowercase
%X = hexadecimal unsigned but alphabetical characters are uppercase

%c = unsigned character

For characters, ASCII codes for alphabets are (A=65, Z=90, a=97, z=122, 0=48, 9=57)

Example -

```
char letter;
letter = 'A';                    /*" " used for strings)*/
letter = 65;                     /*both mean same thing)*/
```

assignment suppression = when inputting dates as xx-yy-zzzz , scanf("%d-%d-%d",&d,&m,&y); can be used to ignore slashes but if the dates are inputted as xx/yy/zzzz the dashes won't be eliminated.

So assignment suppression ability of %c specifier can be used to input dates scanf("%d*c%d*c%d",&d,&m,&y); will ignore the dash or slash characters

%s = string **char string[] = "this is a string";** or **const char *stringptr = "this is also a string";** will both be representated by %s as **printf("%s",string** or **stringptr);**

%f = decimal floating-point specifier with 7 digits of precision after decimal.
It will always print 6 digits after decimal unless %.xf is used for x digits after decimal.

It rounds up the value if the number after decimal precision is more than 5
so, for 1.2345678, since default decimal precision is 6 and 7th digit after decimal is 8, %f will print 1.234568.

%lf = decimal double point value with 15 digits of precision after decimal.
It will always print with 6 digits after decimal unless %.xlf is used to ensure x digits after decimal point.

same rounding rules apply as %f

%e = converts numbers into ___e+0X & has 6 digits of precision after decimal.
It will always print 6 digits after decimal unless %.xe is used to ensure x digits after decimal point.
so for 150.459 the data is printed as 1.50459e+02

same rounding up rules apply as %f

%E = same as %e except the output has capital E instead of e.

%g = it removes all succeeding zeroes after decimal point and can print the number either as %f form or %e form

in %.xg the x is representing number of total significant digits in the number so 1.234 at %.2g will give 1.2 and not 1.23.

same rounding up rules apply as %f based on precision after decimal applied

if it's a very small number (e-05 to e-infinity) then it is printed in scientific form so 0.0000123456 will be printed as 1.23456e-05 but 0.000123456 will be printed as 0.000123456

if it's a very big number (e+06 to e+infinity) then it is printed in scientific form so 123450 will be printed as 123450 but 1234500 will be printed as 1.2345e+06

in %f form only 6 significant digits are printed so 1234.567 will be printed as 1234.56 and 123456.78 will only print 123456 and any larger number would use scientific notation

if the significant digit precision demanded doesn't cover all number before the decimal then scientific notation is used so 123.45 in %.2g will give 1.2e+02 but in %.3g will give %f form which is 123

%G = same as %g except the output for scientific notation will have capital E.

Field Width = is the restriction of space for the output unless the output is larger than width represented by %xf

```
printf( "%4d\n", 1 );
printf( "%4d\n", 12 );
printf( "%4d\n", 123 );
printf( "%4d\n", 1234 );
printf( "%4d\n\n", 12345 );

printf( "%4d\n", -1 );
printf( "%4d\n", -12 );
printf( "%4d\n", -123 );
printf( "%4d\n", -1234 );
printf( "%4d\n", -12345 );
```

```
   1
  12
 123
1234
12345

  -1
 -12
-123
-1234
-12345
```

Field width can be used in scanf() to select only certain characters from input stream

scanf("%2d%d", &x, &y); for input 123456 will give x = 12 and y = 3456.

Precision = It is accuracy of a value inputted or outputted (represented by %.xf %.xg or %.xs for %.xe & %.xf & %.xlf, it is the digits after decimal to be printed or inputted.

for %.xg, it is the significant digits in the number to be printed or inputted.

For %.xs, it is the number of characters to be printed or inputted in the string.

%x.yf can be used to input both field width and precision for a dataset

printf("%*.*f",x,y,num); can also be used to input both field width and precision

Operation in C Programming

Increment(++) and Decrement operators (--)

Prefix variation like ++num will instantly increment by one to num and return the value

Postfix variation like num++ will first return the value and then increment by one

so for m=5 and y=++m, both will be 5 in value but in m=5 and y=m++ , y will become 6

also ++(x+1) will give syntax error as x+1 isnt a modifiable variable

Relational Operators

Operator	Meaning
==	Is equal to
!=	Is not equal to
<	Is less than
<=	Is less or equal
>	Is greater than
>=	Is greater or equal

The relational operators will give value 1, if the statement is TRUE
& 0 if the statement is FALSE

Logical Operators

Operator	Symbol	Example
AND	&&	expression1 && expression2
OR	||	expression1 || expression2
NOT	!	!expression1

The result of logical operator is always 0 for false and 1 for true (whereas bitwise operator return actual integral value)

They consider every nonzero value as 1 so 2&&5 is true or 1 as both 1&&1 is true.
For example, **result = (5==5) && (10>5);** will be **result = 1** as both statements are true

Assignment Operator

a = a + 10 can be written as a += 10 & a = a / 10 can be written as a /= 10

Bitwise Operators

&	**Bitwise AND**
|	**Bitwise OR**
^	**Bitwise XOR**
~	**Bitwise complement**

in bitwise operators the numbers are operated on their binary forms and then output is converted back to decimal

for example, 12 & 25 will be the AND GATE for 00001100 (12 in binary) and 00011001 (25 in binary) which is 00001000 (8 in binary) by applying and gate on each position of binary number and the answer is returned as 8.

also x&1 would be zero if x is even

Bitwise Shift Operators

x<<n will add n zeroes from right

for 0000 0000 0000 1011

✓x<<1 0000 0000 0001 0110 ⬅ Add ZEROS

✓x<<3 0000 0000 0101 1000 ⬅ Add ZEROS

x<<1 is same as x*2 in decimal format so 10<<1 should give 20

x>>n will add n zeroes from left

for 0000 0000 0000 1011

✓x>>1 Add ZEROS ➡ 0000 0000 0000 0101

✓x>>2 Add Z EROS ➡ 0000 0000 0000 0010

x>>1 is same as x/2 in decimal format so 10>>1 should give 5

Bitwise Complement Operators

It will flip all values in the binary form and return in decimal form

x=100110001000111

~x=011001110111000 (complement)

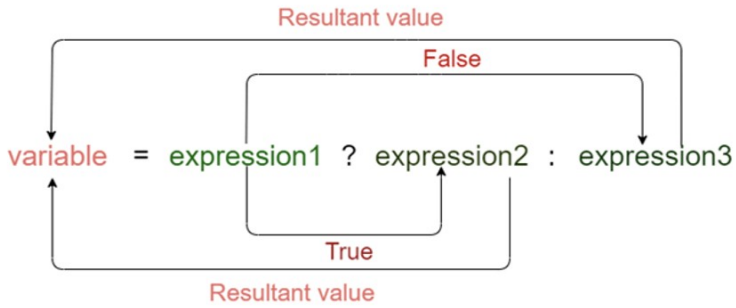
If x = 8 then ~x = -9

bitwise complement on small numbers leads to large values that when stored in signed specifier may give negative value so when x=1 the **%d for x~** will be **-2** but **%ud for x~** will be **4294967294** as unsigned format will have enough range for it.

Conditional Operators

Decision making operator in the format of

(expression1)? (expression2): (expression3);



so for example

(age>=18)? (printf("eligible for voting")) : (printf("not eligible for voting"));

will print eligible for voting only if the age is more than or equal to 18

Comma Operator

```
int a = (5,10);          /* will assign 10 to a*/
int a = 5,10;            /* error as its wrong syntax*/
int a; a=5,10;           /*will assign 5 to a as "=" precedes "," in importance*/
```

Precedence Order Of Operators

Precedence	Operator	Description	Associativity
1 highest	::	Scope resolution	None
2	++ -- () [] . ->	Suffix increment Suffix decrement Parentheses (Function call) Brackets (Array subscripting) Element selection by reference Element selection through pointer	Left-to-right
3	++ -- + - ! ~ (*type*) * & sizeof	Prefix increment Prefix decrement Unary plus Unary minus Logical NOT Bitwise NOT (One's Complement) Type cast Indirection (dereference) Address-of Size-of	Right-to-left
4	.* ->*	Pointer to member Pointer to member	Left-to-right
5	* / %	Multiplication Division Modulo (remainder)	Left-to-right
6	+ -	Addition Subtraction	Left-to-right
7	<< >>	Bitwise left shift Bitwise right shift	Left-to-right
8	< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left-to-right
9	== !=	Equal to Not equal to	Left-to-right
10	&	Bitwise AND	Left-to-right
11	^	Bitwise XOR (exclusive or)	Left-to-right
12	|	Bitwise OR (inclusive or)	Left-to-right
13	&&	Logical AND	Left-to-right
14	||	Logical OR	Left-to-right
15	?:	Ternary conditional	Right-to-left
16	= += -= *= /= %= <<= >>= &= ^= |=	Direct assignment Assignment by sum Assignment by difference Assignment by product Assignment by quotient Assignment by remainder Assignment by bitwise left shift Assignment by bitwise right shift Assignment by bitwise AND Assignment by bitwise XOR Assignment by bitwise OR	Right-to-left
17 lowest	,	Comma	Left-to-right

2*((i/5)+(4*(j-3))%(i+j-2)) **i→8, j→5**

2*((8/5)+(4*(5-3))%(8+5-2))

2*(1+(4*2)%11)

2*(1+8%11)

2*(1+8)

2*9

18

Type Conversion

when operating between data of different types, all the data is converted into largest datatype **(implicit conversion)**

```
bool -> char -> short int -> int -> unsigned int -> long ->
unsigned -> long long -> float -> double -> long double
```

and for forced conversions **(explicit conversion)**

float to int causes removal of numbers after decimal
double to float causes rounding up of numbers
for example ;

float a = b + c /* here b is a int and c is a float*/

or a = (float) b/c /*here both b and c are int & division will be done in int mode but answer will be in float*/

or a = (double) b/c /*here both b and c are float & division will be done in float mode but answer will be in double*/

```c
int main()
{
    int x = 10;    // integer x
    char y = 'a';  // character c

    x = x + y;


    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

Here x will be 107 & z will be 108.000000

If else statement

if (x >= 5 && x <= 10) is the correct way to check range and **if (5 <= x <= 10)** is a semantical error due to priority order

WAP to verify if a number is even or odd.

```
if ((x % 2) == 0)                    /*even number give 0 as remainder with  2*/
{
        printf("It is an even number");
}

else
{
        printf("It is an odd number");
}
```

WAP to calculate the absolute value of a integer.

```
if ( number < 0 )          /*a negative number will have -number as absolute value*/
   number = -number;
```

WAP to check if a year is leap year

```
if(year%4 == 0)          /*a leap year needs to be divisible by 4, 100, 400 for it to be a leap year*/
{
   if( year%100 == 0)
   {
        if ( year%400 == 0)
        printf("%d is a leap year",year);

        else
        printf("%d is not a leap year",year);

    }  else printf("%d is a leap year",year);

} else   printf("%d is not a leap year",year);
```

WAP to check the character type inputted.

```
if (ch >= 'a' && ch <= 'z')
                printf("lowercase char\n");
if (ch >= 'A' && ch <= 'Z')
                printf("uppercase char\n");
if (ch >= '0' && ch <= '9')
                printf("digit char\n");
else
                printf(" special char\n");
```

WAP to check the smallest among three numbers

If a>b & (**if** b>c then c is smallest but **else** b<c then b is smallest) **else** (**if** a>c then c is smallest **else** a is smallest)

else if ladder tool

```
if (Expression_1 )
{
    statement _block1
}
else if (Expression_2)
{
    statement _block2
}
.....
else if (Expression_n)
{
    statement _blockn
}
else
{
    last_statement
}
```

- **Find the roots of a quadratic equation** ax^2+bx+c **using** if else**control statements.**
- **Roots of a quadratic equation**

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

If discriminant < 0 ---------- imaginary roots 1+i2.45
else if discriminant = 0 ---------- real equal roots -b/2a
else discriminant > 0 ---------- real unequal roots [-b plus or minus ((discriminant)^-2)]/2a

switch() statement

different method to solve if else like questions with format:

scanf("%d",&a)

switch(a)

```
{
    case x: printf("x is true");      /*if a = x then this statement will run*/
            break;                    /*if break isn't there switch will run continuously till end*/

    case y: printf("y is true");
            break;

    default: printf("default is true");   /*default case will run if no other case is true*/
            break;
}
```

WAP to operate a calculator with switch()

```
scanf("%f %c %f", &value1,&operator,&value2);
switch (operator)
{case '+':                                  /*checks operator*/
result=value1+value2;
printf("%f",result);
break;

case '-':
    result=value1-value2;
printf("%f",result);
break;

case '*':
    result=value1*value2;
printf("%f",result);
break;

case '/':
     if ( value2 == 0 )
printf("Division by zero.\n");              /*division by zero impossible*/
else result=value1 / value2;
printf("%f",result);
break;

default;
printf("unknown operator");}
```

example one

```
switch (115)
{
    case 110 : printf(“A”);
    case 115 : printf(“B”);
    case 125 : printf(“C ”);
    case 135 : printf(“D”);
}
```

will print **B C D** as the code will run till the end of switch case statement

example two

```
int num = 1;
switch(num)
{
    case 1.5 : printf(“1.5”);          /*code wont as 1.5 is not a integer*/
    break;
    case 2: printf(“2”);
    break;
    case A: printf(“A”);               /*this is fine as A will be taken as 65*/
}
```

example three

```
  switch (x)
  {
     x = x + 1;
     case 1: printf("choice is 1");          /*this will get printed as x+=1 inside the switch case isn’t applicable to statement*/
           break;
     case 2: printf("choice is 2");
           break;
     default: printf("choice is other than 1 and 2");
           break;
  }
```

example four

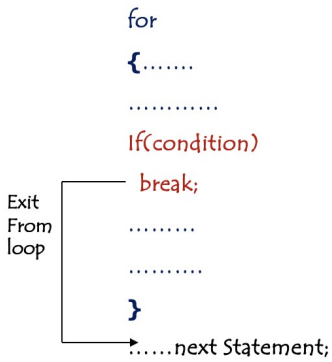
```
  int x = 1;
  switch (x)
  {
     case 2: printf("Choice is 1");
          break;
     case 1+1: printf("Choice is 2");
           break;
  }
  return 0;
```

//compiler error as duplicate cases aren’t allowed

for loop

for (initial expression; loop condition; loop expression)
{
body of loop
}

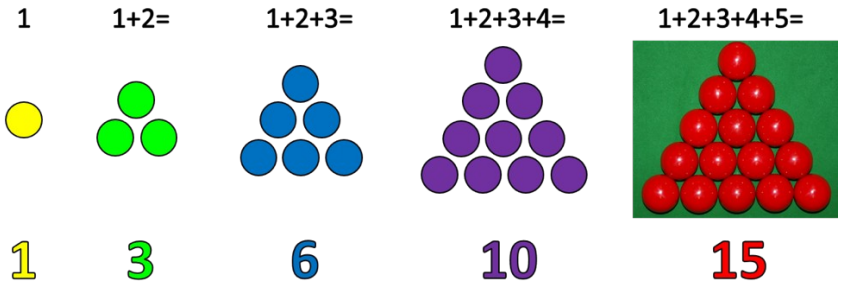
break statement in an if (condition) statement inside a for loop exits the code out of the loop



WAP for sum of first 100 natural numbers using for loop

```
sum=0;
for(n = 1; n <=100; n=n + 1)     /*n is incremented by 1 after every iteration*/
{          /*without curly braces, only first statement is considered part of loop*/
    sum=sum + n;                 /*adds every number till 100 numbers to sum*/
}
```

WAP for write triangular numbers upto n




```
int n, triangularNumber=0;
for ( n = 1; n <= 10; n++ )
{
triangularNumber += n;                    //learn code to print triangular numbers upto n
printf("The %d th triangular number is %d",n,triangularNumber);
}
```

while loop

```
while (expression)
{
    body of loop
}
```

//you can write any while loop as a for loop and vice versa

WAP for sum of first 100 natural numbers using while loop

```
sum=0;
n=1;
while(n<100)
{
    sum+=n                          /*adds every number till 100 to sum*/
    n+=1                            /*n is incremented*/
)
```

WAP for reversing the digits of a number

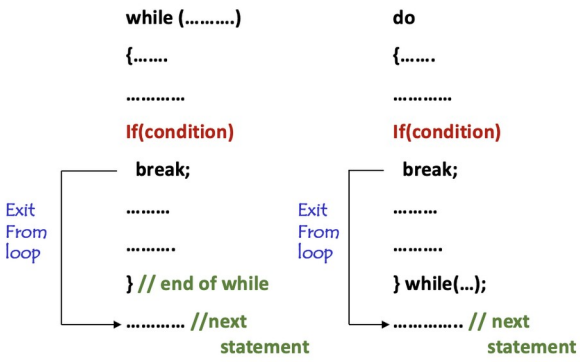
```
int number, rev=0, right_digit;
printf("Enter your number.\n");
scanf("%d",&number);
while ( number != 0 )
{
right_digit = number % 10;          /*unit digit is added to right_digit*/
rev=rev*10 + right_digit;           /*unit digit is added to new reverse with
previous unit digit in tens place */
number = number / 10;               /*unit digit is eliminated to make tens place
digit new unit digit*/
}
printf("The reversed number is %d", rev);
```

do while loop

```
do
{
    body of loop
}
while(expression);        //the loop is executed once regardless of while expression.
```

//exit controlled loop so any while loop can be written as a do while loop if the loop needs to be executed atleast once before termination

break statement inside an if(condition) statement inside a loop makes the code leave the loop even if the loop is a do while loop, the while condition isn't even checked.



nested for-loop

WAP for write nth triangular number.

```
for ( counter = 1; counter <= 5; counter++ )
{
    printf("What triangular number do you want? ");
    scanf("%d",&number);
    for ( n = 1; n <= number; n++ )              //adds subsequent numbers
    {
            triangularNumber = triangularNumber + n;  //1 , 1+2 , 1+2+3 , 1+2+3+4 ……..
            printf("The %d th triangular number is %d:",n-1,triangularNumber);
    }
}
```

WAP for write multiplication tables till "n" for "k" terms

```
scanf("%d %d",&n,&k);

for (i=1; i<=k; i++)
   {
    for (j=1; j<=n; j++)
            {
            prod = i * j;
    printf("%d * %d = %d \t", j,i,prod);
            }
    printf("\n");
  }
```

```
Enter n & k values: 3  5
The table for 3 X 5 is
1 * 1= 1    2 * 1= 2    3 * 1= 3
1 * 2= 2    2 * 2= 4    3 * 2= 6
1 * 3= 3    2 * 3= 6    3 * 3= 9
1 * 4= 4    2 * 4= 8    3 * 4= 12
1 * 5= 5    2 * 5= 10   3 * 5= 15
```

WAP to check if a number is prime or not

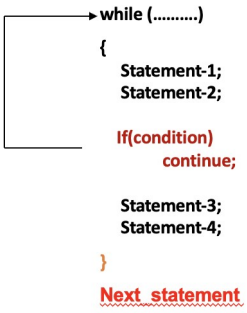
```
int  j, prime=1;
    scanf("%d",&N);
     for( int j=2; j<N; j++ )
    {
                if( (N % j) == 0)                              /*a prime number will never have
                                                          remainder 0 with any number less than it*/
                {
                prime=0;
                break;
                }
     }
     if (prime == 1)
                printf("%d is a prime no",N);
     else
                printf("%d is a not a prime no",N);          //prime will only be 0 if the if condition of
remainder comes true
```

WAP to generate a prime number between 2 numbers

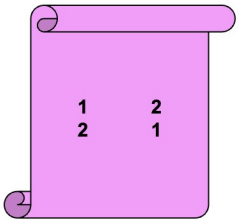
```
scanf("%d %d",&m,&n);
for( int i=m; i<=n; i++)
{
    int prime=1;
    for( int j=2; j<i; j++ )          //same solution as before but checking is done b/w a given range
    {
       if( i % j == 0)
       {
                        prime=0;
                        break;
        }
     }
     if (prime == 1) printf("%d",i);
}
```

continue; statement in loop

continue statement skips the whole loop in its current iterations and starts the loop again in next iteration.




```
for ( i = 1 ; i <= 2 ; i++ )
{
   for ( j = 1 ; j <= 2 ; j++ )
   {
      if ( i == j )
         continue ;

      printf("\n %d\t %d\n",i, j);
   }
}
```



(do all lab questions separately from lab6 to lab12, it would be a wastage of paper to print answers that u have to learn anyways ;)

NUMBER PATTERNS

main.c

```c
#include <stdio.h>
void main()
{
    int i,j,rows;
    printf("Please enter the number of rows=");
    scanf("%d",&rows);
    for(i=1;i<=rows;i++)
    {
        for(j=2*i-1;j>=i;j--)
        {
          printf("%d\t",j);
        }
        printf("\n");
    }
}
```

```
Please enter the number of rows=6
1
3       2
5       4       3
7       6       5       4
9       8       7       6       5
11      10      9       8       7       6
```

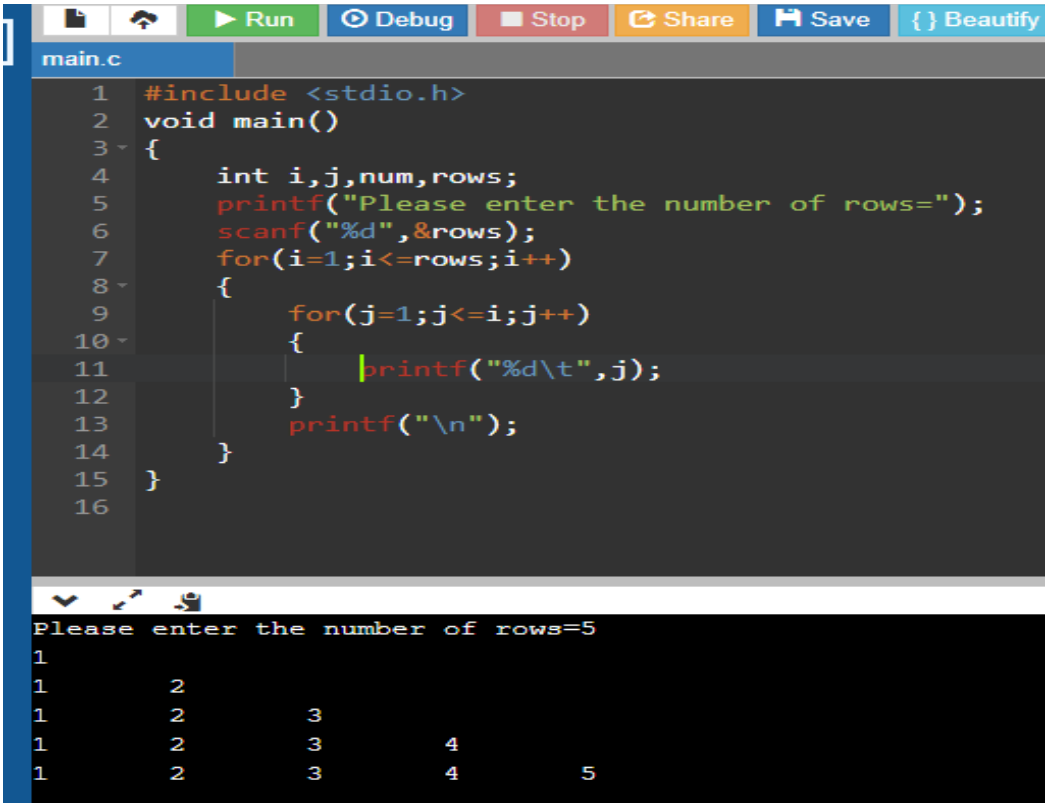
main.c

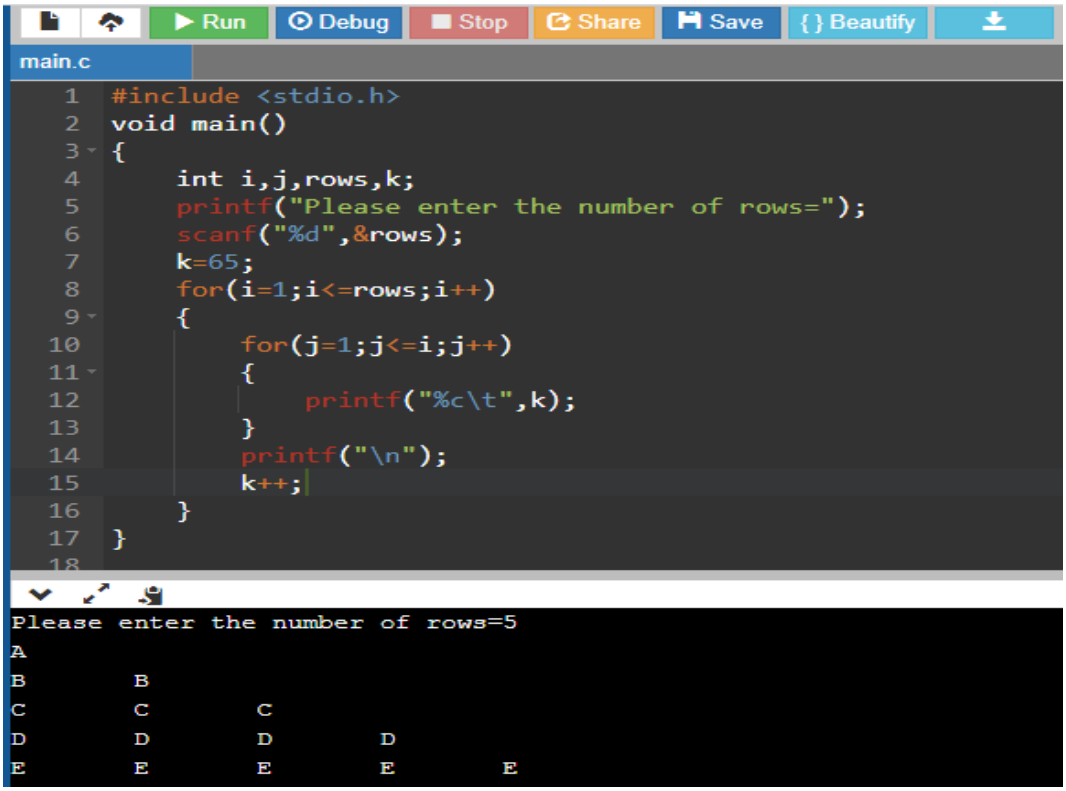
```c
#include <stdio.h>
void main()
{
    int i,j,rows;
    printf("Please enter the number of rows=");
    scanf("%d",&rows);
    for(i=1;i<=rows;i++)
    {
        for(j=i;j>=1;j--)
        {
         printf("%d\t",j);
        }
        printf("\n");
    }
}
```

```
Please enter the number of rows=5
1
2       1
3       2       1
4       3       2       1
5       4       3       2       1
```


```c
#include <stdio.h>
void main()
{
    int i,j,num,rows;
    printf("Please enter the number of rows=");
    scanf("%d",&rows);
    for(i=1;i<=rows;i++)
    {
        num=i;
        for(j=1;j<=i;j++)
        {

            printf("%d\t",num);
            num++;
        }
        printf("\n");
    }
}
```

```
Please enter the number of rows=5
1
2       3
3       4       5
4       5       6       7
5       6       7       8       9
```


Run
Debug
Stop
Share
Save
{ } Beautify
main.c
#include <stdio.h>
void main()
{
int i,j,num,rows;
printf("Please enter the number of rows=");
scanf("%d",&rows);
for(i=1;i<=rows;i++)
{
for(j=1;j<=i;j++)
{
printf("%d\t",j);
}
printf("\n");
}
}
Please enter the number of rows=5
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5


Run
Debug
Stop
Share
Save
{ } Beautify
main.c
#include <stdio.h>
void main()
{
int i,j,rows,k;
printf("Please enter the number of rows=");
scanf("%d",&rows);
k=65;
for(i=1;i<=rows;i++)
{
for(j=1;j<=i;j++)
{
printf("%c\t",k);
}
printf("\n");
k++;
}
}
Please enter the number of rows=5
A
B B
C C C
D D D D
E E E E E

c Copy code

```c
#include <stdio.h>

int main() {
    int rows = 5; // Number of rows in the pattern
    int number = 1; // Starting number in the pattern

    // Loop through each row
    for (int i = 1; i <= rows; i++) {
        // Print numbers for each row
        for (int j = 1; j <= i; j++) {
            printf("%d ", number);
            number++;
        }
        printf("\n");
    }

    return 0;
}
```

Copy code

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

c Copy code

```c
#include <stdio.h>

int main() {
    int n = 5; // Number of rows in the pattern

    // Loop through each row (first half)
    for (int i = 1; i <= n; i++) {
        // Print numbers for each row
        for (int j = 1; j <= i; j++) {
            printf("%d ", i);
        }
        printf("\n");
    }

    // Loop through each row (second half)
    for (int i = n - 1; i >= 1; i--) {
        // Print numbers for each row
        for (int j = 1; j <= i; j++) {
            printf("%d ", i);
        }
        printf("\n");
    }

    return 0;
}
```

Copy code

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

c Copy code

```c
#include <stdio.h>

int main() {
    int rows = 5; // Number of rows in the pattern
    int number = 1; // Starting number in the pattern

    // Loop through each row
    for (int i = 1; i <= rows; i++) {
        // Print numbers for each row
        int currentNumber = number;
        for (int j = 1; j <= i; j++) {
            printf("%d ", currentNumber);
            currentNumber += 2; // Increment by 2 for the next number
        }
        printf("\n");
        number += 3; // Increment by 3 for the starting number of the n
    }

    return 0;
}
```

Copy code

```
1
2 4
3 5 7
6 8 10 12
9 11 13 15 17
```

WAP to print nth Armstrong Number

c Copy code

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

int main() {
    int n;

    printf("Enter the value of n: ");
    scanf("%d", &n);

    if (n <= 0) {
        printf("Please enter a positive integer for n.\n");
    } else {
        int count = 0;
        int number = 0;

        while (count < n) {
            int originalNumber = number;
            int result = 0;

            int digits = (int)log10(number) + 1;

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

            if (result == number) {
                count++;
            }

            number++;
        }

        printf("The %dth Armstrong number is: %d\n", n, number - 1);
    }

    return 0;
}
```

\

WAP to print Fibonacci sequence between a given range of numbers.

```c
#include <stdio.h>

int main() {
    int lowerLimit, upperLimit;

    printf("Enter the lower limit: ");
    scanf("%d", &lowerLimit);

    printf("Enter the upper limit: ");
    scanf("%d", &upperLimit);

    int first = 0, second = 1, next;

    printf("Fibonacci numbers between %d and %d are:\n", lowerLimit, up

    while (first <= upperLimit) {
        if (first >= lowerLimit) {
            printf("%d\n", first);
        }
        next = first + second;
        first = second;
        second = next;
    }

    return 0;
}
```

```mathematica
Enter the lower limit: 0
Enter the upper limit: 50
Fibonacci numbers between 0 and 50 are:
0
1
1
2
3
5
8
13
21
34
```

WAP to convert a number to binary from decimal number system.

c Copy code

```c
#include <stdio.h>

int main() {
    int decimalNumber, remainder, binaryNumber = 0, base = 1;

    printf("Enter a decimal number: ");
    scanf("%d", &decimalNumber);

    // Convert decimal to binary
    while (decimalNumber > 0) {
        remainder = decimalNumber % 2;
        binaryNumber = binaryNumber + remainder * base;
        decimalNumber = decimalNumber / 2;
        base = base * 10;
    }

    printf("Binary equivalent: %d\n", binaryNumber);

    return 0;
}
```

ONE DIMENSIONAL ARRAY

array is declared as **datatype name[size]** for example **int salary[5]**

the numbering of elements in a one dimensional array starts from [0],[1],[2]….[n]

int number[3] = {0,0,0}; and int number[3] = {0}; mean same thing

int num[] = {0,1,2,3,4}; and int num[5]] = {0,1,2,3,4}; mean same thing

array sorting starts from 0 to n for start & -1 to n from end of array

adding elements to an one dimendional array

```
int arr[50]
printf(“no of elements in array”);
scanf(“%d”,&n);

for(int i=0; i<n; i++)
{
    scanf(“%d”,&arr[i]);
}
```

printing elements of an one dimensional array

```
for(int i=0; i<n; i++)
{
    printf(“%d”, arr[i]);
}
```

adding elements of 2 one dimensional arrays

```
if(m==n)                    //m & n are number of elements in both arrays respectively
{
   for(i=0; i<m; i++)
   c[i]=a[i]+b[i];
   printf(“sum of given array elements”);
   for(i=0; i<n; i++)
   printf(“%d”,c[i]);
}
else
printf("cannot add“);
}
```

displaying elements of array in reverse

```
for (i=n-1; i>=0; i--)
printf("%d", a[i]);
```

reversing the elements of original array without creating new array

```
for( i=0, j=n-1; i<n/2; i++, j--)
{
    temp=a[i];
    a[i]=a[j];
    a[j]=temp;
}
```

inserting a element in an array at a specific position

```
printf("enter the element and position of insertion");
scanf("%d%d,&ele,&pos);
for(i=n; i>=pos; i--)                    //shift the elements to right to make space
   a[i]=a[i-1];
a[pos-1] = ele;                          //ele is inserted at  the specified  position
n = n + 1;          // increment the count of no of elements for further printing the array
```

deleting a element from an array

```
printf("enter the position at which the element to be deleted");
scanf("%d",&pos);
for(i=pos-1; i<n-1; i++)
    a[i] =a[i+1];
n = n-1;
for(i=0;i<n;i++)
    printf("%d",a[i]);
```

inserting an element in a sorted array based on the ascending order

```
for(i=0;i<n;i++)
    if (ele<a[i]) break;                 //ele is element needs to be inputted
pos = i+1;
for(i=n; i>=pos; i--)
     a[i]=a[i-1];
a[pos-1] = ele;
n = n + 1;
```

WAP to take an array of ten elements and split it in two equally divided arrays

```c
#include <stdio.h>

int main() {
    int originalArray[10], firstArray[5], secondArray[5];

    // Input: Taking an array of 10 elements
    printf("Enter 10 elements for the array:\n");
    for (int i = 0; i < 10; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &originalArray[i]);
    }

    // Splitting the array into two halves
    for (int i = 0; i < 5; i++) {
        firstArray[i] = originalArray[i];
        secondArray[i] = originalArray[i + 5];
    }
```

1. Write a program to find the largest and smallest element in an array.

sort the array which will make 1st and last elements the answer.

2. Write a program to find the sum of odd index numbers in an array.

sum += a[i] where i%2!=0

3. Write a program to print the subarray that lies between the two indexes.

can be done with splitting of arrays acc to certain coordinates and adding to subarray.

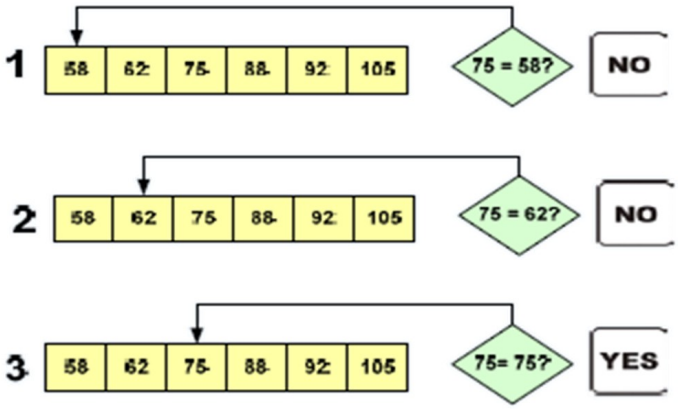
4. Write a program to reverse an array with an auxiliary array.

auxiliary array is another array specifically designed to give the result of the code.

Linear Searching

Is used to search a data item present in a data set of items

It searchs the data one by one by checking equality with each index



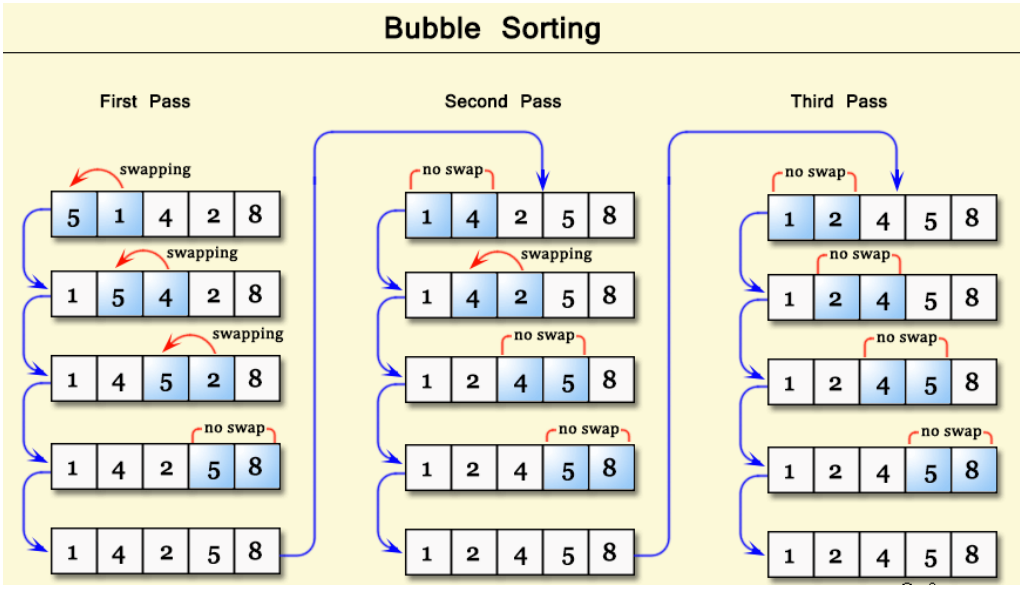
//pseudo code for linear searching

```
printf("enter the element that needs to be searched");
scanf("%d",key);

for(i=0; i<n; i++)
{
    If (a[i]==key)                  //comparison one by one
    {
            found=1;                //to verify if data is there in array
            pos=i+1;                //i+1 will be actual position if key is equal to a[i]
            break;
    }
}

If(found==1)
    printf("data found in %d position",pos);
else
    printf("data is not found");
```

Bubble Sorting

It is a sorting method where values are compared one by one from beginning to end so if the left term is bigger than the right term they are swapped

hence at every iteration of sorting, the largest term will get pushed to the right most position in array



//pseudo code for bubble sort

```
for(i=0; i<n-1; i++)
{
    for(j=0; j<n-i-1; j++)
    {
            if(a[j]>a[j+1])
            {
            temp=a[j];
            a[j]=a[j+1];
            a[j+1]=temp;
            }
    }
}
```

STRINGS

strings are an array of characters

declaration

```
char string_name[size]              //size is max size of elements that can be stored

char string_name[ ] = { 'H', 'e', 'l', 'l', 'o', '\0' };          //mean same thing
char string_name[] = “Hello”;                                      //mean same thing
```

//both are strings with size of 6 elements and mean the same thing as \0 is a null character that would get appended at the end of double quotes string declaration.

scansets = filters input stream and stops once a character doesn’t match the filter

```
scanf(“%[aeiou]”,z);
```
//for “aeiouxxaeiou” inputted, the value-string stored in z would be “aeiou” as the storage will stop once “x” comes

```
scanf(“%[^aeiou]”,z);
```
//for inverted scan set, the string “string1” will be stored as “str” as it will stop once “l” comes

taking string from user

```
scanf(“%s”, string_name);                    //strings don’t need “&” as its only for variables
```

printing string

```
printf(“%s”, string_name);
```

counting number of words in string

```
while(string[i]!='\0')
{
  if (string[i]==' '&& string[i+1]!=' ')
   count++;
   i++;
}
printf(“ %d no. of words = " ,count);
```

how to input a string with many lines

gets(string); //will accept string unless enter key is pressed

strlen() = used to measure the length of string

n = strlen(string);

printf("%d",strlen("string1")); //can be used as argument as well

int n = strlen(string1); //can be used to convert length to variable

WAP to encrypt and decrypt a string (using addition +1 cipher)

```
for (i=0; string[i]!='\0'; i++)
sent[i]=sent[i]+1;

printf(" the encrypted string is \n");
puts(sent);

for(i=0; sent[i]!='\0'; i++)
sent[i]=sent[i]-1;

printf(" the decrypted string is \n");
puts(sent);
```

strcpy() = used to copy the characters from one string to another.

strcpy(string1,"hello"); //assigning hello to string1

strcpy(string1,string2); //assigning string2 to string1

strcmp() = used to compare two strings, will return 0 if both strings are same and if they aren't same then the value returned will be the first difference encountered in ASCII values

strcmp(string1,string2); //value returned is negative if first string is less than second string

strcmp("their","there"); //will return -9 as "r" is 9 ahead of "l" in ASCII

strcat(string1,string2); = appends(adds) string2 to string1 after removing \0 character while string2 remains unchanged

WAP to check if a string is palindrome or not

```
for(i=0; i<n/2; i++)
{
if(str[i]!=str[n-i-1])
    {
    flag=0;                          //if flag==1 it's a palindrome and flag==0 then its not
     break;
    }
}
```

WAP to change all lower case letters to upper case letters

```
for(i=0; string[i]!='\0'; i++)
    n++;
    for(i=0; i<n; i++)
    {
     if(string[i]>=97 && string[i]<=122)
    string[i]=string[i]-32;
    }
```

WAP to check the last occurrence of a character in a string

```
    int lastIndex = -1;  // Initialize the index to -1 (not found)

    // Iterate through the string
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == ch) {
            lastIndex = i;  // Update the index when the character is f
        }
    }
```

//ch is the variable for character that needs to be searched input by user

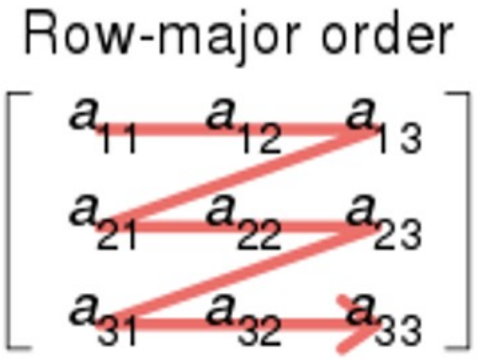
TWO DIMENSIONAL ARRAY

declaration : datatype arrayname[rowsize][collumnsize];

for example
int marks[5][3];
float matrix[3][3];
char page[25][80];

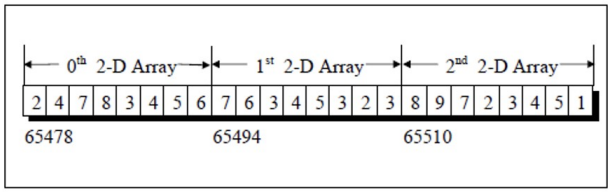
	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

```
int table [2][3] = {0,0,0,1,1,1};          //all three mean same thing
int table [2][3]={{0,0,0},{1,1,1}};        //all three mean same thing
int table [2][3]=   {       {0,0,0},       //all three mean same thing
                            {1,1,1}    };
```



//declaration in array is always done row by row

In memory the array elements are stored linearly.



//arrays are always stored linearly in memory

adding elements to an two dimendional array

```
for(i=0; i<m; i++)                    //m is number of rows in array
{
    for(j=0;vj<n; j++)                //n is number of collumns in array
    {
    scanf("%d", &a[i][j]);
    }
}
```

printing elements of an two dimendional array

```
for(i=0; i<m; i++)
{
    for(j=0; j<n; j++)                //m is number of rows in array
    {
      printf("%d\t",a[i][j]);         //n is number of columns in array
    }
     printf("\n");
}
```

adding elements of two matrixes (two dimensional arrays)

```
for(i=0; i<m; i++)
    for(j=0; j<n; j++)
         c[i][j]=a[i][j]+b[i][j];     //a[][] & b[][] are two matrixes needed to be added
```

adding finding trace (sum of elements of the principal diagonal which goes left to right,top to bottom) and norm (square root of sum of squares of all elements in a matrix)

```
int trace=0,sum=0,I,j,norm;                  //declaring variables for code

for(i=0; i<m; i++)  //m and n(rows and column size) are already declared before the
code
  trace=trace + a[i][i];                     //finding trace

for(i=0; i<m; i++)                           //finding norm
    {
    for(j=0; j<n; j++)
    sum=sum+a[ i ][ j]*a[ i ][ j ];
    }
norm=sqrt(sum);
printf(" trace is %d", trace );
printf(" norm is %d", norm );
```

finding rowsum & collumnsum of the matrix

```c
#include <stdio.h>

int main() {
    int a[10][10], rowsum[10], colsum[10];
    int m, n; // Added variable declarations for m and n
    int i, j; // Added variable declarations for i and j

    printf("Enter dimensions for a (m n): ");
    scanf("%d %d", &m, &n);

    // Reading
    printf("Enter elements for a:\n");
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++)
            scanf("%d", &a[i][j]);
    }

    // Row sum
    for (i = 0; i < m; i++) {
        rowsum[i] = 0;
        for (j = 0; j < n; j++)
            rowsum[i] = rowsum[i] + a[i][j];
    }

    printf("\n");
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++)
            printf("\t %d", a[i][j]);
        printf(" -> %d\n", rowsum[i]); // Corrected the formatting
    }
    printf("\n");

    // Display column sums
    for (i = 0; i < n; i++)
        printf("\t %d", colsum[i]);

    return 0; // Added return statement
}
```

```
2       3       4       3 -> 12
3       3       3       3 -> 12
3       3       3       3 -> 12

0       0       0       0
```

to check if a matrix is symmetrical or not

//if(rows!=columns) then matrix is not a square
//if (a[i][j]!=a[j][i]) the matrix is not symmetric

to swap the prinicipal diagonal(main diagonal; left to right) with secondary diagonal

```
for(i=0; i<n; i++)
for(j=0; j<n; j++)
  if(i==j){
    temp=arr[i][j];                    //temp is declared as int temp before code
    arr[i][j]=arr[i][n-i-1];
    arr[i][n-i-1]=temp;                //printing arr[i][j] now should give swapped matrix
  }
```

to exchange rows

```
for(j=0; j<n; j++) {
  temp=arr[r1-1][j];                   //r1 and r2 are rows to be exchanged inputted by user
  arr[r1-1][j]=arr[r2-1][j];
  arr[r2-1][j]=temp; }
```

to exchange collumns

```
for(i=0; i<m; i++) {
  temp=arr[i][c1-1];                   //c1 and c2 are rows to be exchanged inputted by user
  arr[i][c1-1]=arr[i][c2-1];
  arr[i][c2-1]=temp; }
```

to multiply two matrixes a[i][j] & b[i][j]

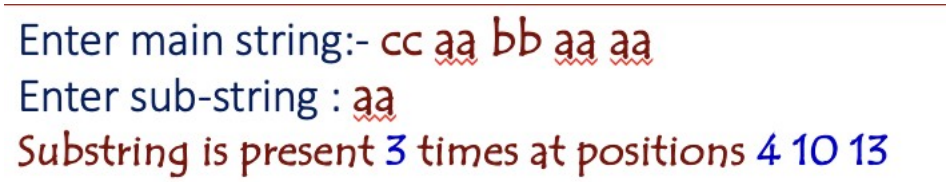
//if collumns size of a[i][j] & row size of b[i][j] aren't equal then its not multiplicable

```
for(i=0; i<m; i++)                     //m&n are rows&collumns of a[i][j]
{
    for(j=0; j<q; j++)                 //p&q are rows&collumns of b[i][j]
    {
    c[i][j]=0;
    for(k=0; k<n; k++)
     c[i][j]=c[i][j]+a[i][k]*b[k][j];
    }
}
```

to sort names in alphabetic order

```
int main()
{
    char string[30][30],temp[30];
    int no, i, j;
    printf("\nEnter the no of strings:“);
    scanf(“%d”,&no);
    printf("\nEnter the strings:“);
    for(i=0; i<no; i++)
    gets(string[i]);

    for(i=0; i<no-1; i++)
            for(j=i+1; j<no; j++)
            {
             if(strcmp(string[i],string[j])>0)
            {
                    strcpy(temp,string[i]);
                    strcpy(string[i],string[j]);
                    strcpy(string[j],temp);
            }
            }
    printf(“the sorted array is”);
    for(i=0; i<no; i++)
    puts(string[i]);
    return 0;
}
```

to find the number of times a substring appears in a string



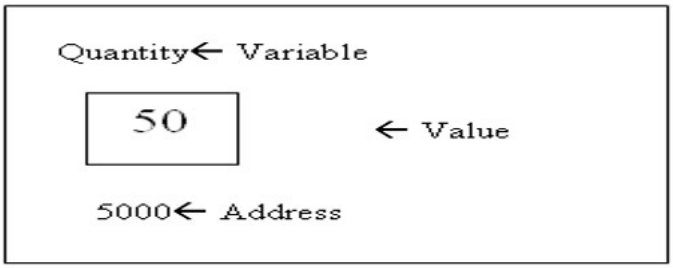
//should be possible by going through every term and adding the position of every matching term to the index value of another array

to check if an array is magic square or not

//make the sum of first row as referential value, then verify if the sum of every row, every column, sum of both the diagonals is same as sum of first row for the array to be a magic square

POINTERS

for the expression -> int Quantity=50
//the variable Quantity will have a value 50 inside the whatever address(lets assume 5000) it is located at



//both 5000 and variable Quantity can be used to access value 50

%p is used to print memory addresses of variables
printf("%p",%randomvariable); //will print memory address of variable

declaration = pointers can't be used without being declared or initialized

datatype*pointername;
pointername =&variable; //variable needs to be defined before assigning pointer to it

OR

datatype*pointername=&variable; //variable needs to be defined before assigning pointer to it

this statement declares a variable that is called a pointer(pointing to a datatype) and can be used to store the address of any variable.

//& is address operator(reference) and is used to assigning the address of a variable whereas * is value at address operator(deference)

//int*p = &x, x; is a syntax error as x needs to be declared before a pointer is declared to it

//one can not assign a numerical absolute value to a pointer so **int*p; p=5000;** is a invalid

pointer comparison
lets say there are two pointers (p1 and p2) pointing to two variable (a and b)
p1=p2; //this means that p1 will now point to "b" and not "a"

so if a=10 and b=10 as well,
*p1=20 //means that b is now 20 as p1 points to b after p1=p2, whereas a is still 1

printing address of a variable

int var1=33;
printf("%x",&var1);

//will give output be 0x29feec or whatever is the address of variable "var1"

accessing variables through pointer

lets assume p is a pointer that stores the address of variable1 or **int*p=&**variable1;
variable2 = *p;
will mean that variable2 will get assigned the value of variable1

*pointername is value at address operator and will always refer to the actual value inside the address

printf("%d",*p); //prints the value inside address of pointer that it is pointing to

int*p
p=&variable1
*p=100 //same as variable1 = 100
variable2 = *p //same as variable2 = variable1

operating with value at address operating pointers

if p1 and p2 are pointers to two variables a and b

c = *p1**p2; or c = (*p1)*(*p2); are both completely valid expressions

operating with pointers on its own

if p1 and p2 are pointers with p1 = 2004

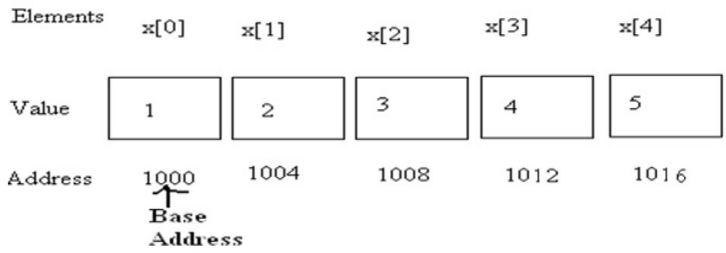
p2=p1+1; //means that p2 will be 2004+4 as one int unit is 4 bytes in c

a=p2-p1; //will give answer as 1 as the difference divided by 4 is one unit

while p1++ is a valid statement, pointers cannot be added to each other legally so p1+p2 is a invalid statement

multiplication and division of pointers is not allowed

pointers and arrays = the first character of array is stored in base address and remaining characters are stored in consequent addresses



declaring pointers to arrays

```
int x[5]={1,2,3,4,5};
int*p
p=x          or          p=&x[0]          //p=&x is invalid
```

printing successful elements of arrays

```
int array[n] = {   ,   ,   ,   ,   };
int*p;
p=array;

for(int i=0; i<n; i++)
printf("%d",*p++);
```

summing elements of an array

```
int *p, sum=0, i=0;
int array[n] = {   ,   ,   ,   ,   };
int*p;
p=array;

while(i<5)
    {
      sum+=*p;
     i++;
     p++;
     }

while(*pointer!="\0")
   pointer++
```

finding length of string using pointers

```c
#include <stdio.h>

int main() {
    char name[15];
    char *cptr = name;

    printf("Enter some word to find its length: \n");
    scanf("%s", name);

    // Move the pointer to the end of the string
    while (*cptr != '\0') {
        cptr++;
    }

    // Calculate and print the length
    printf("Length: %ld\n", cptr - name);

    return 0;
}
```

assigning pointer to a constant string

```
char*pointer1;
pointer1=”stringexample”;          //here pointer1 is a pointer to stringexample
```

printing a 2d array using pointers

```
int a[][no_of_collumns] = {{x,y},{a,b}};

int(*p)[no_of_collumns];

p=a

for(i=0;i< no_of_collumns;i++)
{
    for(j=0;j<no_of_collumns;j++)
    printf(“%d\t”,*(*(p+i)+j));
    printf(“\n”);
}
```

two dimensional array with strings & pointers

char*pointer[3][25]; //here 3 Is number of strings in array and 25 is max characters in each string

char*pointer[3] = {“string1”,”string2”,”string3”}; //the declaration allocates 24 bytes including null character

for(i=0; i<=2;i++) //to print all the strings in array
printf(“%s”,pointer[i]); or printf(“%s”, *(pointer + i));

*(pointer[i] +j) //used to access j^{th} character in i^{th} name

void pointers

void pointers like void*pointer1; are used when there is not a specific datatype that pointer needs to point to

for example void*pointer1 can be used to point pointer1 to variables of any datatype including int, float or even characters

Lab 12. Pointers

1. Write a program to access two integers using pointers and add them.
2. Write a program to find out the greatest and the smallest among the three numbers using pointers.
3. Write a program to determine the length of a character string using a pointer.
4. Write a program to compute the sum of all elements stored in an array using a pointer.
5. Write a program to determine whether a substring (string 1) is in the main string or not. If present, return the pointer of the first occurrence.

(practice to see if you can solve these questions)

FUNCTIONS

functions are used to do similar tasks without writing the whole code again and again

datatype_to_be_returned functionname(int x)//int x is called formal parameters
{
body of function
}

for example : void displaynumber(int n) and int swap(int a)
int change(int*p) //we can even use pointer as function arguments

functioname(variable1); //calling function for variable1 where variable1 is called actual parameter

function parameters variable names aren't needed but datatype must be mentions for the order of function calling

void display(int, char) //is valid statement and function call is display(integer1, character1);

difference between global and local variables

global variables are outside function and hold their value throughout the program
ex int a; int main() {……}

local variables are inside function and hold their value only inside the function and can't be used outside the function
ex int main() {int a; ……}

pass by value

pass by value is when the formal parameters are variables of datatypes like int

example void swap(int x, int y) { body of function }

passing one dimensional array to a function

int function1(int a[], int a) //declaring function with array
……
function1(b, n); // if b[] is a array, then the function is called in this format

passing two dimensional array to a function

int function2(int x[][column_size], int m, int n) //second dimension must be specified
……
function2(a, m, n); //if a[][] is a array, then the function is called in this format

pass by reference (pointers)

pass by reference is when the formal parameters are pointers

```
void swap(int *x, int *y)
{
    int t=*x;                  //function code to swap the integer at addresses x and y
    *x=*y;
    *y=t;
}

int main()
{
 int a=5,b=7;
 swap(&a, &b);              //this is how u do a function call using pointers
 printf("After swap: a=%d and b= %d",a,b);
 return 0; }
```

return statement

you can return 0 in if you already print the answer you need with the solution of code or you can return a the solution of function which will return it to a variable like
z = fn(x); //if there was a return variable statement where variable is answer.

anything after return statement won't run in code so return 0; printf("hello"); in function wont printf hello in function call

prototype

a function written after call statement wont work but we can write a prototype code before the call statement to run the program without error

for example

```
printf("%d",squareofnumber(10));
int squareofnumber(int number) { body of function }
```

wont work but

```
int squareofnumber(int number);          //prototype code

printf("%d",squareofnumber(10));
int squareofnumber(int number) { body of function }
```

will work

Nested Functions

you can nest functions inside functions

```
int function1(int x)
    {

    int function2(int y)
            {
            //body of inside function
            }

    function2(x)            //running function2 in the body of function1
    return 0;
    }
```

WAP to use pass by reference functions to compute simple and compound interest

```c
// Function to calculate simple interest
void calculateSimpleInterest(float p, float r, float t, float *simpleInterest) {
    *simpleInterest = (p * r * t) / 100.0;
}

// Function to calculate compound interest
void calculateCompoundInterest(float p, float r, float t, float *compoundInterest) {
    *compoundInterest = p * pow((1 + r / 100.0), t) - p;
}

int main() {
    float principal, rate, time, simpleInterest, compoundInterest;

    // Input from user
    printf("Enter principal amount: ");
    scanf("%f", &principal);

    printf("Enter rate of interest: ");
    scanf("%f", &rate);

    printf("Enter time in years: ");
    scanf("%f", &time);

    // Calculate simple interest
    calculateSimpleInterest(principal, rate, time, &simpleInterest);

    // Calculate compound interest
    calculateCompoundInterest(principal, rate, time, &compoundInterest);

    // Display results
    printf("\nSimple Interest: %.2f\n", simpleInterest);
    printf("Compound Interest: %.2f\n", compoundInterest);

    return 0;
}
```

FUNCTION WITH NO ARGUMENT/PARAMETER AND NO RETURN VALUE EXAMPLE

c Copy code

```c
#include <stdio.h>

void functionName() {
    // Function body
    printf("This function has no arguments and no return value.\n");
}

int main() {
    // Call the function
    functionName();

    return 0;
}
```

FUNCTION WITH NO ARGUMENT/PARAMETERS BUT ONE RETURN VALUE EXAMPLE

c Copy code

```c
#include <stdio.h>

// Function with no arguments and a return value
int generateRandomNumber() {
    // Some logic to generate a random number
    return rand(); // Note: You may need to include <stdlib.h> for ran
}

int main() {
    // Call the function with no arguments and get the return value
    int randomNumber = generateRandomNumber();

    // Display the result
    printf("Random number: %d\n", randomNumber);

    return 0;
}
```

FUNCTIONS WITH ARGUMENTS/PARAMETERS BUT NO RETURN VALUE

c Copy code

```c
#include <stdio.h>

// Function with parameters and no return value
void displaySum(int a, int b) {
    int sum = a + b;
    printf("Sum: %d\n", sum);
}

int main() {
    // Call the function with arguments
    displaySum(5, 7);

    return 0;
}
```

FUNCTION WITH ARGUMENT/PARAMETERS AND ONE RETURN VALUE

c Copy code

```c
#include <stdio.h>

// Function with parameters and a return value
int add(int a, int b) {
    int sum = a + b;
    return sum;
}

int main() {
    // Call the function with arguments
    int result = add(5, 7);

    // Display the result
    printf("The sum is: %d\n", result);

    return 0;
}
```

STRUCTURE

The general format of a structure **definition**

```
struct structure_name
{
  data_type member1;
  data_type member2;
  ...
};
```

advantages of structures over arrays

array is homogenous since it can contain only one datatype whereas struct is heterogenous and can be composed of different datatypes

structures allow user defined datatypes for more abstractions in program which is not allowed in arrays

structures provide flexibility in terms of grouping different types of variables together

assigning and printing values in structure

for example we have a structure

```
struct student{

    int age;
    double gpa;
    char name[50];
    char major[50];

};                              //semi colon after curly braces

int main(){
    struct student student 1;

    student1.gpa = 5;
    student1.age = 19;
    strcpy(student1.name,"varun");      //only way to assign strings to struct datatypes
    strcpy(student1.major,"stem");      //student1.name = "varun"; doesn't work


    printf("%f",student1.gpa);          //code to print the structure stored values
    printf("%d",student1.age);
    printf("%s",student1.name);
    printf("%s",stuident1.major);

    return 0;
}
```