**C Language**

Text editor - VS CODE

COMPLILER – MinGW-64 for windows > install compiler for all lang > copy path from this PC > paste in path of advanced system settings in control panel.

EXTENSION – download c/c++ extension pack (Microsoft).

**Compiler**

A computer program that translate C program into machine code.

**Variable**

Name of a **memory location** which stores some data.

**Rules**

Variables are case sensitive.

First character should be alphabet.

No comma or blank space.

No symbol except “\_”.

**Datatypes**

There are different type of data types (type of data float, string, etc)

int age = 22;

here **int** is the **type of data , age** is **memory location** and **22** is the **value.**

float pi = 3.14;

here **float** is the **type of data , pi** is **memory location** and **3.14** is the **value.**

char hashtag = #;

here **char** is the **type of data , hashtag** is **memory location** and **#** is the **value.**

**Constants**

Values that do not change.

Integer constants

1, 2, 3, 0, -1

Real constants

1. ,2.0 ,3.14, -2.14

Character constants

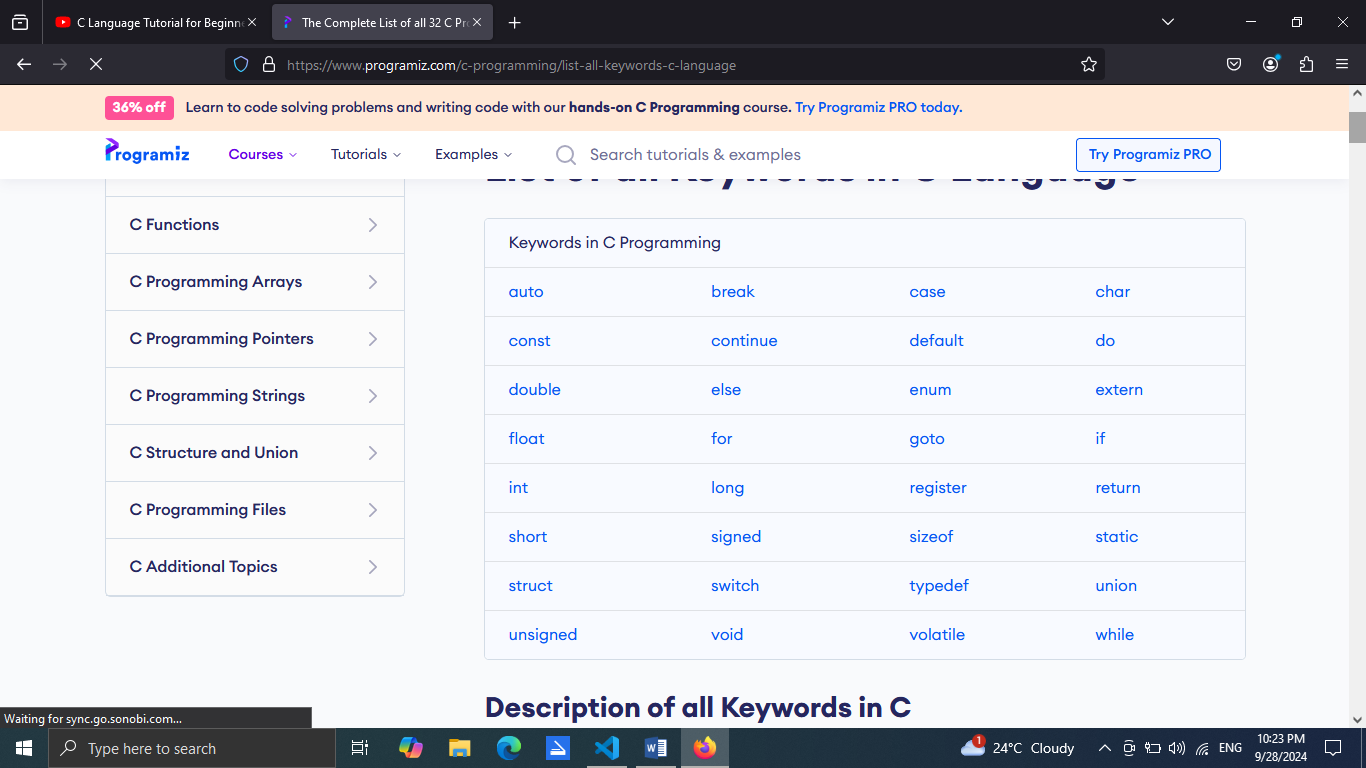
‘a’ ,’b’, ‘A’ ,’# ‘, ‘&’

**Keywords**

**Return type** of a function

**Reserved words** that has special meaning.

There are **32** keywords in C language



Variables should not be named in the name of keywords coz they are reserved.

**Functions**

**Library function –** inbuilt in C

**User defined function –** definedby user

Function that take the value is called **Parameter**

Function that return value is called **Return Value.**

**Variable** defined in a function is known to that function only.(you cannot assume that other function know that variable without declaring.)

**Program structure**

# include<stdio.h>

int main(){

    printf("hello world ");

    return 0;

}

<stdio.h> is **preprocessor directive.**

We write **int** before the **main function** because it has to **return 0.**

The **execution** of program starts from **Main function.**

A function can be called **directly or indirectly** from the main function.

**;** is like full stop to a line and is **mandatory**

**return 0** means there is no error.

**Comments**

Extra instruction or phrases which are not the part of code.

//this is a single line comment

/\*this is a multiline comment\*/

So the part which doesn’t make sense to compiler are comments.

**To print a phrase and leave a line we use**

    printf("hello world \n");

here \n is used to go to next line.

**The stuff inside a string (” ”) will print as it is.**

* **Integer**

Here we decaled a variable (age)

int age = 22

but we cannot ask output like this

 printf ("age is age");

instead we will be specifying a **datatype**.

 int age = 22;

    printf("age is %d",age);

* **Long integer**

**%ld** is the format specifier for long integer.

* **Real number**

Similarly **%f** for real numbers.

  float pi = 3.14;

    printf("value is %f", pi);

* **Characters**

And **%c** for characters.

  char star = '\*';

    printf("star is %c",star);

These **%d %f %c ld%** is known as **Format specifiers**.

* **Input and output**

**Scanf** is a library in **C** which read the formatted input.

for input

scanf("%d" , &anjani);

for output

printf("%d" , anjani);

Here is an example

int main(){

    int age;

    printf("enter age:");

    scanf("%d", &age);

    printf("age is :%d", age);

    return 0;

}

& is used to store a value to the memory location of which it has declared.

Here is another example to print 2 number

int main() {

    int a, b;

    printf("enter first number");

    scanf("%d",&a);

    printf("a is :%d\n", a);

    printf("enter second number");

    scanf("%d",&b);

    printf("b is :%d", b);

    return 0;

}

And one more example for sum of two numbers.

int main() {

    int a, b;

    printf("enter first number");

    scanf("%d",&a);

    printf("enter second number");

    scanf("%d",&b);

    int sum = a + b;

    printf ("sum is %d", sum);

    return 0;

}

Alternatively we can bring an output without even writing 3rd variable i.e. int sum = a + b;

By

int main() {

    int a, b;

    printf("enter first number");

    scanf("%d",&a);

    printf("enter second number");

    scanf("%d",&b);

    printf ("sum is %d**", a + b**);

    return 0;

}

Similarly we can use -,\*,/,% operations.

To find area of square following code can be used.

int main() {

  int side;

  printf ("enter side");

  scanf("%d",&side);

  printf("area is: %d" , side\*side);

  return 0;

}

To find area of circle following code can be used.

int main() {

  float R;

  float pi=3.14;

  printf ("enter R:");

  scanf("%f",&R);

  printf("area is: %f" , pi\*R\*R );

  return 0;

}

To find circumference of circle

    float r ;

    float pi = 3.14;

    printf("enter radius:");

    scanf("%f" , &r);

    printf( "Circumference:%f", 2\*pi\*r);

**Instructions**

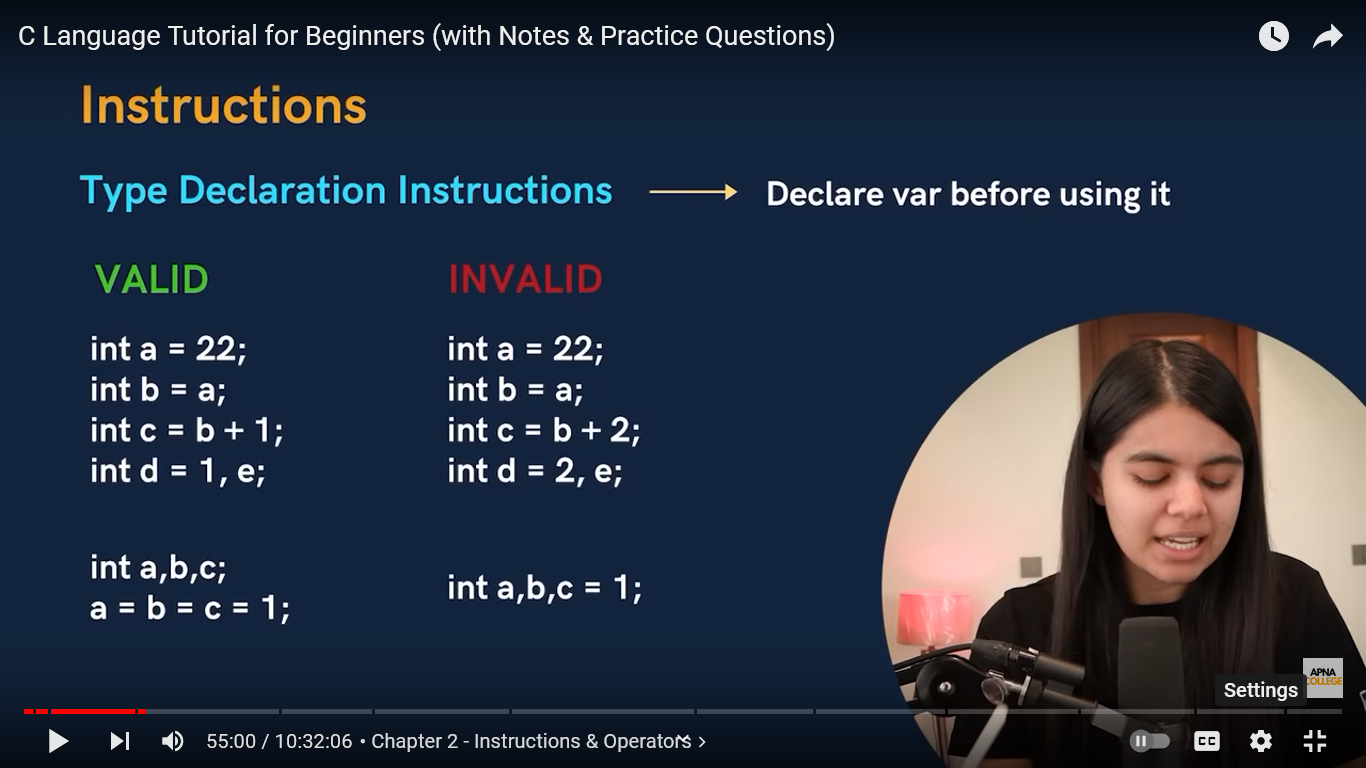
**Instructions** are statements in a program.

* **Type declaration**

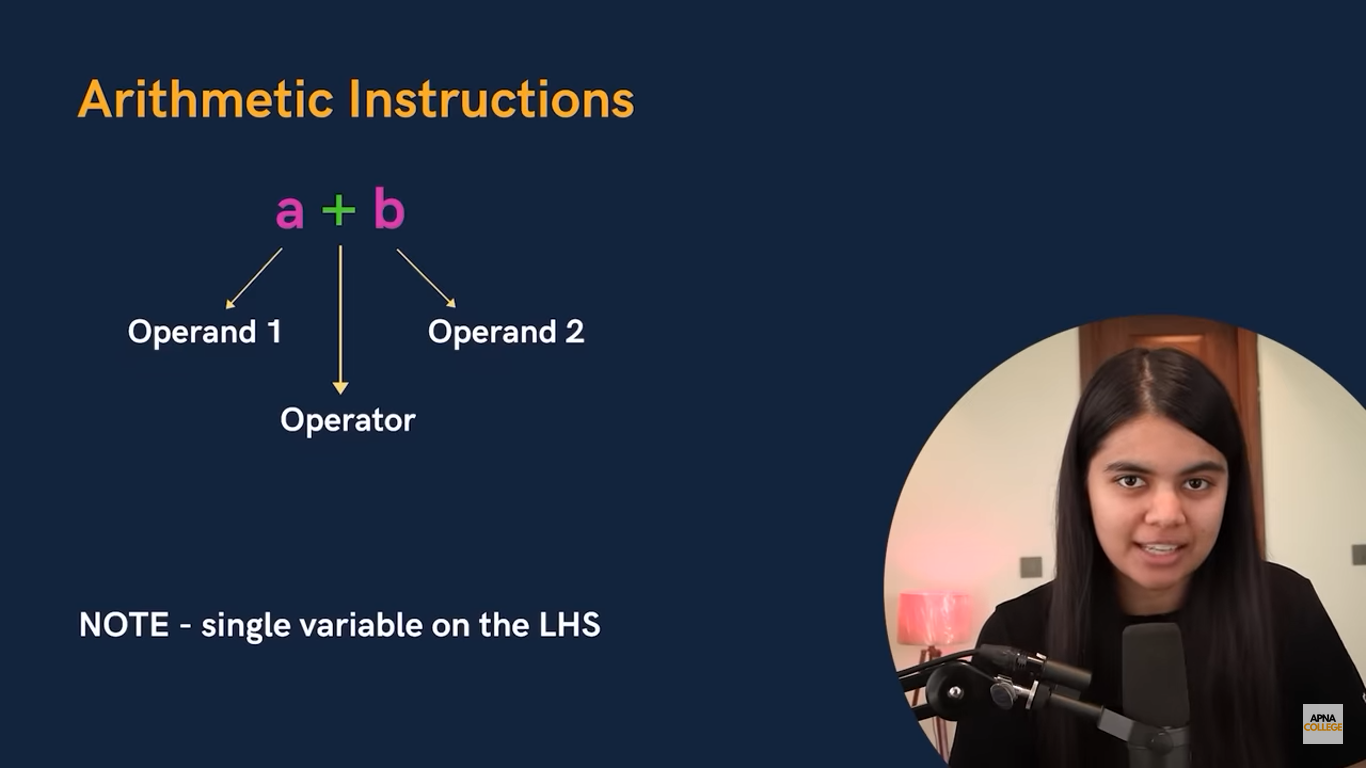
To **declare** the variable before using it.

Compiler of C goes line by line so **declare the variable first then use** it.

We **cannot use and declare** at same time.



* **Arithmetic instruction**



Nowalways there should be **single variable on LHS.**

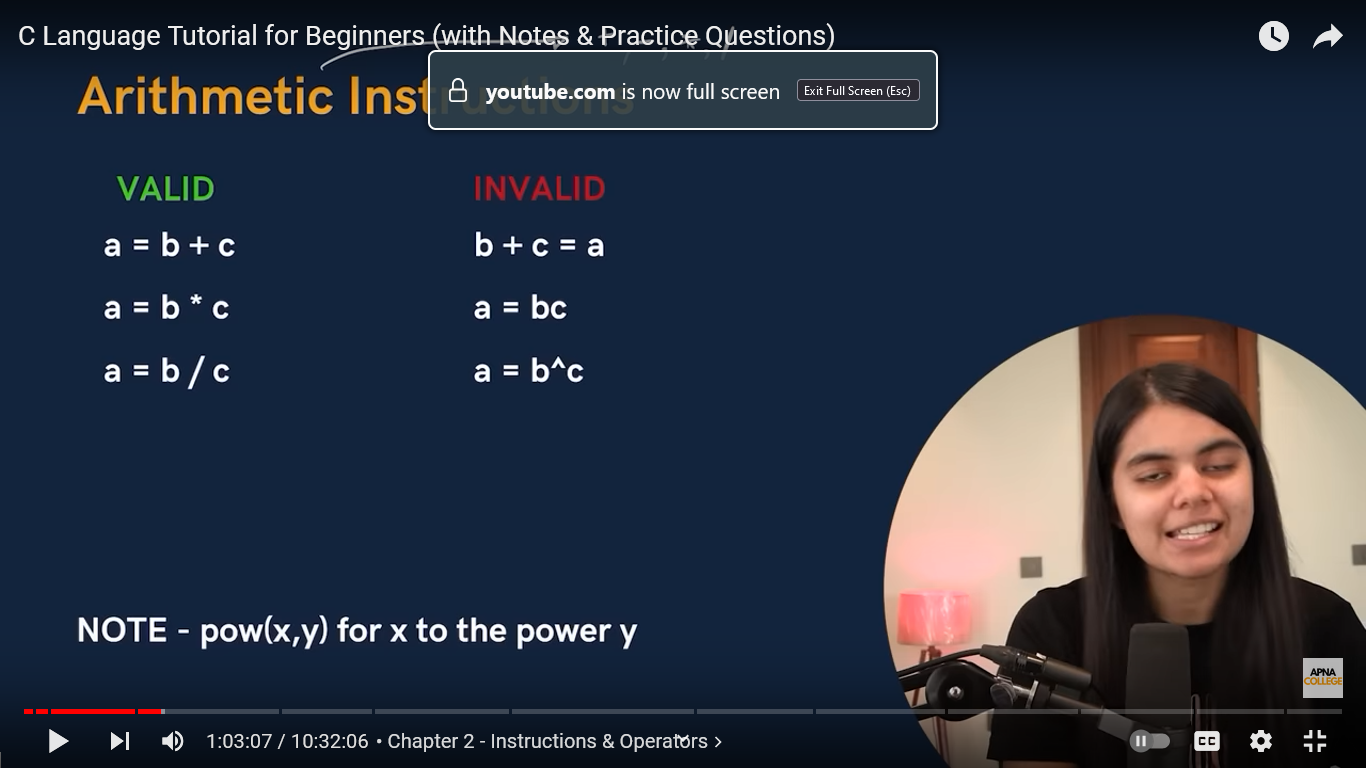
int main() {

  int a=1 , b=2;

  int sum = a+b;

   printf("The sum is: %d", sum);

  return 0;



For **power** we use **pow(x,y)** function.

int main() {

  int a=6,b=2;

  int power = pow(a,b);

   printf("The ans is: %d", power);

  return 0;

}

To find the square of a number.

 int a ,b=2 ,c;

 printf("Enter no.:");

 scanf( "%d" , &a);

 c= pow(a,b);

 printf("Square of the no. is:%d", c );

return 0;

avg of 3 number

  int a , b ,c;

  printf("Enter first number:");

  scanf("%d", &a);

    printf("Enter second number:");

  scanf("%d", &b);

    printf("Enter third number:");

  scanf("%d", &c);

  printf("Average is:%d" , (a+b+c)/3);

**Prime number**

int a , i, isPrime = 1;

    printf("Enter a number: ");

    scanf("%d", &a);

    if (a <= 1) {

        isPrime = 0;

    }

    else {

        for (i = 2; i \* i <= a; i++) {

            if (a % i == 0) {

                isPrime = 0;

                break;

            }

        }

    }

    if (isPrime) {

        printf("%d is a prime number.\n", a);

    }

    else {

        printf("%d is not a prime number.\n", a);

    }

    return 0;

}

There is a **Modular operation**. It is a **type of operation** which means **remainder** of the two and is denoted by **%**.

**Ex 5%3=2**

**(**sign of numerator decides the sign of remainder.**)**

**Type conversion**

**Explicit –** user define the type.

int main() {

  int a = (int)1.999999;

printf("%d", a);

  return 0;

}

**Implicit –** compiler decide the type**.**

int main() {

int a = 1.999;

printf("%f",a);

  return 0;

}

**If output is big then it will be needing more container so it will decide itself whether to keep in float or int.**

int main() {

printf("%d", 2/3);

  return 0;

}

**Here if we want exact value then we will have to take numerator float so that well get the exact value.**

int main() {

printf("%f", 2.0/3);

  return 0;

}

**Operator precedence** (priority of arithmetic operators)

\*,/,%

+,-

= (assignment operator)

**When the sum comes under same precedence (**x=6\*3/5\*6**) then it is calculated left to right by associative law.**

int main() {

  int a = 4\*8/6\*9;

printf("%d", a);

  return 0;

}

**Control instructions**

It is used to  **control the flow of instructions**  like it is not necessary that every time 1st line will be compiled first we can change the flow of program with the help of these control instructions.

**Sequence control**

Default, sequence wise compiled

**Decision control**

Only if statements – if a condition is true then the statements will be executed but if it is false it will not execute.

**Conditional loops**

If else problems

**if-else**

  int age;

  printf("Enter age:");

  scanf("%d", &age);

  if (age>17){

     printf("you are an adult");

  }

  else{

    printf("you are minor");

  }

**Loop control**

To repeat some part of the program.

They are of 3 types **for loop , while loop and do while loop.**

     break;

**used to end a loop.**

**For loop**

for (initialisation; condition; upadation)

{

    /\* code \*/

}

**Here I is a iteration or counter variable which means we are repeating a work with the help of this variable.**

**Best when number of iteration is known.**

**Note: if you left the blank space in any of the condition then it will create a problem.eg. infinite loop.**

**Loop can be float or can character.**

To repeat string

for (int i = 0; i <=10; i=i+1)

    {

       printf("anjani \n");

}

Backward counting

for(int i=100; i>1; i=i-1){

    printf("%d \n" , i);

    }

0 to 10

 for(int i=0; i<11; i=i+1){

    printf("%d \n" , i);

    }

To print in reverse

for (int i = 10; i >= 1; i--)

**Let us do a question**

To print the sum of first n natural number(n given by user) and print the numbers in reverse.

int n , i;

int sum = 0;

printf("enter any positiive integer:");

scanf("%d", &n);

for(int i=1; i<=n  ; i++)

{

    sum+=i;

}

printf("sum is: %d\n" , sum);

for ( i = n; i >=1; i--) //here i is declared again and will not create any error as the life span of a variable is defined till the loop is over.{

    printf("%d\n" , i);

}

Or we can merge both the for loop condition by declaring new variable

int n , i , j;

int sum = 0;

printf("enter any positiive integer:");

scanf("%d", &n);

for(int i=1 , j=n; i<=n && j>=1 ; i++ , j--)

{

    sum+=i;

    printf("%d\n" , j);

}

printf("sum is: %d\n" , sum);

code to print table of number n (given by user).

**Code to print of a number if its even and square of number if its odd.**

   int a;

    int i;

    long factorial = 1;

    printf("enter any number:");

    scanf("%d", &a);

    if (a % 2 == 0) {

        for (i = 1; i <= a; i++) {

            factorial \*= i;

        }

        printf("%ld", factorial);

    } else {

        printf("%d", a \* a);

    }

**Pyramid question**

int rows, i, j;

    printf("Enter the number of rows: ");

    scanf("%d", &rows);

    for (i = 1; i <= rows; i++) {

        for (j = 1; j <= rows - i; j++) {

            printf(" ");

        }

        for (j = 1; j <= (2 \* i - 1); j++) {

            printf("\*");

        }

        printf("\n");

    }

Badmosi code

 for(int i=1; i<1; i = i-1){

    printf("%d \n" , i);

    }

int rows = 8, i, j, space;

for (i = rows; i >= 1; ++i)

{

    for (space = 0; space < rows - i; ++space)

        printf("");

    for (j = i; j <= 2 \* i - 1; ++j)

        printf("\*");

    for (j = 0; j <  i - 1; ++j)

        printf("\* ");

        for (j = 0; j <  i - 1; ++j)

        printf("\*  ");

    printf("\n");

}

**While loop**

**(jb tk)**

Declaration;

While(condition/termination){

Work;

Updation;

}

To print a string as many times as you want but it is fixed.(already declared the number of repeatition).

int i=1;

while(i<=5){

    printf("hi \n");

    i++;

To repeat a string for the number of times user input.

    int n;

    int i=1;

    printf("enter any number:");

    scanf("%d" , &n);

while(i<=n){

  printf("anjani \n");

  i++;

}

**Do-while loop**

**First do work then put condition**

int i=1;

do{

    printf("%d \n" , i);

    i++;

}

while (i<=5);

do is used to write the work we have to get and then it check the condition being in while.

Simple calculator using do while

 int choice;

    double num1, num2;

    do {

        printf("\nSimple Calculator\n");

        printf("1. Add\n");

        printf("2. Subtract\n");

        printf("3. Multiply\n");

        printf("4. Divide\n");

        printf("5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        printf("Enter first number: ");

        scanf("%lf", &num1);

        printf("Enter second number: ");

        scanf("%lf", &num2);

        if (choice == 5) {

            break;

        }

        switch (choice) {

            case 1:

                printf("Result: %.2lf\n", num1 + num2);

                break;

            case 2:

                printf("Result: %.2lf\n", num1 - num2);

                break;

            case 3:

                printf("Result: %.2lf\n", num1 \* num2);

                break;

            case 4:

                if (num2 != 0)

                    printf("Result: %.2lf\n", num1 / num2);

                else

                    printf("Error! Division by zero.\n");

                break;

            default:

                printf("Invalid choice.\n");

        }

    } while (choice != 5);

    printf("Exiting calculator...\n");

**Case control**

**Operators**

**Arithmetic operators**

The five operators we have already read [+ - \* / %]

**Relational operators**

This operator is used to justify the relation between 2 variable.

==[equal to] check if **x=y , (x==y)**

>, >=

<, <=

!=[not equal to]

True =1

False = 0

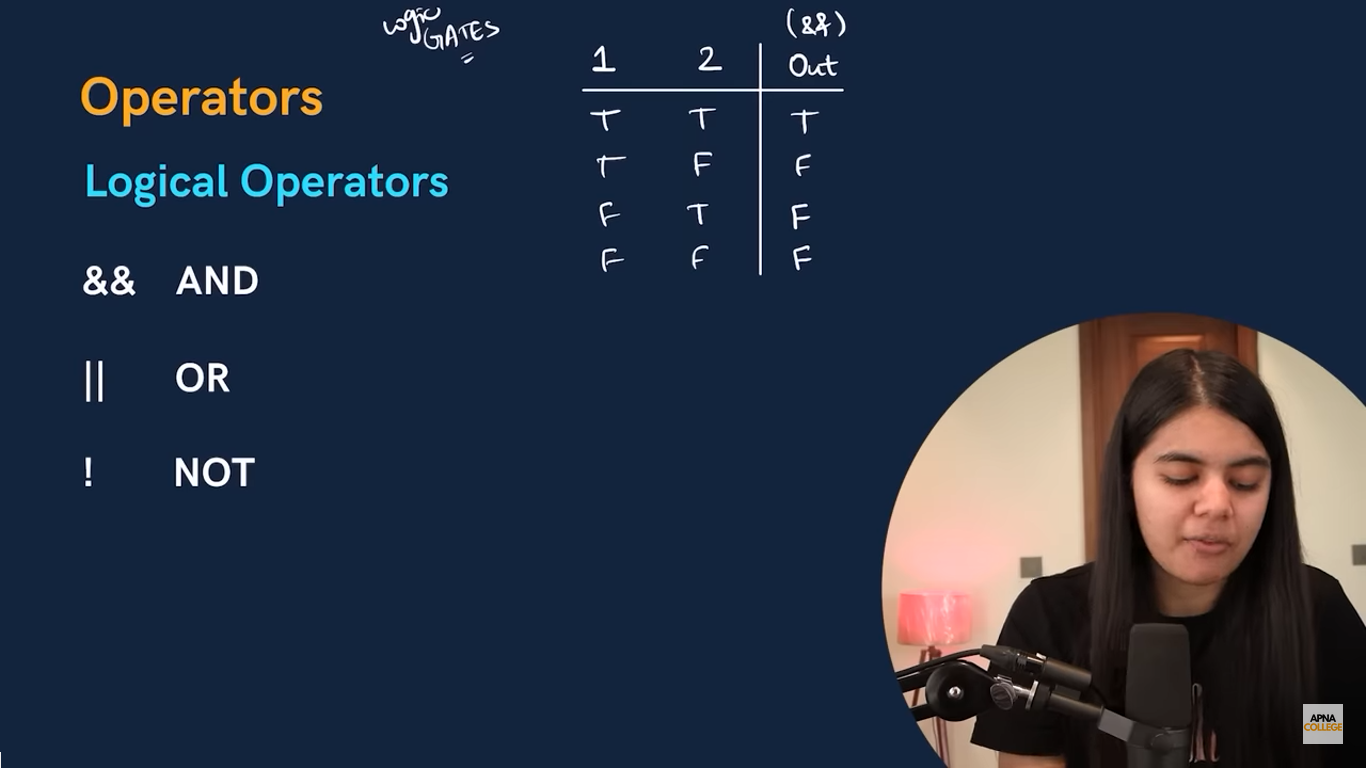
**Logical operators**

This operator is used to justify real life cases.

**&&[and]** – if two or more condition has to be checked whether its true or false then && is used. If one of the statement Is false then output will be false .(all the statement should be true.)

printf("%d",2>3 && 4>2);

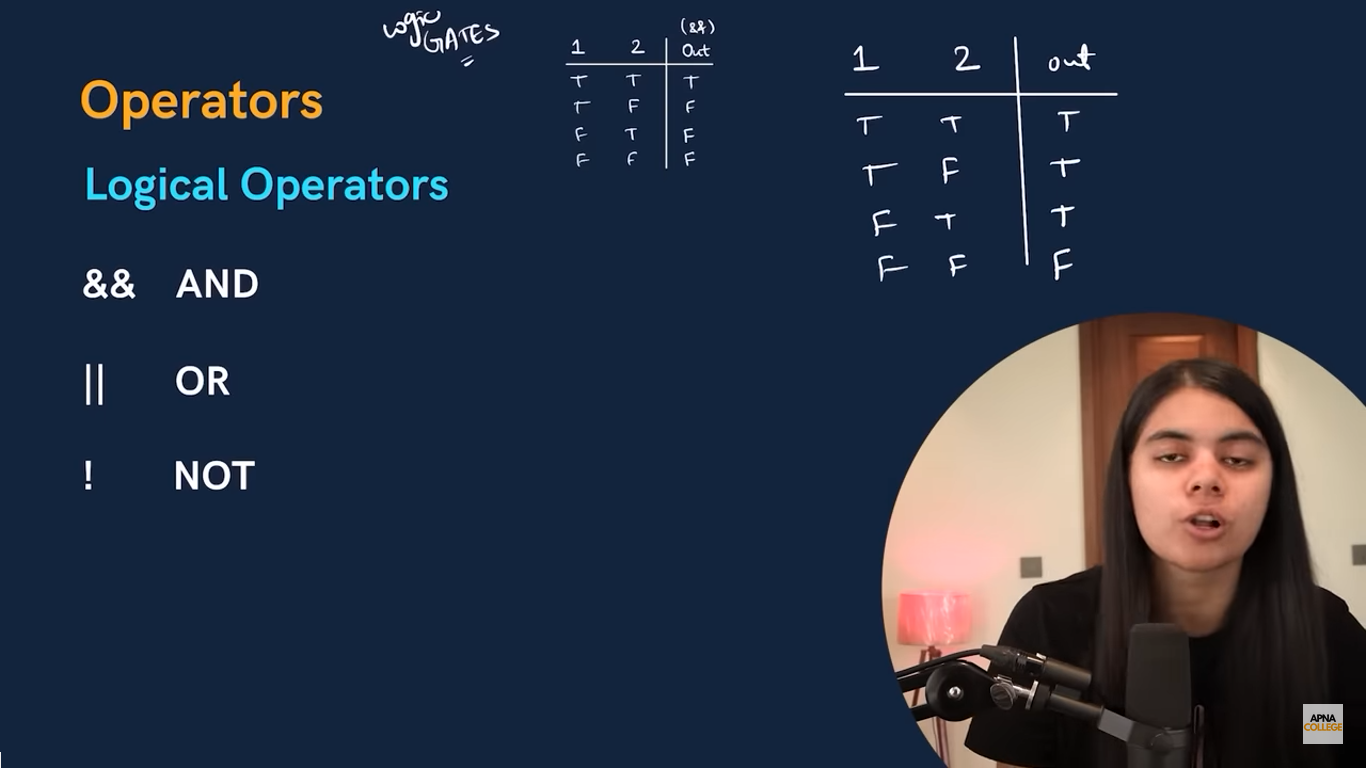
0



**||[or]** – at least one statement should be true then it will give true output.

printf("%d\n",2>3 || 4>2);

1



**! [not]** – not operation is used to convert true to false and false to true.

printf("%d",!(3>8));

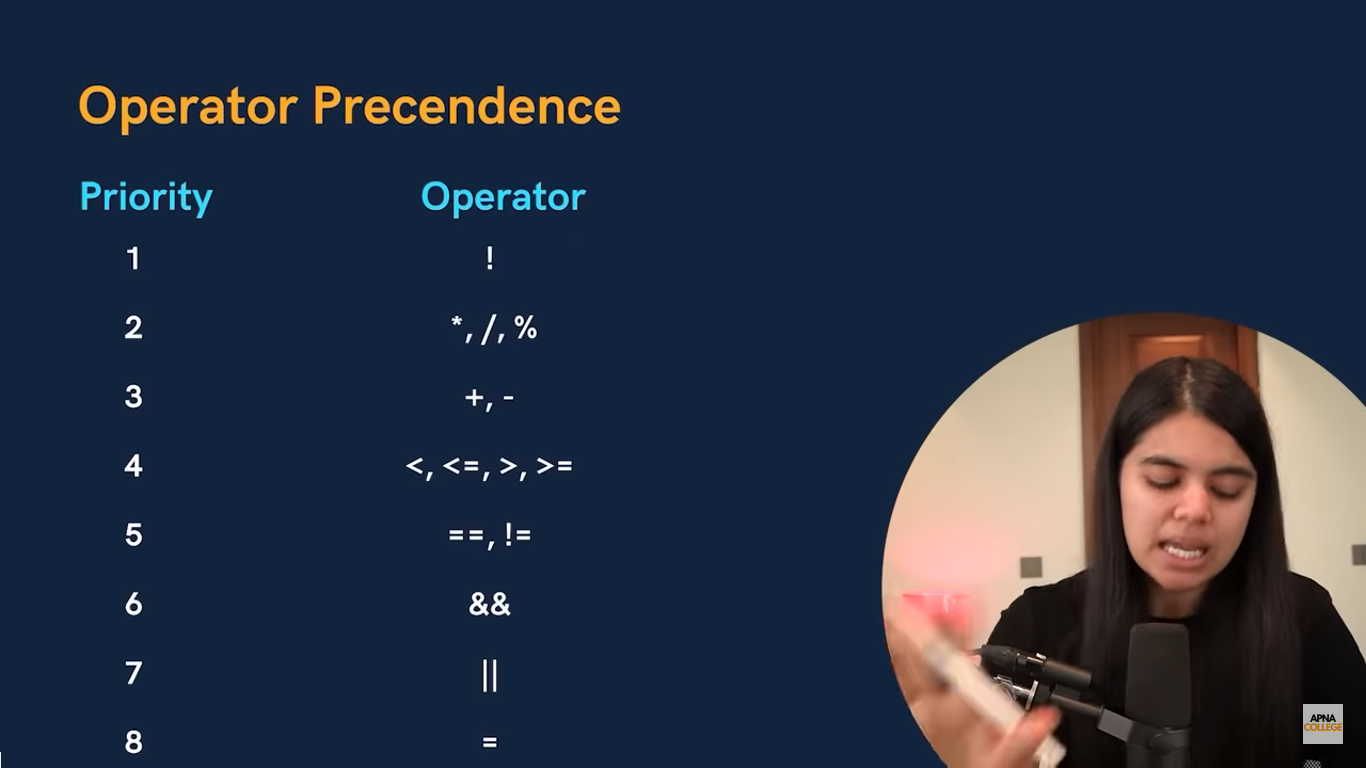
1

Lets see what happens when we apply more than one operator

printf("%d\n",!(2>3 && 4>2));

1

**It always Compare bits of the given object and always return a Boolean result**

**We can use arithmetic condition also with these logical operator**

**Assignment operators**

**=** used to assign a value to variable

**+=** used when both lhs and rhs have same variable(it will add 2 variable)

 int a=1;

  int b=4;

a=a+b;

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

in place of a=a+b

we can write

  int a=1;

  int b=4;

a+=b;

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

**\*=** used when both lhs and rhs have same variable(it will multiply 2 variable)

**-=** used when both lhs and rhs have same variable(it will subtract 2 variable)

**Lets do a question**

Write a program to find whether a number is divisible by 2 or not.

 int x;

  printf("enter a number");

  scanf("%d",&x);

  printf( "%d" , x%2 == 0);

**one more question**

make a true false for the following statement.

1.if it is Sunday and snowy – T

  int issunday =1;

  int issnowy =1;

  printf("%d" , issunday && issnowy);

2. if its Monday or rainy – T

int ismonday =1;

  int israiny=1;

  printf("%d" , ismonday||israiny);

3. if a number greater than 9 and less than 100 – T

  int x;

  printf("enternumber");

  scanf("%d" , &x);

  printf("%d" , x>9 && x<100);

**Increment and decrement operator**

**Post increment i++**

  int i=1;

   printf("%d \n" , i++);

   printf("%d" , i );

here i++ means to first use and then increase the value of i by one.

In the first printf statement – 1 will be printed but the value of i (1) is increased by 1.

So in the next printf statement 2 will be printed because in the compiler value of i is increased.

**Pre increment ++i**

int i=1;

   printf("%d \n" , ++i);

   printf("%d" ,i);

here ++i means increase the value of I by one then use it.

Now in the first printf statement the output will be 2

And same for the next printf statement.

**Pre decrement --i**

 int i=1;

 printf("%d \n" , --i);

 printf("%d" , i);

here for first and second prinf statement output is 0

**Post decrement i--**

 int i=1;

 printf("%d \n" , i--);

 printf("%d" , i);

here for first printf output is 1 and for second its 0.

**Bitwise operators**

**It Perform operations on individual bits, and the result is also always a bit**

**Ternary operators**

**( : )** It is a shorthand for if-else statement.

**Special operators**

**Comma Operator ( , ), Sizeof Operator (sizeof), Pointer Operators (& and \*), and Member Selection Operators (.** **and ->).**

age >= 18? printf("adult") : printf("minor");

**Switch**

int day;

printf("Enter day:");

scanf("%d", &day);

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("satday");

  break;

  case 7 :printf("sunday");

  break;

  default : printf("not a valid day!");

}

**Cases in switch**

Cases can be in any order.

Nested switch (switch inside switch) is allowed.

**Some cases**

int x=2;

if (x = 1) {

  printf("x value is one");

}

  else{

    printf("x value is not 1");

  }

Here output will not be error but the “x value is one“

Coz we have assigned x a value or we given a non zero number (which is considered as true.).

**Every character has an ASCII value(integer) of a character .**

char x;

printf("enter:");

scanf("%c" ,  &x);

if (x>= 'A' && x<= 'Z'){

  printf("valid");

}

// here x>= 'a' & x>= 97  both are same to recognise to the compiler

if(x>= 'a' && x<='z'){

  printf("invalid");

}

**Ascii value of a is 97 for example.**

**Natural number test.**

int x;

    printf("Enter a number: ");

    scanf("%d", &x);

    if (x > 0) {

        printf("Number is natural\n"); // Added missing semicolon

        return 0;

    } else {

        printf("Number is not natural\n");

    }

**Break and continue statements**

    break;

it stops whole loop and stop executing any statements above it.

     continue;

it continues the **loop leaving the satisfying if condition**

to print prime number 1 to 50

 for (int i = 5; i <= 50; i++)

 {

    if (i%2==0)

    {

     continue;

    }

    else

    {

      printf("%d\n" , i);

    }

 }

**Sum of numbers between 5to 50**

 int i;

  int sum = 0;

 for (int i = 6; i <50; i++)

 {

   sum += i;

 }

 printf("%d\n" , sum);

**Function and Recursion**

**Function**

It is a block of code that perform a specific task.

Take argument -> do work -> return result

It can be use multiple times and increases code reusability.

void printhello(){

(work);

}

here **printfhello** is a **function** which is **user defined**.(**function definition**).

**Void** is to tell compiler that a **function exist**.

It will **not return anything**.

//declaration or prototype

   void printhello();

   int main() {

     printhello();//function call

      return 0;

   }

   //function definition

   void printhello(){

      printf("helloworld");

   }

**You can print your function several times by calling printhello(); in main function .**

**A simple program to greet in native language using void .**

printf("nationality - indian(1)/french(2):");

     int a;

   scanf("%d", &a);

   if (a==1)

   {

     indian();

   }

   if (a==2)

   {

      french();

   }

      return 0;

   }

   //function definition

   void indian(){

      printf("namaste");

   }

   void french(){

      printf("bonjour");

   }

**Argument and Parameter**

**Arguments**  are the values that are passed in function call.

Used to send the value.(they are actual parameter)

**Parameter** are the values in the function declaration and definition.

Used to receive the value. (they are formal parameter)

void printable (int n);

int main() {

   int n;

   int i=1;

   printf("enter a number:");

   scanf("%d" , &n);

   printable(n);// here n is argument

   return 0;

}

void printable(int n){// here n is formal parameter

      for (int i = 1; i <= 10; i++)

{

     printf("%d\n" ,i\*n );

   }

}

**Function can only return one value at a time.**

**Changes to parameter in function don’t change the value n the calling function.(**because copy of the argument is passed to the function.**)**

**Lets see an example**

#include <math.h>

float squarearea(float side);

float rectanglearea(float a , float b);

float circlearea(float radius);

int main(){

   float radius = 3.0;

   printf("area is:%f" , circlearea(radius));

   return 0;

}

float squarearea(float side){

return side \* side;

}

float rectanglearea(float a, float b){

return a \* b;

}

float circlearea(float radius){

   float pi = 3.14;

return pi \* radius \* radius;

}

**Here we can test for rectangle area square area also by calling it in main function.**

**Recursion**

When **a function calls itself**.

Here is simple code to print a string for 5times.

void printfB(int count);

int main(){

   printfB(5);

   return 0;

}

void printfB( int count){

   if (count == 0) {

      return;

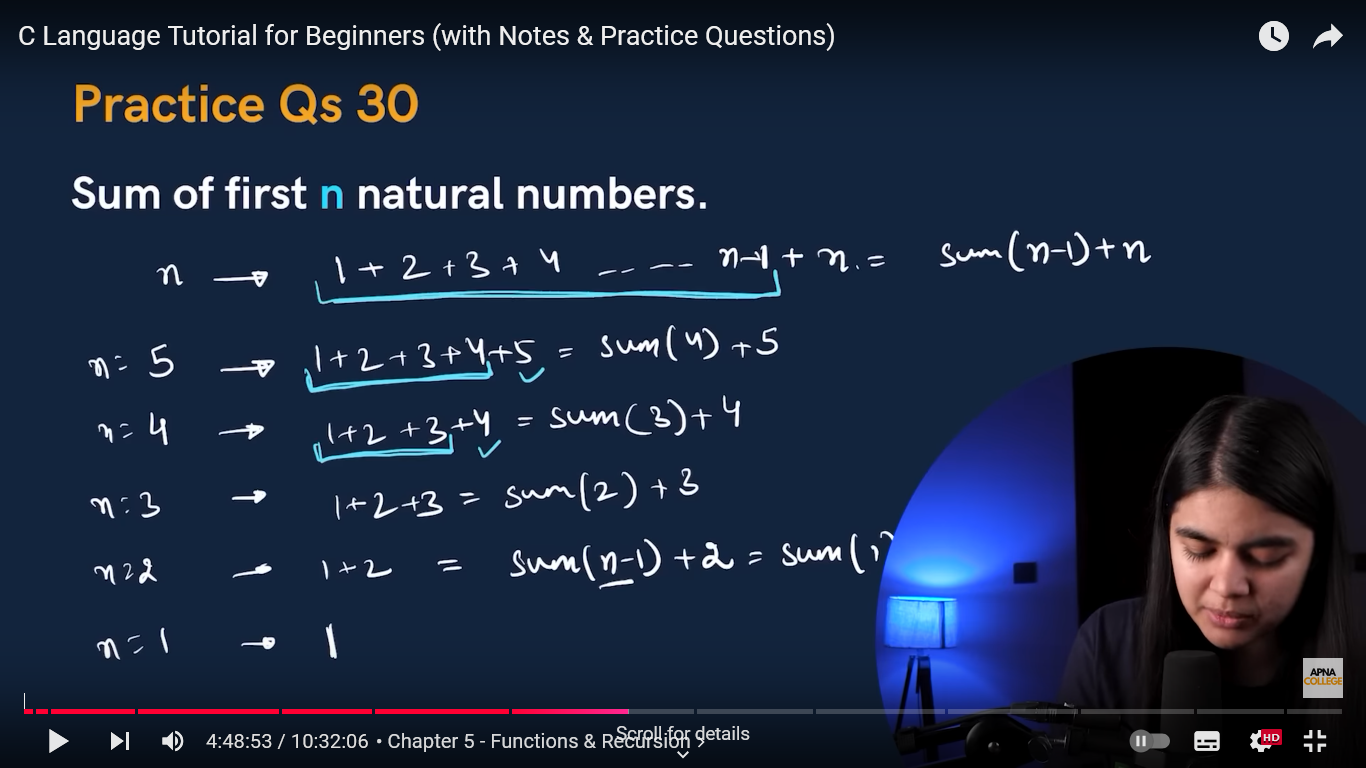
   }

   printf("Badmosi nahi mitr\n");

   printfB(count - 1);

}

**sum of n number using recursion**



int sum(int n);

int main(){

   printf("%d" , sum(6));

   return 0;

}

int sum(int n){

   if(n==1){

      return 1;

   }

   int summ = sum(n-1);

   int sumN = summ + n;

   return sumN;

}

**Factorial of a number**

int fact(int n);

int main(){

   printf("%d" , fact(6));

   return 0;

}

int fact(int n){

   if( n ==1){

      return 1;

   }

   int factN = fact(n-1);

   int factNn = factN \* n;

   return factNn ;

}

int factorial(int n){

   if (n==1 || n==0)

   {

    return 1;

   }//base case

   else{

    return n\*factorial(n-1);

   }//recursion case

}

int main(){

   int number = 5;

   printf("factorial of number of %d is %d" , number ,factorial(number));

}

**Fibonacci series**

int fibonacci(int n){

    if (n==0){

        return 0;

        }

        else if(n==1){

            return 1;

        }

        else {

            return fibonacci(n-1)+fibonacci(n-2);

        }

    }

    int main(){

    int n;

    printf("enter number of element in series" , n);

    scanf("%d", &n);

    for (int i = 0; i < n; i++)

    {

        printf("%d\n" , fibonacci(i));

    }

    }

**POINTERS**

A **variable** that **stores memory address of another variable** is known as a **pointer.**

Different function can have same name variable but they definitely will have different unique address

**Syntax**

 int age = 22;

  int \*ptr = &age;

  int age1 = \*ptr;

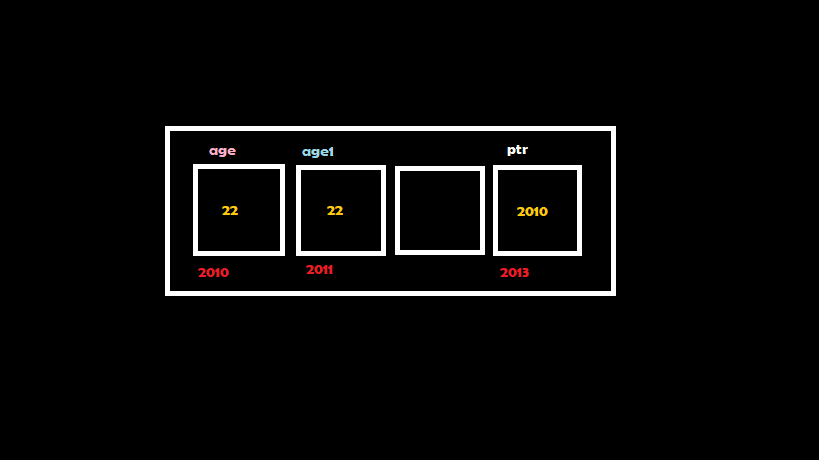
here **\*** is **value at address** operator.

**&age** means **address of age.**

And **ptr (user defined)** is the **variable that stores the memory address of variable age**.

**\*ptr** means **value at address.**

Now that we made a **new variable(age1)** which will store the value at the **address of ptr(\*ptr)**



**Format specifier for pointer would be %p (pointer address)** hexadecimal value **and %u (unsigned integer).**

int age = 17;

  int \*ptr = &age;

 printf("%p" , &age);

**%p will give a hexadecimal value.**

**Output –** 0060FF14

**%u is used to convert hexadecimal to integer so that it is easy to read.**

printf("%u" , &age);

or

printf("%u" , ptr);

**Output –** 6356764

printf("%u" , &ptr);

**Output –** 6356760

**Also**

 printf("%d\n" , age);

   printf("%d\n" , \*ptr);

    printf("%d\n" , \*(&age));

**this will give the same output – 17**

**ex**

 int \*ptr;

    int x;

    ptr = &x;

    \*ptr = 0;

    printf("x is =%d\n" , x);

    printf("\*ptr is =%d\n" , \*ptr);

    \*ptr += 5;

    printf("x is =%d\n" , x);

    printf("\*ptr is =%d\n" , \*ptr);

    (\*ptr) ++;

    printf("x is =%d\n" , x);

    printf("\*ptr is =%d" , \*ptr);

**Output –**

**0**

**0**

**5**

**5**

**6**

**6**

**to pointer**

A **variable** that **stores address** of **another pointer.**

**Syntax**

int \*\*ptr = &age;

int i = 17;

   int \*ptr = &i;

   int \*\*pptr = &ptr;

**here \*\*pptr is used to store the address of ptr.**

**Where \*\* is value at address.**

**Pointers in Function Call**

**Call by Value Call by Reference**

**We pass value of variable as argument. We pass address of variable as argument.**

**Values are copied. Values are same and we access the address**

**directly.**

**(by default)**

**For ex(call by value)**

void square(int n);

int main(){

    int number = 8;

    square(number);

    printf("number = %d", number);

    return 0;

}

void square(int n){

    n = n\*n;

    printf("square = %d\n" , n);

}

**Output –**

**Square = 64**

**Number = 8**

square(number);

**number is the argument**(Arguments are the values that are passed in function call).

The **copy of the argument(**number**)** in **square function** has been **passed.**

That is why the **value of (number) has not changed** in the next line of code**.**

printf("number = %d", number);

**Changes to parameter in function don’t change the value n the calling function or main fn. (**because copy of the argument is passed to the function.**)**

**(refer pg 24)**

**For ex(call by reference)**

void square(int\* n);

int main(){

    int number = 8;

    square(&number);

    printf("number = %d", number);

    return 0;

}

void square(int\* n){

    \*n = (\*n) \* (\*n);

    printf("square = %d\n" , \*n);

}

**Output –**

**Square = 64**

**Number = 64**

This is because **we made the changes directly in the address.**

**Swap of 2 number**

void swap(int a, int b);

int main(){

  int x = 10;

  int y = 20;

 printf("x =%d\n y= %d\n" , x , y);

 swap(x , y);

    return 0;

}

void swap(int a, int b){

   int c = a;

   a = b;

   b = c;

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

}

**This didn’t swap x and y (call by value example)**

**Output –**

**x = 10**

**y = 20**

**a = 20**

**b = 10**

void swap(int \*a, int \*b);

int main(){

  int x = 10;

  int y = 20;

  printf("x =%d\n y= %d\n" , x , y);

 swap(&x , &y);

 printf("x =%d\n y= %d\n" , x , y);

    return 0;

}

void swap(int \*a, int \*b){

   int c = \*a;

   \*a = \*b;

   \*b = c;

}

**Output –**

**x = 10**

**y = 20**

**x = 20**

**y = 10**

it **swapped x and y after calling** the **swap function (call by reference).**

We **use call by reference** when we wantsystem **to return multiple values.**

**Best example**

void ad(int n);

void add(int\* n);

int main(){

    int n = 10;

    printf("%d\n" , &n);// address of n in main function

    ad(n);             // address of n in ad function

    add(&n);          // address of n in add function(same as main function cause we made changes in the addres it self)

    return 0 ;

}

void ad(int n){

    printf("%d\n" , &n);

}

void add(int\* n){

    printf("%d\n" , n);

}

**Output –**

**654343**

**654300**

**654343**

**We use pointers(pass by reference) when we want system to return more than one value(as a function can return one value only at a time).**

void doWork(int a , int b , int \*sum , int \*pdt , int \*avg);

int main(){

    int n = 2;

    int m = 4;

    int sum , pdt , avg;

    doWork( n , m , &sum , &pdt , &avg);

    printf("sum is = %d\n product is = %d\n average is = %d\n" , sum , pdt , avg);

    return 0;

}

void doWork(int a , int b , int\*sum , int\*pdt , int \*avg){

     \*sum = a+b;

     \*pdt = a\*b;

     \*avg = (a+b)/2;

}

**Here in the doWork function**

void doWork(int a , int b , int\*sum , int\*pdt , int \*avg){

**int \*sum … are the pointers and**

     \*sum = a+b;

     \*pdt = a\*b;

     \*avg = (a+b)/2

**\*sum … are the value at address sum …**

**Now in the main function**

 doWork( n , m , &sum , &pdt , &avg);

**we called doWork and called address of sum …**

**max of 3 number using pointer**

void doWork( int \*a ,int \*b , int \*c);

int main(){

    int n = 7;

    int m = 10;

    int o = 12;

    doWork(&n , &m , &o);

    return 0;

}

void doWork(int \*a , int \*b , int \*c){

     if (\*a>\*b && \*a>\*c)

     {

        printf("%d is maximum" , \*a);

     }

     else{

        if (\*b>\*c)

        {

            printf("%d is maximum", \*b);

        }

        else{

            printf("%d is maximum", \*c);

        }

     }

}

**To print alphabets using pointer**

int main(){

 char \*ptr;

 ptr = "abcdefghijklmnopqrstuvwxyz";

 for (int i = 0; i < 26; i++)

 {

   printf("%c " , \*(ptr + i));

 }

 return 0;

}

**ARRAYS**

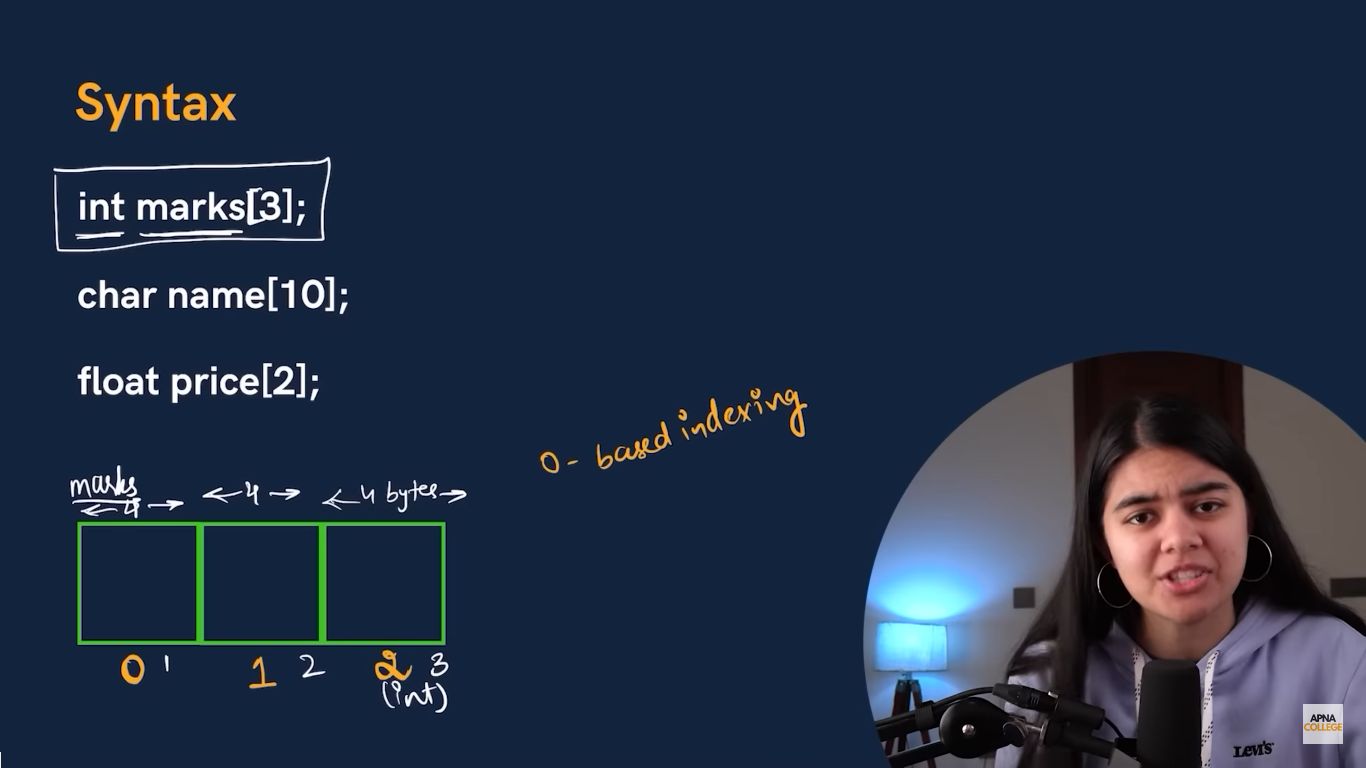
**Collection** of similar datatypes stored at **Contiguous** of similar datatypes

Syntax

Datatype name(userdef) [no.of the data];

int marks[3];

now 3 blocks of 4 byte each will be reserved in the memory location of variable **marks** with **0 based indexing.**



**Input and output**

**input**

scanf("%d" , &marks[3]);

**output**

printf("%d" , marks[3]);

**during input output value inside [] is the position of the data (which is starting from zero)**

**ex**

int marks[5];

    printf("enter marks of English:");

    scanf("%d" , &marks[0]);

    printf("enter marks of Chemistry:");

    scanf("%d" , &marks[1]);

    printf("enter marks of PPs:");

    scanf("%d" , &marks[2]);

    printf("enter marks of BCEM:");

    scanf("%d" , &marks[3]);

    printf("enter marks of Maths:");

    scanf("%d" , &marks[4]);

    printf("English:%d Chemistry:%d PPs:%d BCEM:%d Maths:%d", marks[0],marks[1], marks[2], marks[3],marks[4] );

**gst by user**

int main() {

    float gst;

    float p[3];

    printf("Enter the GST rate (as a percentage): ");

    scanf("%f", &gst);

    gst = gst / 100;

    printf("Enter price of Product 1: ");

    scanf("%f", &p[0]);

    printf("Enter price of Product 2: ");

    scanf("%f", &p[1]);

    printf("Enter price of Product 3: ");

    scanf("%f", &p[2]);

    printf("Product 1: %.2f\n", p[0] + (gst \* p[0]));

    printf("Product 2: %.2f\n", p[1] + (gst \* p[1]));

    printf("Product 3: %.2f\n", p[2] + (gst \* p[2]));

**%.2f**: This indicates the precision of the floating-point number. The .2 means you want to display **2 digits after the decimal point**.

**Initialization of an array**

 int arr[]={60,40,50,100,20};

we need not to declare the size of the array because the system will automatically take the size with the reference of number of elements.

**or**

  int arr[5]={60,40,50,100,20};

you may declare the size but it is not necessary.

**You may need to calculate the memory an array would take.**

**You can find it by**

Memory used by the datatype multiply by the size of the array.

int arr[5];

in this case **int** takes **4 bytes** and **size** is **5**

memory taken by array = **4\*5= 20 bytes**

in the same way **char** takes **1 byte.**

**to print elements of an array in reverse**

int arr []={10 , 20 ,30 , 40 , 50};

int n = sizeof arr / sizeof arr[0];

int main(){

    for (int i = n-1; i >= 0; i--)

    {

       printf("%d\n" , arr[i]);

    }

    return 0;

}

**Pointer Arithmetic**

**Pointers can be increment and decrement**

int age = 22;

  int \*dam = &age;

  printf("ptr = %u\n" , dam);

  dam++;

  printf("ptr = %u" , dam);

**output**

ptr = 6356760

ptr = 6356764

Here dam++ means increase of a datatype value by one

(ex int – by 4 bytes

float – by 4 bytes

Char – by 1 byte

And so on.)

**We can subtract one pointer from other (it will give difference of the datatypes ) , also we can compare them.**

**Array is a pointer**

**\*Array is an address of the 0th position of that array[name of the array is a pointer]**

**So this lines are same**

 int \*a = &arr[0];

 int \*a = arr;

**to take input in an arrays**

int adhaar[5];

 int \*ptr = &adhaar[0];

 for (int i = 0; i < 5; i++)

 {

  printf("%dindex:" , i);

  scanf("%d" , (ptr+i));

 }

 for (int i = 0; i < 5; i++)

 {

  printf("%d index :%d\n" ,i , adhaar[i]);

 }

**Arrays as function argument**

**Function declaration**

 void printnumbers (int arr[] , int n);

OR

 void printnumbers (int \*arr , int n);

**we don’t need to declare size of the array and hence we have to pass int n to declare the size later**

**also {\*arr} can be written in the place of arr[] as array itself is a pointer.**

**Function call**

   printnumbers( arr,10);

#include<stdio.h>

void printnum (int arr[], int n);

int main(){

   int arr[]={1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10};

   printnum( arr,10);

   return 0 ;

}

void printnum (int arr[], int n){

   for (int i = 0; i < n; i++)

   {

      printf("%d\t" , arr[i]);

   }

   printf("\n");

}

**\t is for tab space.**

**Multidimensional array**

**2D Arrays**

  int arr[2][2]={{1,2} , {4,5}};

**access or indexing**

arr[0][0]

arr[0][1]

arr[1][0]

arr[1][1]

**ex**

**CT – 1 , 2**

**English – 24 , 23**

**Maths – 20 , 12**

**Chemistry – 27,18**

**PPS – 28,28**

**BCEM – 16 , 22**

 // 2\* 5

 int arr[2][5]; // \_ \_ \_ \_ \_|\_ \_ \_ \_ \_

 arr[0][0]= 24;

 arr[0][1]= 20;

 arr[0][2]= 27;

 arr[0][3]= 28;

 arr[0][4]= 16;

 arr[1][0]= 23;

 arr[1][1]= 12;

 arr[0][2]= 18;

 arr[1][3]= 28;

 arr[1][4]= 22;

 printf("%d" , arr[1][4]);

**to count the number of odd number in an array**

int count(int arr[],int n);

int main(){

int arr[9]={3,4,5,6,7,8,9,2,3};

   printf("%d", count(arr , 9));

 return 0;

}

int count(int arr[], int n){

   int count = 0;

   for (int i = 0; i < n; i++)

   {

     if (arr[i] % 2 != 0)

     {

      count++;

     }

   }

   return count;

}

**Whats \*(arr + n)**

**ans**

int arr[5]={1,2,3,4,5};

   printf("%d", \*(arr + 1));

   printf("%d", \*(arr + 2));

   printf("%d", \*(arr + 3));

   printf("%d", \*(arr + 4));

**output**

**2**

**3**

**4**

**5**

**To reverse an array**

void reverse (int arr[], int n);

int printarr(int arr[] , int n);

int main(){

 int  arr[]={1 , 2 , 3 , 4 , 5 , 6 };

 reverse(arr, 6);

 printarr(arr , 6);

 return 0;

}

int printarr(int arr[] ,int n){

  for (int i = 0; i < n; i++)

  {

     printf("%d\t" , arr[i]);

  }

   printf("\n");

 }

void reverse (int arr[], int n){

  for (int i = 0; i < n/2; i++)

  {

    int firstval = arr[i];

    int secondval = arr[n-i-1];

    arr[n-i-1] = firstval ;

    arr[i] =  secondval;

  }

}



**To store n Fibonacci series.**

int main(){

  int n;

  printf("enter n(n>2):");

  scanf("%d" , &n);

  int fib[n];

  fib[0]= 0;

  fib[1]= 1;

   printf ("%u\t" , fib[0]);

      printf ("%u\t" , fib[1]);

  for (int i = 2; i < n; i++)

    {

       fib[i]= fib[i-1] + fib[i-2];

       printf("%d\t" , fib[i]);

    }

 return 0;

}

**To store 2 tables in a 2d array**

void storetable (int arr[][10], int n ,int m , int number );

int main(){

  int tables[2][10];

   storetable(tables , 0 , 10 , 9);

   storetable(tables , 1 , 10 , 5);

   for (int i = 0; i < 10; i++)

    {

        printf("%d\t" , tables[0][i]);

    }

    printf("\n");

    for (int i = 0; i < 10; i++)

    {

        printf("%d\t" , tables[1][i]);

    }

 return 0;

}

void storetable (int arr[][10], int n , int m , int number){

    for (int i = 0; i < m; i++)

    {

         arr[n][i]= number \* (i+1);

    }

}

**In an array of numbers find how many times does a number x occur**

int count(int arr[], int n ,int x);

int main(){

   int x;

   int arr[]={2,3,4,2,4,5,3,2,4,7,3,2,5,3,2,4,4,7,7,2,5,7,8,5,3,1,4,6,8,4,2,2};

    int n = sizeof(arr) / sizeof(arr[0]);

   printf("enter number you want to count :");

   scanf("%d" , &x);

   printf("%d" , count(arr , n , x));

   return 0 ;

}

int count(int arr[], int n,int x){

   int count = 0;

   for (int i = 0; i < n; i++)

   {

     if (arr[i] == x)

     {

      count++;

     }

   }

   return count;

}

**Here if we only use sizeof arr then it would pass the address of the the pointer array.**

**S we defined n as**

int n = sizeof(arr) / sizeof(arr[0]);

**so that we can get the size of array(no. of elements).**

**Wap to print the largest number in an array**

int pb(int arr[] , int n);

int main(){

   int arr[]={3432,23,5555,22,400};

   int n = sizeof(arr) / sizeof(arr[0]);

    int largest = pb(arr , n);

   printf("largest number is :%d" , largest);

   return 0 ;

}

int pb(int arr[] , int n){

       int max = arr[0]; // Assuming first element is the largest

    for (int i = 1; i < n; i++) {

        if (arr[i] > max) {

            max = arr[i]; // Updating max if a larger number is found

        }

    }

    return max;

}

**Wap to insert an element at the end of an array.**

void ins(int arr[] , int \*size,int capacity, int element );

int main(){

   int arr[100]={10,20,30,40,50,60};

   int size = 6;

   int element;

   printf("enter the number you want to insert:");

   scanf("%d" , &element);

   ins(arr , &size ,100,  element);

   printf("Updated array: ");

    for (int i = 0; i < size; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

   return 0;

}

void ins( int arr[] , int \*size,int capacity , int element ){

   if (\*size >= capacity) {

        printf("Array is full. Cannot insert more elements.\n");

        return;

        }

   arr[\*size]= element;

   (\*size)++;

}

**Sorting**

The process of arranging data in particular order.

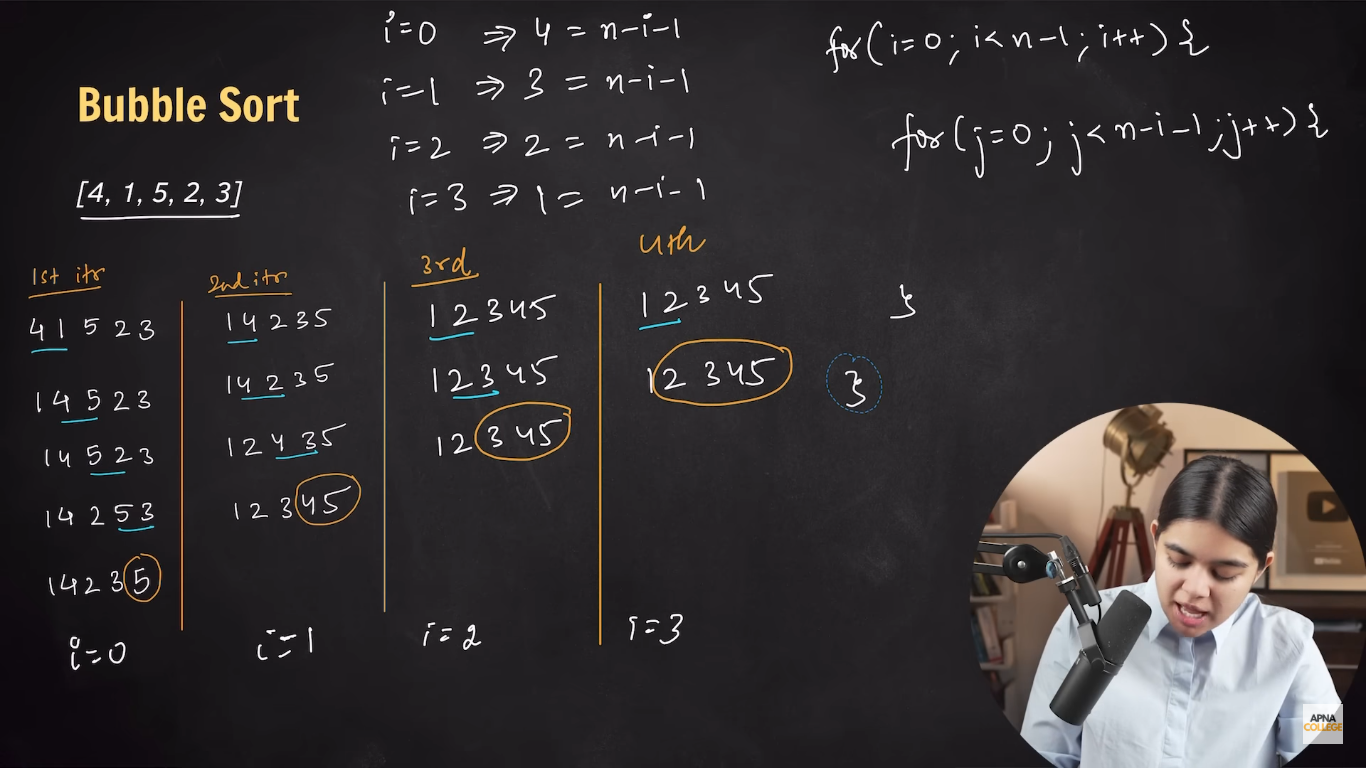
**Bubble Sort**

**It is a technique to sort elements of an array by iterating (n-1) times and leaving behind n-1th largest in the last. Its time complexity is o(n^2)**

**The outer loop of how many iterations be performed**



**The inner loop of how many times each iteration be performed.**



**Code**

void bubblesort(int arr[] , int n);

void printarray(int arr[], int n);

int main(){

   int n = 5;

   int arr[]={58,51,30,70,69};

   bubblesort(arr , n);

   printarray(arr , n);

   return 0;

}

void bubblesort(int arr[] , int n){

      for (int i = 0; i < n-1; i++)

      {

         for (int j = 0; j < n-i-1; j++)

         {

           if (arr[j]>arr[j+1])

           {

            int a = arr[j];

            arr[j]=arr[j+1];

            arr [j+1]= a;

         }

      }

   }

}

      void printarray(int arr[] , int n){

         for (int i = 0; i < n; i++){

           printf(" %d" , arr[i]);

         }

         printf("\n");

      }

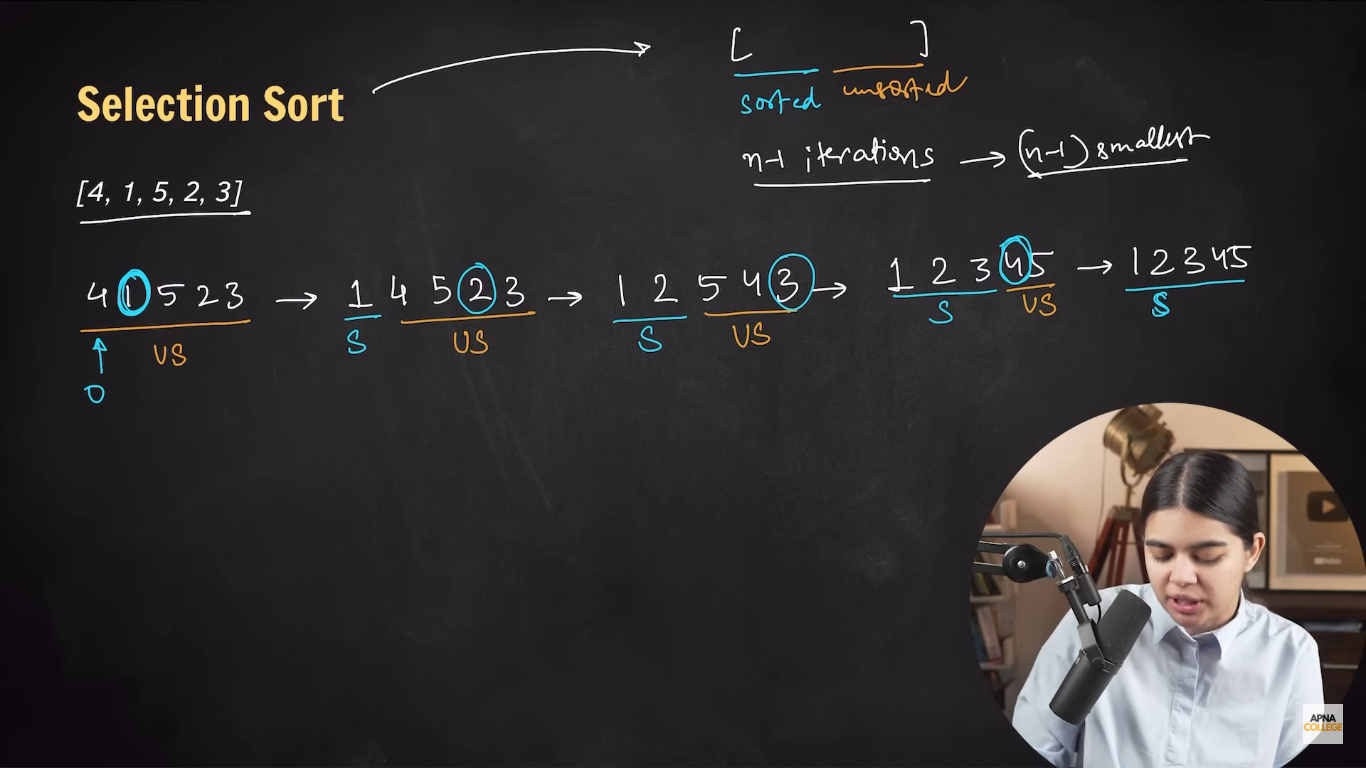
**For descending order we will just change the comparison sign**

if (arr[j]<arr[j+1])

**Selection sort**

**In this sort we divide the array in 2 parts sorted and unsorted. Its time complexity is o(n^2).**

**We find smallest number of the array and swap it with the 1st element which is hence called sorted group or part of the array , after continuing we get sorted array.**



**For every iteration we assume the first element is smallest therefore we compare it with rest of the elements.**

void selectionsort(int arr[] , int n);

void printarray(int arr[], int n);

int main(){

   int n = 5;

   int arr[]={58,51,30,70,69};

   selectionsort(arr , n);

   printarray(arr , n);

   return 0;

}

void selectionsort(int arr[] , int n){

      for (int i = 0; i < n-1; i++)

      {

         int si = i;

         for (int j = i+1; j < n; j++)

         if (arr[j]<arr[si])

         {

          si = j;

         }

         int a = arr[i];

         arr[i]=arr[si];

         arr[si]= a;

      }

   }

      void printarray(int arr[] , int n){

         for (int i = 0; i < n; i++){

           printf(" %d" , arr[i]);

         }

         printf("\n");

      }

**For descending order we will just change the comparison sign**

if (arr[j]>arr[si])

**Insertion sort**

**In this type of sort we assume first element(o index or current variable) is sorted and the rest elements are not sorted. Now we take 1st element of unsorted elements every time and compare it with the sorted element(s) , hence shift the position if fail the comparison.**

So if arr previous is > arr previous +one we have to change the position of elements in the same iteration.

We will start our outer loop from **i=0 to** assume that **2nd element(index 1)** is the first element of unsorted part of that array and **1st element (index 0 )** is assumed to sorted**.**

void insertionsort(int arr[] , int n);

void printarray(int arr[], int n);

int main(){

   int n = 5;

   int arr[]={3,1,9,5,8};

  insertionsort(arr , n);

   printarray(arr , n);

   return 0;

}

void insertionsort(int arr[] , int n){

      for (int i = 1; i < n; i++)

      {

         int current = arr[i];

         int previous= i-1;

         while(previous>=0 && arr[previous]>current){

              arr[previous+1] = arr[previous];

              previous --;

         }

         arr[previous+1]=current;

      }

   }

      void printarray(int arr[] , int n){

         for (int i = 0; i < n; i++){

           printf(" %d" , arr[i]);

         }

         printf("\n");

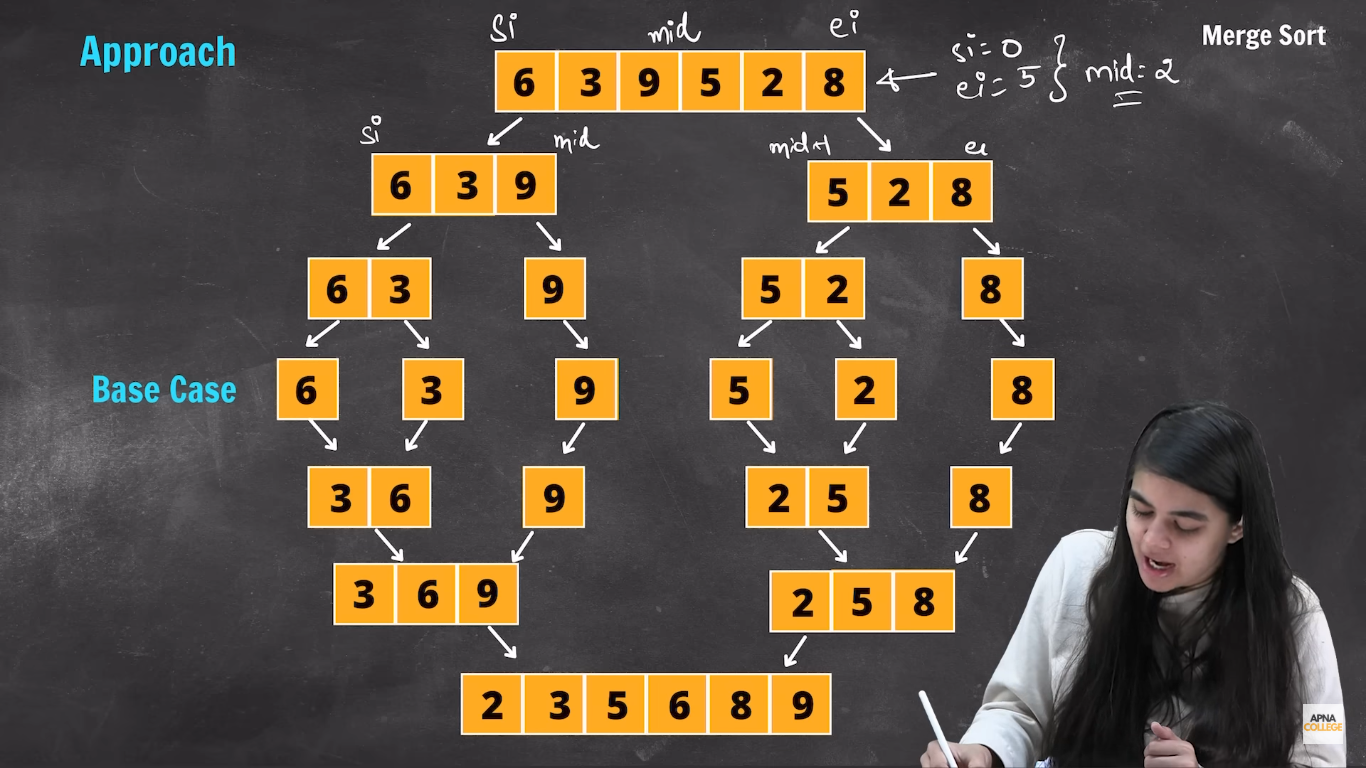
      }

**For descending order we will just change the comparison sign**

while(previous>=0 && arr[previous]<current)

**Merge sort**

This works on the rule of **Divide and Conquer . 1)divide**



**2) conquer**

#include<stdio.h>

void divide(int arr[] , int si , int ei);

void conquer(int arr[] , int si , int mid , int ei);

int main(){

   int arr[]={30,10,95,53,80,42,43,67,27,101};

    int n = sizeof(arr) / sizeof(arr[0]);

  divide(arr , 0 , n-1);

  for(int i = 0; i < n; i++)

  {

  printf("%d ", arr[i]);

  }

   return 0;

}

void divide(int arr[] , int si , int ei){

   if (si>=ei)

   {

      return;

   }

      int mid = si+(ei-si)/2;

      divide(arr , si , mid);

      divide(arr , mid+1 , ei);

      conquer(arr , si , mid , ei);

}

void conquer(int arr[] , int si,int mid , int ei){

   int mergedSize = ei - si + 1;

   int merged[mergedSize];

   int idx1= si;

   int idx2= mid+1;

   int x = 0;

   while (idx1<= mid && idx2<=ei)

           {

            if (arr[idx1]<=arr[idx2])

            {

             merged[x] = arr[idx1];

             x++; idx1++;

            }

            else{

               merged[x++]=arr[idx2++];

            }

           }

   while (idx1<= mid )

           {

           merged[x++]= arr[idx1++] ;

}

   while (idx2<=ei)

{

  merged[x++] = arr[idx2++];

}

          for (int i = 0 , j=si;  i < mergedSize; i++ , j++)

          {

            arr[j] = merged[i];

          }

}

**String**

It is a **character array terminated by a null character (\0)**

When we make a **character array the characters are stored individually** but when we add **null character** at the and of that array **it joint** all character to make **a string.**

 char name[]= {'a','n','j','a','n','i'};

   char nam[]="anjani";

**both are acceptable way of writing strings.**

If we are writing  **character**  we always use **‘’,**

But when we write a **string** we always use **“”.**

**Each character is saved in individual block of address of 1 byte.**

void prtstr(char arr[]);

int main(){

   char firstName[]= "Christain ";

   char lastname[]="Loboutin";

   prtstr(firstName);

   prtstr(lastname);

  return 0;

}

void prtstr(char arr[]){

        for (int i = 0; arr[i] != '\0' ;i++)

   {

     printf("%c" , arr[i]);

   }

}

**In this way we can print our string character by character.**

**We have %s format specifier specially for strings.**(it automatically includes \0 in the string)

**To take an input and print it**

 char name[]={};

 printf("enter:");

   scanf("%s" , name);

   printf("%s" , name);

**no scanf function cannot input strings with spaces even with the %s format specifier.**

So now we will have to study bout

**gets(str)** input (dangerous and outdated)

**puts(str)**  output

**fgets(str , n , file)** input (it stops n-1 characters input or new line is entered.)

   char str[100];

   gets(str);

   puts(str);

**as array is an pointer , strings are character array and hence strings are pointers too.**

**The only difference between them is**

char \*str= "wakey wakey";//can be reinitialized

   char str[]= "wakey wakey";//cannot be reinitialized

  char \*str= "wakey wakey";//can be reinitialized

   puts(str);

   str= "wakey"; //after initializing

   puts(str);

**to get length of string input by user.**

int countlength(char arr[]);

int main(){

 char name[100];

 fgets(name , 100 , stdin);

 printf("length is : %d\n" , countlength(name));

   return 0;

}

int countlength(char arr[]){

   int count = 0;

  for(int i=0; arr[i]!='\0'; i++)

   {

   count ++;

   }

   return count-1;

   }

**To count the number of specific character in a string input by user**

 char word[100];

 fgets(word , 100 , stdin);

 printf("length is : %d\n" , countchar(word));

   return 0;

}

int countchar(char arr[]){

   int count = 0;

  for(int i=0; arr[i]!='\0'; i++)

   {

      if (arr[i]=='i')

      {

         count++;

      }

   }

   return count;

   }

**Standard Library Function**

**we have some string library function for that we have to**

#include<string.h>

1. **Strlen()**

this function is used count the length of string excluding “\0”.

 char word[100];

 fgets(word , 100 , stdin);// to get input string

 char name[] = "vedikaaa"; [// to](file:///\\to) count already given string

 printf("length is :%d\n" , strlen(word)-1);

 printf("length is :%d" , strlen(name));

**2 strcpy(newstr , oldstr)**

To copy the value of old string to new string.

Old ki value new me copy hojaegi

 char oldstr[]= "bbb";

  char newstr[]= "habibi";

  strcpy(newstr,oldstr);

  puts(newstr);

**3 strcat(firststr,secstr)**

Concatenates (joins) first string with second string.

 char oldstr[]= "bbb";

  char newstr[]= "habibi";

  strcat(newstr,oldstr);

  puts(newstr);

note that no space will be printed between them unless we have a space after 1st string.

**4 strcpm**

It compare 2 strings which is passed and returns an integer

**0 – if strings are equal**

**+ve – if first > second**

**-ve – if first < second**

**()** it compares based on ASCII values.

char str1[]= "HHH";

char str2[]= "HHH";

printf("%d" , strcmp(str1,str2));

**output – 0**

char str1[]= "HHHA";

char str2[]= "HHHB";

printf("%d" , strcmp(str1,str2));

**output – -1**

char str1[]= "HHHB";

char str2[]= "HHHA";

printf("%d" , strcmp(str1,str2));

**output – 1**

**To input a string from user using (%c).**

  char str[100];

   char ch;

   int i=0;

   while (ch !='\n')

   {

    scanf("%c" , &ch);

    str[i]= ch;

    i++;

   }

   str[i]='\0';

   puts(str);

**this code act as gets / fgets.**

**Salts**

**{the way in which passwords are secured are called salting.}**

Salt for a company or cooperate can be different which is known to the security sector of that company , it is randomly allocated in between front or end of the password input by the user so that if the company system is hacked still the hacker does not know the password.

**To find the salted form of password entered by user if the salt is 123 & added in the end.**

char pass[20];

 printf("enter password:");

scanf("%s" , &pass);

 char salt[] = "123";

 printf("Your password is:%s" , strcat(pass , salt));

**or**

void salting(char password[]);

int main(){

   char pass[20];

 printf("enter password:");

 scanf("%s" , &pass);

 salting(pass);

  return 0;

}

void salting(char password[]){

   char salt[] = "123";

   char newpass[100];

   strcpy(newpass , password);

   strcat(newpass , salt);

   puts(newpass);

}

**Slicing**

**to return slicing the main string**

void slice(char word[] , int n , int m);

int main(){

char word[200];

printf("Enter word:");

scanf("%s" , &word);

slice(word , 3 , 6);

  return 0;

}

void slice(char word[] , int n , int m){

   char neword[100];

   int j =0;

   for (int i = n; i < m; i++ , j++)

   {

    neword[j]=word[i];

   }

   neword[j]='\0';

   puts(neword);

}

**To count the number of vowel and consonant.**

int countvowel(char word[]);

int main(){

char word[100];

printf("Enter word:");

scanf("%s" , &word);

int vowels = countvowel(word);

int consonants = strlen(word) - vowels;

printf("Number of vowels: %d\n", vowels);

countvowel(word);

printf("Number of consonants: %d\n", consonants);

}

int countvowel(char word[]){

   int count = 0;

   for (int i = 0; word[i]!= '\0'; i++)

   {

     if (word[i] == 'a'|| word[i] == 'e'|| word[i] == 'i'||word[i] == 'o'|| word[i] == 'u' )

     {

      count++;

     }

   }

  return count;

}

**To check whether a character is present**

int check(char word[] , char a[]);

int main()

{

   char a[10];

char word[100];

printf("Enter word:");

scanf("%s" , &word);

printf("Enter char:");

scanf("%s" , &a);

if(check(word , a)){

   printf("char is present");

   }

else{

   printf("char is not present");

}

return 0;

}

int check(char word[] , char a[]){

for (char i= 0; word[i] != '\0' ; i++)

{

  if (word[i] == a[0])

  {

  return 1;

  }

}

return 0;

}

**ASCII Representation of Characters:**

* In the ASCII table, lowercase letters (a-z) have higher values than their corresponding uppercase letters (A-Z).
* The difference between a lowercase and uppercase letter is **32**.
  + 'a' = 97 and 'A' = 65 → Difference = **32**
  + 'e' = 101 and 'E' = 69 → Difference = **32**
  + 'i' = 105 and 'I' = 73 → Difference = **32**
  + 'o' = 111 and 'O' = 79 → Difference = **32**
  + 'u' = 117 and 'U' = 85 → Difference = **32**

**To convert all vowel in a string to upper case**

int transform(char word[]);

int main(){

char word[100];

printf("Enter word:");

scanf("%s" , &word);

transform(word);

printf("%s" , word);

return 0;

}

int transform(char word[]){

   for (int i = 0; word[i] != '\0'; i++)

   {

     if (word[i] == 'a'|| word[i] == 'e'|| word[i] == 'i'||word[i] == 'o'|| word[i] == 'u' )

     word[i]= word[i] - 32;

   }

}

**To print a input number’s digits sum and product**

int main(){

   int a;

   printf("enter number");

   scanf("%d" , &a);

   int n = a%10;

     int m = a/10;

   for (int i = n; n<=0 ; i++)

   {

     n = a%10;

   }

   int sum = m+n;

   int product =m\*n;

   printf("%d\n" , sum);

   printf("%d" , product);

}

**Structures**

It is a **collection of data** with **different datatypes.**

It comes under the category of **user defined datatype.**

#include <stdio.h>

#include <string.h>

 struct student{

  char name[100];

  int roll;

  float gpa;

 };

int main(){

  struct student s1;

  // s1.name = 'captain'; wrong!!

  s1.roll = 10;

  s1.gpa = 8.9;

  strcpy(s1.name , "captain");

  printf("student name = %s\n" , s1.name);

  printf("student roll = %d\n" , s1.roll);

  printf("student gpa = %f" , s1.gpa);

  return 0;

}

#include <stdio.h>

#include <string.h>

 struct student{

  char name[100];

  int roll;

  float gpa;

 };

int main(){

  struct student s1;

  s1.roll = 10;

  s1.gpa = 8.9;

  strcpy(s1.name , "captain");

  printf("student name = %s\n" , s1.name);

  printf("student roll = %d\n" , s1.roll);

  printf("student gpa = %f\n" , s1.gpa);

  struct student s2;

  s2.roll = 11;

  s2.gpa = 8.2;

  strcpy(s2.name , "elon");

  printf("student name = %s\n" , s2.name);

  printf("student roll = %d\n" , s2.roll);

  printf("student gpa = %f\n" , s2.gpa);

  struct student s3;

  s3.roll = 10;

  s3.gpa = 8.9;

  strcpy(s3.name , "eddy");

  printf("student name = %s\n" , s3.name);

  printf("student roll = %d\n" , s3.roll);

  printf("student gpa = %f" , s3.gpa);

  return 0;

}

**Array of Structures**

struct student{

  char name[100];

  int roll;

  float gpa;

 };

int main(){

struct student ECE[100];

struct student CSE[100];

struct student CIV[100];

ECE[0].roll = 01;

ECE[0].gpa = 9.2;

strcpy(ECE[0].name , "anjani" );

printf("name = %s\n  roll = %d\n gpa = %f" , ECE[0].name , ECE[0].roll , ECE[0].gpa);

  return 0;

}

**Instead of using dot operator multiple times we can use initializing of structure**

struct student{

  char name[100];

  int roll;

  float gpa;

 };

int main(){

  struct  student s1 = {"anjani" , 10 , 9.5};

  printf("name = %s\n roll = %d\n gpa = %f" , s1.name , s1.roll , s1.gpa);

  return 0;

}

**Pointers of Structures**

  struct  student s1 = {"anjani" , 10 , 9.5};

  struct student \*ptr = &s1;

  printf("roll = %d\n" , (\*ptr).roll );

struct student{

  int roll;

  float gpa;

  char name[100];

};

int main(){

  struct student s1 ={1160 , 9.3 , "anjani"};

  printf("student roll : %d\n" , s1.roll);

 // or

  struct student  \*ptr = &s1;

   printf("student roll : %d\n" , (\*ptr).roll);

}

**an arrow operator**

(\*ptr).code is same as ptr -> code

  struct student  \*ptr = &s1;

   printf("student roll : %d\n" , (\*ptr).roll);

// or

   printf("student->roll : %d\n" , ptr->roll);

**Passing a structure in a function**

Same as we used to pass datatypes in function as parameters

struct student{

  int roll;

  float gpa;

  char name[100];

};

void printinfo(struct student ptr);

int main(){

  struct student ptr ={1160 , 9.3 , "anjani"};

  printinfo(ptr);

}

void printinfo(struct student ptr){

  printf("student-roll : %d\n" , ptr.roll);

  printf("student-gpa : %f\n" , ptr.gpa);

  printf("student-name : %s\n" , ptr.name);

}

(its a call by value function)

**typedef (**keyword**)**

**used to create alias (nickname).**

typedef struct student{

  int roll;

  float gpa;

  char name[100];

} stu;

**To print sum of 2 vectors**

 struct vector{

  int x ;

  int y;

 };

 void calcsum(struct vector v1 , struct vector v2 , struct vector sum);

int main(){

  struct vector v1 = {5 , 10};

  struct vector v2 = {6 , 12};

  struct vector sum = {0};

  calcsum(v1, v2 , sum);

}

void calcsum(struct vector v1 , struct vector v2 , struct vector sum){

  sum.x= v1.x + v2.x;

  sum.y= v1.y + v2.y;

  printf("sum of x component is - %d\n" , sum.x);

  printf("sum of y component is - %d" , sum.y);

}

**To print complex number**

struct complex{

  int real;

  int img;

};

int main(){

  struct complex n1= {5 ,6};

  struct complex \*ptr = &n1;

  printf("imaginary part is :%di\n" , ptr->img);

   printf("real part is :%d" , ptr->real);

   return 0;

}