infix, prefix and postfix

it is a notation to write an expression

inflix -> <operand> <operator><operand 2>

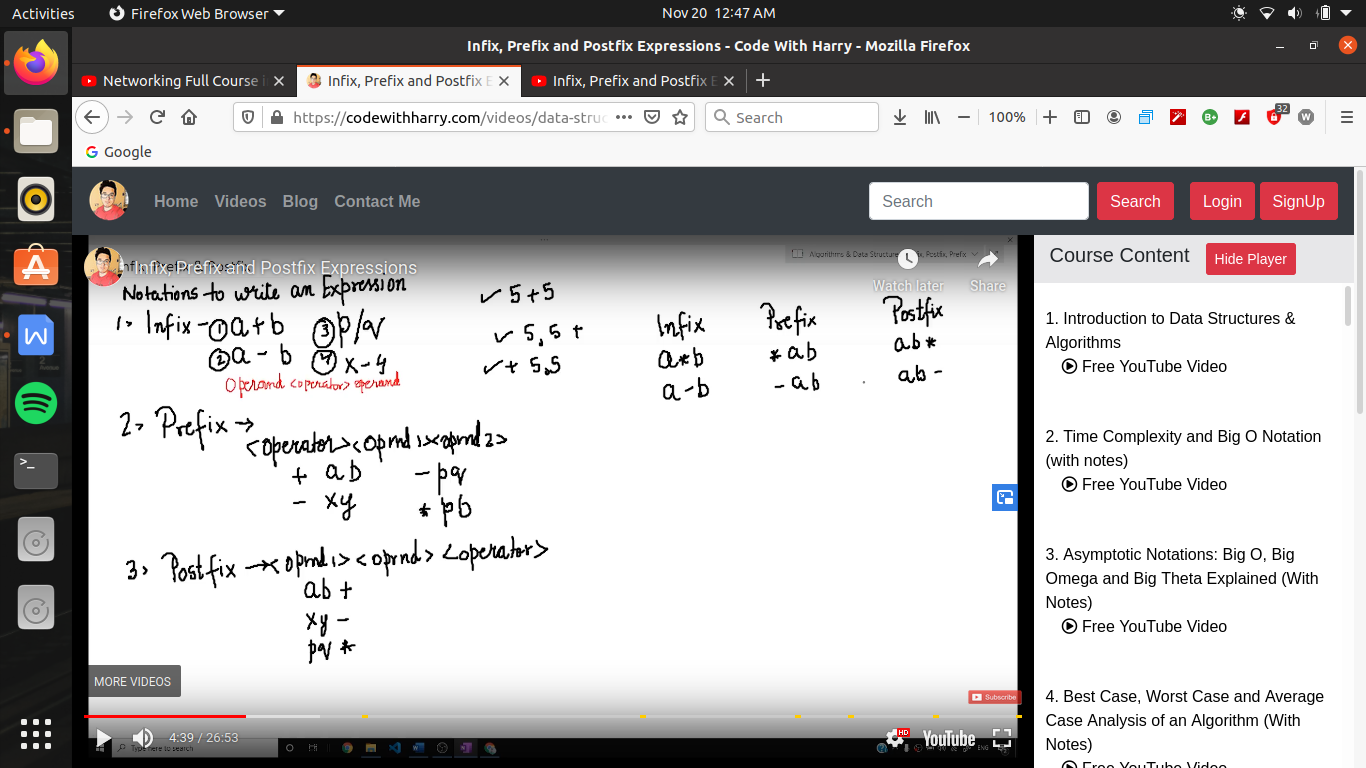
ex- a+b

prefix -> <operator><operand 1><operand 2>

ex -> +ab

postfix -> <operand 1><operand 2><operator>

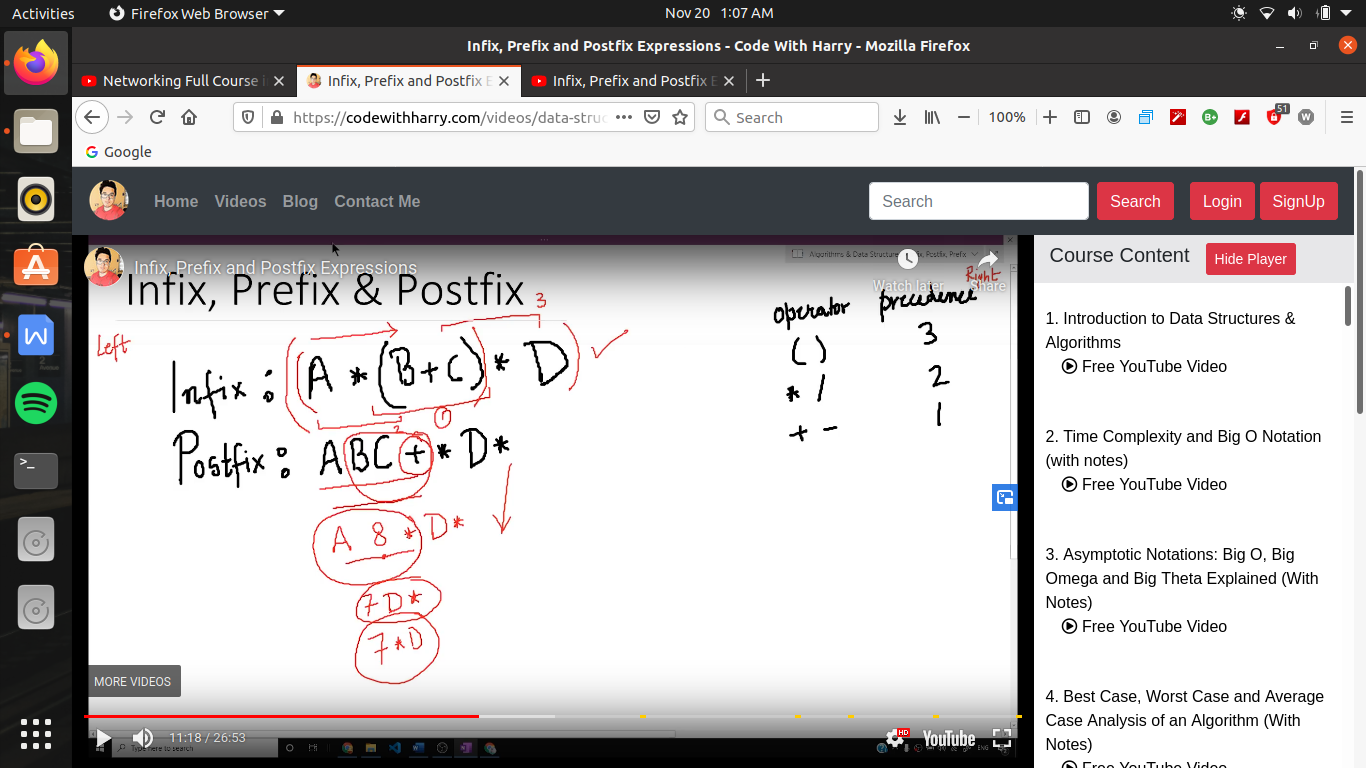
ex -> ab +



machine ke leye postfix or prefix evaluate karana asan hota hai ese liye hum prefix ko postfix me convert karate hai

postfix etana use nhi hota hai jitana postfix hota hai

postfix me operator ke pahale jo bhi 2 value hogi unhe evaluate kr dega



steps to convert infix into prefix or postfix

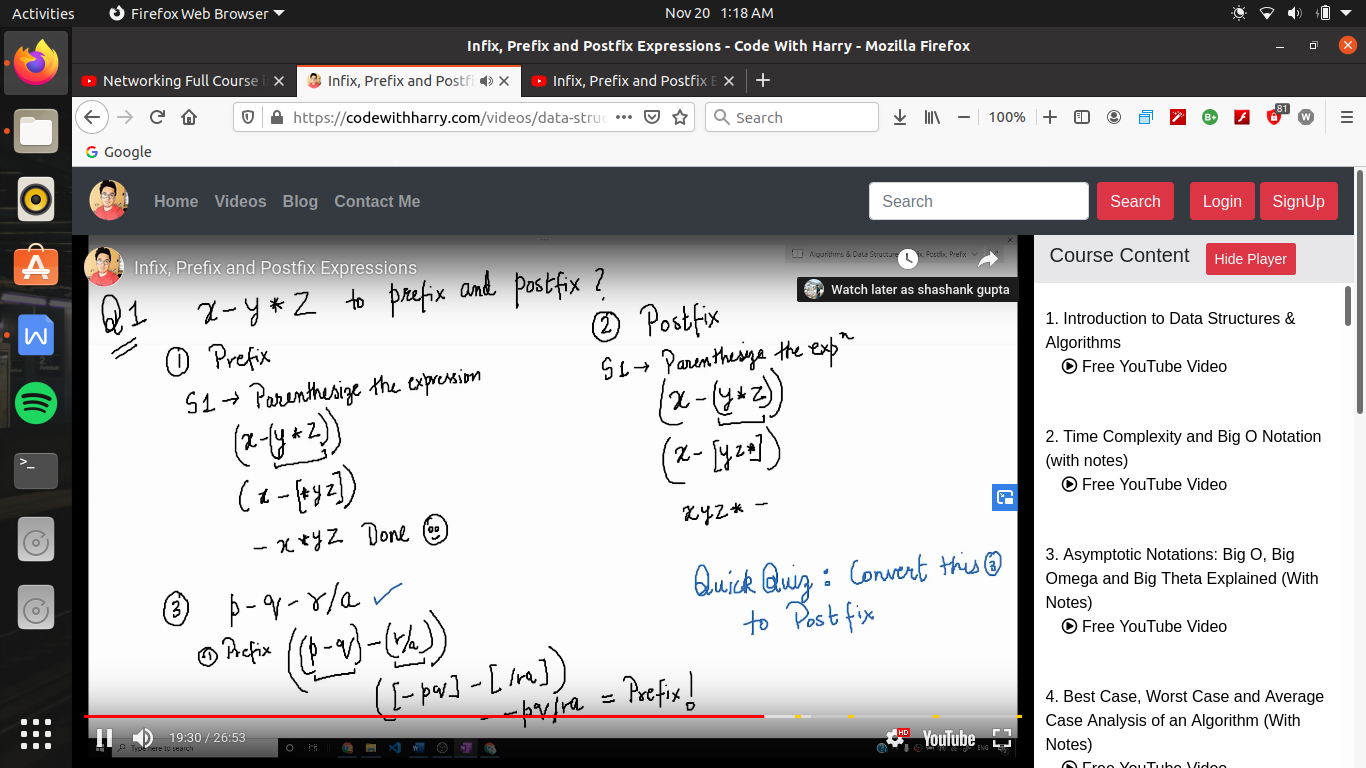
step 1 -> paraenthesis the expresion

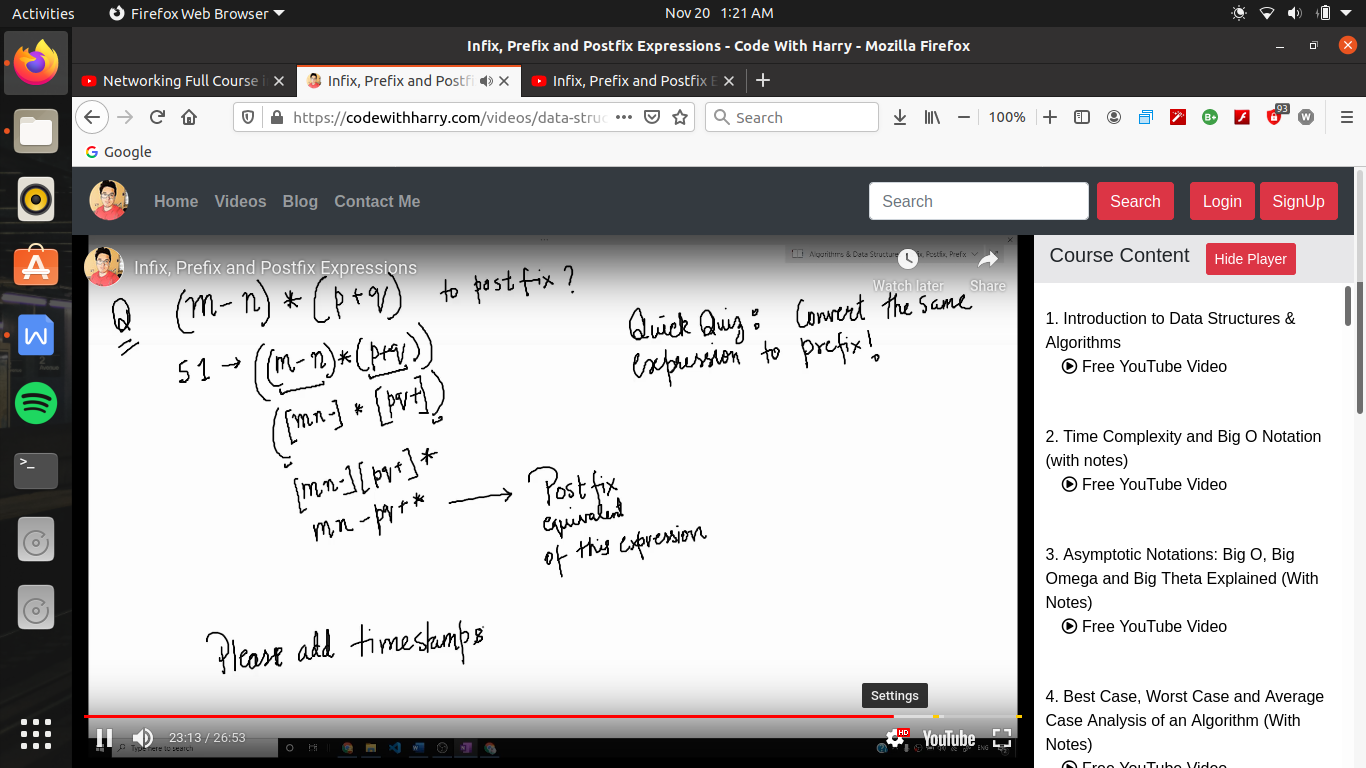
step 2 -> ab inner most bracket me ja kr use parenthis karege

steps 3 -> always consider precedence and associativity

(i.e first apply precedence and the lest to right associtivity)

note -> chota bracket lagate chalana waran confusion hone lagegi





infix to postfix using stack :-

expressioin -> x-y/z-k\*d

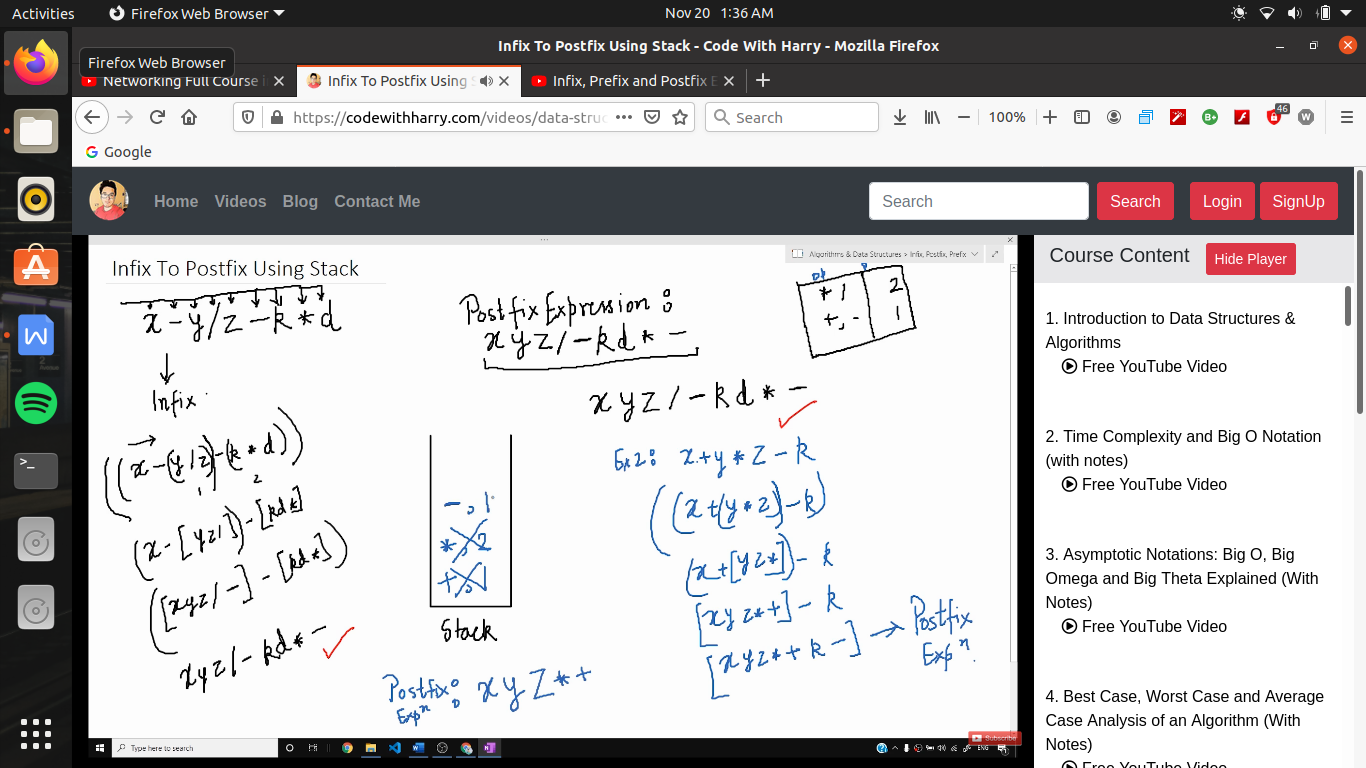
now here we traverse form left to right

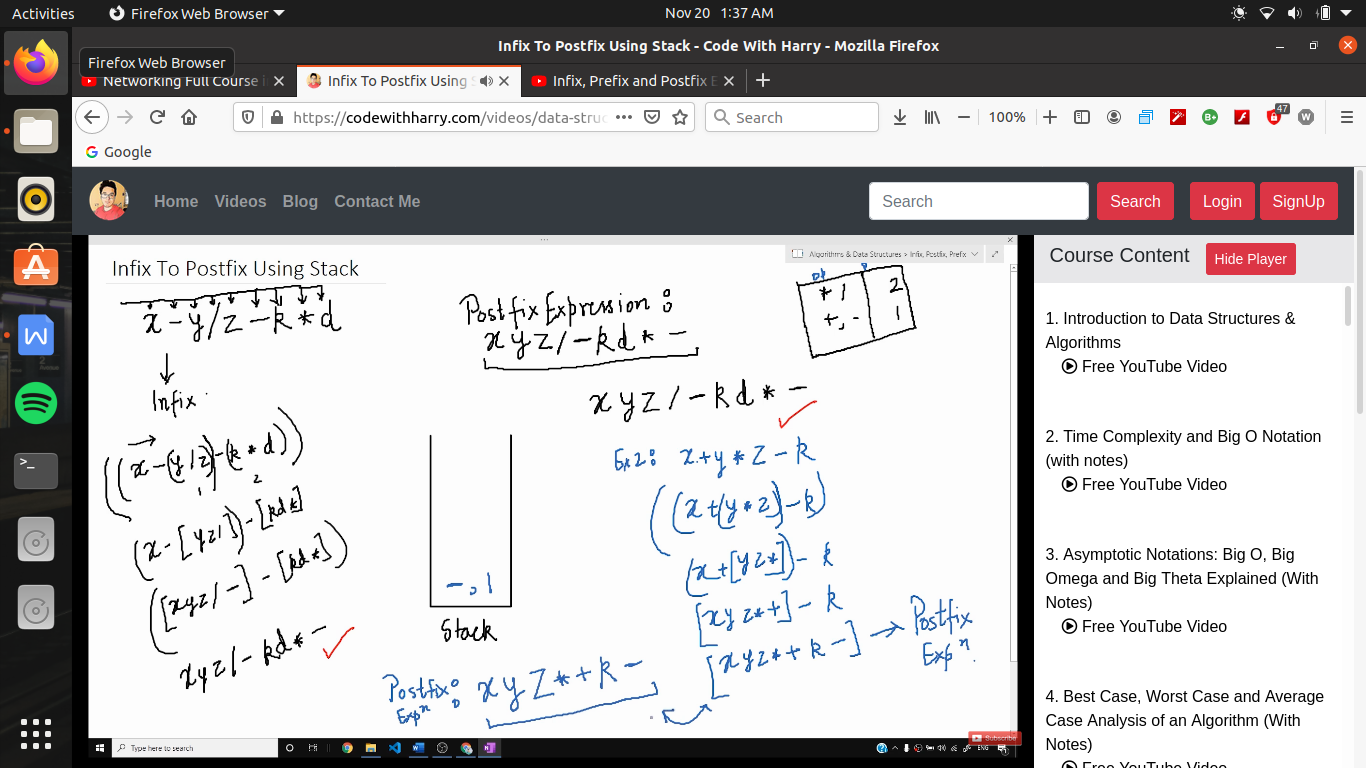
if operant came then we write it in as an expression of postfix

and if operator comes the we push that operator into stack and move further

and if next operator comes then it push that operator only when its has more precedence level as compared to earlier one which already in stack and and if operator have precedence less and equal than the already existent operator in the top of stack so we pop the top operator form the stack

and if expression index reached the last operant then we pop the operators one by one





/\* here we convert the inflix expression to equivalent postfix \*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct stack

{

int top;

int size;

char \*array;

};

int is\_empty(struct stack \*sp)

{

if (sp->top==-1)

return 1;

else

return 0;

}

char stacktop(struct stack \*sp)

{

return sp->array[sp->top];

}

void push(struct stack \*sp, char operant)

{

sp->top++;

sp-> array[sp->top]= operant;

}

char pop(struct stack \*sp)

{

if (is\_empty(sp))

{

printf("stack underflow! cannot pop from the stack \n");

return -1;

}

else

{

char value;

value=sp->array[sp->top];

sp->top--;

return value;

}

}

char \* infix(char \* s)

{

printf("enter the expression \n ");

\_\_fpurge(stdin);

gets(s);

return s;

}

int precedence(char ch)

{

if(ch == '\*' || ch== '/')

return 3;

else if(ch == '+'||ch =='-')

return 2;

else

return 0;

}

int is\_operator(char ch)

{

if (ch== '+' || ch== '-'||ch=='\*'||ch == '/')

return 1;

else

return 0;

}

char \* infix\_to\_postfix(char \* infix)

{

int l ,i=0,j=0;

l=strlen(infix);

char \* postfix=(char \*)malloc((l+1)\*sizeof(char));

struct stack \*sp =(struct stack \*)malloc(sizeof(struct stack));

sp->top =-1;

sp->size=l+1;

sp->array=(char \*)malloc(sp->size \*sizeof(char));

while (infix[i]!='\0')

{

if (!is\_operator(infix[i]))

{

postfix[j]=infix[i];

i++;

j++;

}

else

{

if (precedence(infix[i])>precedence(stacktop(sp)))

{ push(sp,infix[i]);

i++;

}

else

{ postfix[j]=pop(sp);

j++;

}

}

}

while(!is\_empty(sp))

{

postfix[j]=pop(sp);

j++;

}

postfix[j] ='\0';

return postfix;

}

void main()

{

char \*s;

char \*p=(char \*)malloc(sizeof(char));

s=infix(p);

p=infix\_to\_postfix(s);

puts(p);

}

