Lecture10

Infix expression to Prefix expression

To understand, how to convert infix to prefix, watch the presentation.

We assume that we have a stack of characters, with the following functions
1. push(chr) : pushes character 'chr' in stack.
2. x = pop() : Deletes topmost character from stack and stores it in x.
3. x = peep() : Get the value of topmost character from stack and store it in x.
   Remember the differene between Pop() and Peep(). Pop() also gets the topmost value from the stack and decrements top by 1, but Peep() only gets the topmost value from the stack and donot decrement top by 1.

There are 3 character arrays, infixstr, revinfixstr and prefixstr.
Variable 'infixstr' stores infix string entered by user.
Variable 'revinfixstr' stores revers of 'infixstr'.
Variable 'prefixstr' stores the final answer which is prefix of infix string.

Let me summarize what we have learnt from the presenataion.

Each operator has a priority.

'(' has lowest priority of 0.
'+' and '-' has equal priority of 1.
'*' and '/' has equal priority of 2.
'$' has priority of 3.

Our first step is to reverse a string stored in variable 'infixstr', stored the result in 'revinfixstr'. Then find the length of string stored in variable 'revinfixstr'. Scan the string stored in variable 'revinfixstr' starting from 0 until length.

If character scanned is an
1 Operand, like '0', '1': then write it in the string prefixstr.
2 Operator, let it be opr

2.1 If opr is ')' push opr in the stack.

2.2 If opr is '(', start popping operators from stack until ')' comes. Write all the popped operators in string prefixstr.

2.3 If opr is '+' or '–' or '*' or '/' or '$'

2.3.1 If stack is empty, then push opr into Stack.

2.3.2 If stack is not empty

2.3.2.1 Find the priority of opr, let it be a.

2.3.2.2 Find the priority of operator in stack, let it be b.
2.3.2.3 If a >= b,

2.3.2.3.1 just push the current operator.

2.3.2.4 Else if (a < b)

2.3.2.4.1 Keep on popping until a>=b or stack is empty. Then push opr. Write all the popped characters in string revinfixstr.

3 After you reach the end of string 'revinfixstr', pop all the operators from the stack until stack is empty. Write all the popped characters in the string 'prefixstr'.

Here is the program, for converting infix to prefix.
Step1: We assume that we have a stack 'S' of characters with functions
void Push(char)
char Pop()
char Peep( )

Initialization and declaration of necessary varaibles. There are 3 character arrays, infixstr, revinfixstr and prefixstr.
Variable 'infixstr' stores infix string entered by user.
Variable 'revinfixstr' stores revers of 'infixstr'.
Variable 'prefixstr' stores the final answer which is prefix of infix string.

```
char infixstr[20];
char revinfixstr[20];
char prefixstr[20];
```

There is a function priority which returns the priority of operator

```
int priority(char opr)
{
    switch(opr)
    {
        case '('    : return 0;break;
        case '+'    : return 1;break;
        case '-'    : return 1;break;
        case '*'    : return 2;break;
        case '/'    : return 2;break;
        case '$'    : return 3;break;
    }
}
```

Step2: Ask user to enter infix string and store user response in infixstr.

```
printf("Enter infix string");
```

```
scanf("%s", infixstr);
```

Step3: Reverse the string 'infixstr' and store in 'revinfixstr'.

```
revinfixstr = strrev(infixstr)
```

Here is code for function (char *) strrev (char *str),

```
(char *) strrev(char *str)
{
        int i,j;

        int l = strlen(str);
        char *str1 = (char *)malloc(l);
        for(j=l-1,i=0;j>=0;j--;i++)
                str1[i]=str[j];
        str1[i]='\0';
}
```

Step4: Find length of string 'revinfixstr', store in variable 'l'.We will start a loop from 0 until l.

```
int l = strlen(revinfixstr);
for(int i =0;i<l;i++){
```

Step5: Start scanning each character revinfixstr[i], if it is a operand, write operand in string 'prefixstr'.

```
int k=0;
switch(revinfixstr[i])
{
        case '0' : prefixstr[k] = revinfixstr[i]; k++;break;
        case '1' : prefixstr[k] = revinfixstr[i]; k++;break;
        case '2' : prefixstr[k] = revinfixstr[i]; k++;break;
        case '3' : prefixstr[k] = revinfixstr[i]; k++;break;
        case '4' : prefixstr[k] = revinfixstr[i]; k++;break;
        case '5' : prefixstr[k] = revinfixstr[i]; k++;break;
        case '6' : prefixstr[k] = revinfixstr[i]; k++;break;
        case '7' : prefixstr[k] = revinfixstr[i]; k++;break;
        case '8' : prefixstr[k] = revinfixstr[i]; k++;break;
        case '9' : prefixstr[k] = revinfixstr[i]; k++;break;
```

Step6: If it is a operator i.e. ')' or + or - or * or / or $ or '('. If operator is ')' (closing bracket), then Push ')' in stack.

```
        case ')': push(revinfixtr[i]);break;
```

Step7: If it is a operator '(' (open bracket), then start popping from the stack until an closing bracket ')' comes.

Note: When you pop from the stack, write popped character in the string 'prefixstr'. But dont write ')' in string 'prefixstr'.

```
case '(':   char a = pop(); // pop operator from stack
            while(a != '(') // if topmost operator is not '('
            {
                // stored popped character in string prefixstr
                prefixstr[k]=a;
                // increment k by 1
                k++;
                // Pop another character from stack.
                a = pop();

            }
            break;
```

These 3 lines only execute if condition (a != '(') is true

Step8: If it is an operator '+' or '-' or '*' or '/', then
 8.1Find the priority of current operator, let it be a i.e. a = priority().
 8.2Find the priority of operator in stack, let it be b i.e. b = priority(peep());
We used Peep() because we dont want to pop, we only want to know what is the prioirty of topmost character in stack.
if (a>=b) then push revinfixstr[i] into stack.
Else // means a<b
Start popping from the stack until an operator of lower priority comes or stack is empty.

```
            case '+' : decide(revinfixstr[i]);break;
            case '-' : decide(revinfixstr[i]); break;
            case '*' : decide(revinfixstr[i]); break;
            case '/' : decide(revinfixstr[i]);break;
        } // end of switch
} // end of for loop.
```

Here is a function decide()

```
    // This function is outside main()
    void decide(char opr)
    {
            char a = priority(opr); // find the priority of current operator.
            char b = priority(peep()) // find the priority of topmost operator in stack
            if (a >= b) // priorty of a is greater than or equal to b
                    push(opr);
            else
            { /*
```

1. we are here means, priority a is less than b, enter inside while loop.
2. Pop the topmost operator.
3. Write it into stack.
4. Find the priority of new topmost operator again. Go to step1

```
            */
                while(a < b)
                {
                    popr=pop(); // pop the topmost operator.
                    prefixstr[k] = popr;
                    k++;
                    b=priority(peep());
                }
                // when loop finish, that means a>b. So we can push opr now.
                push(opr);
            }
        }
```

Step9: In step8, For loop finished, that means, now we have scanned the string revinfixstr completely and reached at the end of string 'revinfixstr'.
Now, either stack is empty or
if it is not empty,then it consist of operators. Pop all the opreators and write in the string 'prefixstr'.

```
            While(!stack_empty())
            {
                char opr = pop();
                prefixstr[k]=opr;
                k++;
            }
} // end of main
```

Convert Infix to Postfix

Watch the flash presentation for converting infix to postfix. Let me summarize how it works.

We have an infix experssion in string 'infixstr'. Let 'l' be the length of string 'infixstr'. Scan the string 'infixstr' from 0 until l.

1.if it is an operand, write it in string 'postfixstr'.

2 If it is an operator, let it be opr.

2.1 If opr is ')', Start popping operators from stack until '(' comes. Write all the popped operators in string 'postfixstr'.

2.2 Else, if opr is '(', push '(' into stack.

2.3 Else if opr is '+' or '–' or '*' or '/' or '$'

2.4 If stack is empty

2.4.1 Then push opr in the stack

2.5 If stack is not empty

2.5.1.1 then, find the priority of opr, let it be a.

2.5.1.2 Find the priority of operator in stack, let it be b.

2.5.1.3 if(a > b)

2.5.1.3.1 then, push opr into stack.

2.5.1.4 Else (means a<=b)

2.5.1.4.1 pop operators from the stack until a>b or stack is empty.Then push opr into stack. Write all the popped operator in string 'postfixstr'.

3 After we reach the end of string, pop all the characters from stack until stack is empty.Write all the operators in the string 'postfixstr'.

Below is a compete program for converting an infix expression into postfix expression.

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
void decide(char chr, char *pstr, int *pk);
class stack
{
        public :
                int top;
                char arr[10];
        stack()
        {
                top=-1;
        }

        void push(char p);
        char pop();
        char peep();
        int stack_empty();
};



 int stack :: stack_empty()
 {
        if(top==-1)
        return 1;                    // FUNCTION FOR STACK EMPTY
        else
        return 0;
```

```
 }


void stack :: push(char p)
{
        if(top==9)
        cout<<"STACK IS FULL ";             // PUSH
        else
        {
                top++;
                arr[top]=p;
 }

}



char stack :: pop()
{
        int x=top;
        top--;                          // pop
        return arr[x];
}


int priority(char opr)
{
        int p;                  // FUNCTION TO ASSIGN PRIORITY
        switch(opr)
        {
        case '(' : p=0;
                                break;
        case '+' : p=1;
                                break;
        case '-' : p=1;
                                break;
        case '*' : p=2;
                                break;
        case '/' : p=3;
                                break;
        default :  p=-1;
                                break;
}
 return p;
}


char stack::peep()
```

```
{                           //RETURN TOPMOST ELEMENT
return arr[top];
}

void main()
{
        clrscr();
        int a,b;
        char infixstr[30];
        char postfixstr[30];
        int k;
        cout<<"\n\n\t\t ENTER THE INFIX EXPRESSION :  ";
        gets(infixstr);
        cout<<"\n\n\t\t THE EQUIVALENT POSTFIX EXPRESSION IS :\t";
        for(int l=0;infixstr[l]!='\0';l++);
        for(int i=0, k=0;i<l;i++)
        {
                switch(infixstr[i])
                {
                        case '0'      :   postfixstr[k] = infixstr[i];k++;break;
                                                                  break;
                        case '1'      :   postfixstr[k] = infixstr[i];k++;break;
                        case '2'      :   postfixstr[k] = infixstr[i];k++;break;
                        case '3'      :   postfixstr[k] = infixstr[i];k++;break;
                        case '4'      :   postfixstr[k] = infixstr[i];k++;break;
                        case '5'      :   postfixstr[k] = infixstr[i];k++;break;
                        case '6'      :   postfixstr[k] = infixstr[i];k++;break;
                        case '7'      :   postfixstr[k] = infixstr[i];k++;break;
                        case '8'      :   postfixstr[k] = infixstr[i];k++;break;
                        case '9'      :   postfixstr[k] = infixstr[i];k++;break;

                        case '('      :   push(infixstr[i]);break;

                        case ')'      :   char a= s.pop();
                                                  while(a!='(')
                                                  {
                                                        postfixstr[k] = a;
                                                        k++;
                                                        a=s.pop();
                                                  }
                                                  break;

                        case '+'      :  decide(infixstr[i],postfixstr, &k);break;
                        case '-'      :  decide(infixstr[i],postfixstr, &k);break;
                        case '*'      :  decide(infixstr[i],postfixstr, &k);break;
                        case '/'      :  decide(infixstr[i],postfixstr, &k);break;

                } // end of switch-case construct
        } // end of for loop
```

```
            while(!s.stack_empty())
            {
                  a= s.pop();
                  postfixstr[k] = a;
                  k++;

            }
            getch();
}// end of main()

void decide(char chr, char *pstr, int *pk)
{
      char a, b;
      a=priority(chr);
      b=priority(s.peep());
      if(a>b)
            s.push(chr);
      else
      {
            while(a<=b)
            {

                  pstr[*pk] = s.pop();
                  *pk++;
                  b=priority(s.peep());
            }
            s.push(chr);
      }
}
```