```
/*WAP to convert infix to postfix using Stack*/
#include<iostream>
#include<cstring>
using namespace std;
//char stack
char stack[50];
int top = -1;
void push(char item)
{
   stack[++top] = item;
}
char pop()
{
   return stack[top--];
}
//returns precedence of operators
int precedence(char symbol)
{
   switch (symbol)
   {
   case '+':
   case '-':
      return 2;
      break;
   case '*':
   case '/':
      return 3;
      break;
   case '$':
      return 4;
      break;
   case '(':
   case ')':
   case '#':
      return 1;
      break;
   }
```

```
}
//check whether the symbol is operator?
int isOperator(char symbol)
{
   switch (symbol)
   {
   case '+':
   case '-':
   case '*':
   case '/':
   case '$':
   case '(':
   case ')':
      return 1;
      break;
   default:
      return 0;
   }
}
//converts infix expression to postfix
void convert(char infix[], char postfix[])
{
   int i, symbol, j = 0;
   stack[++top] = '#';
   for (i = 0; i < strlen(infix); i++)
   {
      symbol = infix[i];
      if (isOperator(symbol) == 0)
      {
         postfix[j] = symbol;
         j++;
      }
      else
      {
         if (symbol == '(')
         {
            push(symbol);
```

```
      }
      else
      {
         if (symbol == ')')
         {
            while (stack[top] != '(')
            {
               postfix[j] = pop();
               j++;
            }
            pop();//pop out (.
         }
         else
         {
            if (precedence(symbol) > precedence(stack[top]))
            {
               push(symbol);
            }
            else
            {
               while (precedence(symbol) <= precedence(stack[top]))
               {
                  postfix[j] = pop();
                  j++;
               }
               push(symbol);
            }
         }
      }
   }
}
while (stack[top] != '#')
{
   postfix[j] = pop();
   j++;
}
postfix[j] = '\0';//null terminate string.
```

```
}
//int stack
int stack_int[50];
int top_int = -1;
void push_int(int item)
{
   stack_int[++top_int] = item;
}
char pop_int()
{
   return stack_int[top_int--];
}
int main()
{
   char infix[50], postfix[50];
   cout << "Use '+'  ,  '-'  , '*', '/' and '$' (for exponentiation)." << endl;
   cout << "Enter Infix Expression."<<endl;
   cin >> infix;
   convert(infix, postfix);
   cout << "Infix expression is: " <<endl<< infix << endl;
   cout << "Postfix expression is: " <<endl<< postfix << endl;
   return 0;
}

/*WAP to convert infix to postfix using Stack*///or
#include<iostream>
#include<string>
#define max 15
using namespace std;
template<class T>
class Stack
{
   T data[max];
   int top;
public:
   Stack():top(-1) {}
   void push(T value)
```

```
{
   if(top==max-1)
   {
      cout<<"overflow"<<endl;
   }
   else
      data[++top]=value;
}
T pop()
{
   if(top==-1)
   {
      cout<<"underflow"<<endl;
   }
   else
   {
      return data[top--];
   }
}
T peek()
{
   if(top==-1)
   {
      cout<<"underflow"<<endl;
   }
   else
   {
      return data[top];
   }
}
T Top()
{
   return data[top];
}
void display()
{
   cout<<"------------------XX---------------"<<endl;
```

```
        for(int i=top; i>-1; i--)
        {
            cout<<data[i]<<endl;
        }
        cout<<"-----------------XX--------------"<<endl;
    }
};
//precision check
int precision_check(char x)
{
    if(x=='$')
    {
        return 3;
    }
    else if(x=='*' || x=='/')
    {
        return 2;
    }
    else if(x=='+' || x=='-')
    {
        return 1;
    }
    else
    {
        return NULL;
    }
}
//infix expression to postfix expression
string infix_to_postfix(string expression)
{
    Stack<char>converter;
    string postfix;
    char y;
    converter.push('(');
    for(auto x:expression)
    {
        if(x =='('
```

```
      {
         converter.push(x);
      } // if left bracket is encountered
      else if(x == ')')
      {
         while(converter.peek() != '(')
         {
            y=converter.pop();
            postfix+=y;
         }
         converter.pop();
      }
      else if(x =='*' || x =='+' || x =='-' || x =='$' || x=='/')  //if operator is
encounter
      {
         if(converter.peek() =='(' )
         {
            converter.push(x);
         } // if left bracket is at top
         else if(precision_check(x)>precision_check(converter.peek()))
         {
            converter.push(x);
         } // if operator is at top
         else
         {
            y=converter.pop();
            postfix+=y;
            converter.push(x);
         }
      }
      else  //if operand or character is encountered
      {
         postfix+=x;
      }
   }
   return postfix;
}
```

```
//driver function
int main()
{
   string expression;
   cout<<"Enter your expression "<<endl;
   getline(cin,expression);
   expression+=')';
   string x=infix_to_postfix(expression);
   cout<<x<<endl;
}
```