**LABSHEET III**

**PROBLEM 1**

To reverse a string using stack.

ALGORITHM

1. Create a class with member variable top, bottom, input array and size.
2. Class consists of member functions constructor,pop(), push(), check(), input() and pop1(), push1();
3. An object of the class is created.
4. Input is taken from the user.
5. All the elements are pushed into the stack and simultaneously increasing the top ;
6. All elements are popped and added to the result. Decrement top.
7. The result is displayed.

**CODE**

#include<iostream>

using namespace std;

class reverse{

int n, top, bottom;

char \*arr, \*stack, \*result;

public:

reverse(){

top =-1;

bottom=-1;

cout<<"Enter size of the array"<<endl;

cin>>n;

arr = new char[n];

stack = new char[n];

result = new char[n];

}

void input(){

cout<<"Give the input"<<endl;

for(int i = 0; i<n; i++) {

cin>>arr[i];

}

}

void push1(){

for(int i =0; i<n; i++){

push(arr[i]);

}

}

void pop1(){

for(int i =0; i<n; i++){

pop();

}

for(int i =0; i<n; i++)

cout<<result[i];

}

void push(int a){

stack[++top]= a;

}

void pop(){

result[n-top-1]=stack[top];

top--;

}

};

int main(){

reverse r;

r.input();

r.push1();

r.pop1();

}

TESTING

Enter size of the array

12

Give the input

Mynameiskeerthi

ithreeksiemanyM

Process returned 0 (0x0) execution time : 15.010 s

Press ENTER to continue.

**PROBLEM 2**

To implement a program to check the balanced parenthesis using stack.

ALGORITHM

1. Create a stack, an empty array exp( to store the expression) and initialize top as -1.

2. If exp[i] is an open bracket, push exp[i] into the stack and increment top by 1.

3. If exp[i] is a closing bracket, pop out the top element from the stack and decrement top by 1.

4. If exp[i] is a closing bracket and top=-1, increment top by 1.

5. Continue this till the end of expression and if top =-1, then the given expression has balanced parenthesis, else it does not have balanced parenthesis.

CODE

#include<iostream>

#include<string.h>

using namespace std;

char push(string exp,int t)

{

return (exp[t]);

}

char pop(char stack[],int t )

{

return (stack[t]);

}

int main()

{

string exp;

cout<<"Enter an expression : ";

cin>>exp;

int n;

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

{

n=i+1;

}

char stack[n];

int top= -1,pot=-1;

for(int i=0;i<n;i++)

{

if(top<n)

{

if(exp[i]=='(')

{

stack[i]=push(exp,i);

top++;

}

if( (exp[i]==')')&&(top!=-1) )

{

stack[i]=pop(stack,i-1);

top--;

}

}

else

cout<<"Stack overflow\n";

}

if(top==-1)

{

cout<<"The paranthesis is balanced\n";

}

else

cout<<"The paranthesis is not balanced\n";

}

TESTING

a) For balanced

Enter an expression : (a+b)\*c

The paranthesis is balanced

b) For unbalanced

Enter an expression : ((a+b)

The paranthesis is not balanced

**PROBLEM 3**

Implement the program for converting infix to postfix form.

**ALGORITHM**

1. Array , stack and result are created using dynamic memory allocation.
2. Input is taken from the user.
3. Initialize I to 0 .
4. If array[i] is an operand the append it to the result.
5. If it is an operator , then check if the stack is empty then push(array[i])

if it is non empty check if top element is not opening bracket or precedence of top element is greater than or equal array[i] pop() and append to result until the precedence is reversed.

1. If it is an opening bracket then push it in the stack
2. if it is a closing bracket then pop() till top element's Is opening bracket.
3. Pop out all the remaining elements and append it to the result.
4. Out put the result.

**CODE**

#include<iostream>

using namespace std;

class in\_to\_post{

int n, top, bottom,count;

char \*arr, \*stack, \*result;

public:

in\_to\_post(){

top =-1;

bottom=-1;

count = 0;

cout<<"Enter size of the array"<<endl;

cin>>n;

arr = new char[n];

stack = new char[n];

result = new char[n];

}

void input(){

cout<<"Give the input"<<endl;

for(int i = 0; i<n; i++) {

cin>>arr[i];

}

}

int preced(char ch){

switch(ch){

case '-':

case '+': return 1;break;

case '\*':

case '/': return 2; break;

case '^': return 3; break;

}

}

void push(int a){

stack[++top]= a;

}

void pop(){

if(stack[top]!='('){

result[count]=stack[top];

count++;

top--;

}

else

top--;

}

bool check\_cat(char ch){

switch(ch){

case '+':

case '-':

case '/':

case '\*':

case '^':

case ')':

case '(':return 0;break;

default : return 1;break;

}

}

void output(){

while(top>-1)

pop();

top =-1;

bottom = -1;

int i = 0;

while(i<n){

cout<<result[i];

i++;

}

cout<<endl;

}

void check(){

int i = 0 ;

while(i<n){

if(check\_cat(arr[i])){

result[count]=arr[i];

count++;

}

else{

if(top==-1){

push(arr[i]);

bottom =0;

}

else {

if(arr[i]==')'){

if(arr[i]==')')

pop();

while(stack[top]!='(')

pop();

}

else if((preced(arr[i]))<=(preced(stack[top]))&&(stack[top]!='(')){

pop();

push(arr[i]);

}

else

push(arr[i]);

} }

i++;

}

}

};

int main(){

in\_to\_post pt;

pt.input();

pt.check();

pt.output();

return 0;

}

TESTING

Enter size of the array

21

Give the input

((a+b)\*c-(d-e))/(f+g)

ab+c\*de--fg+/

Process returned 0 (0x0) execution time : 30.792 s

Press ENTER to continue.

**PROBLEM 4**

Implementing a program for converting infix to prefix.

ALGORITHM

1.create an empty stack.

2.store the given expression in a character array arr[n-1]

3.Create a result array

4.initilize I as length -1;

1. if arr[i] is an operand append it to the result
2. If arr[i] is an operator , check whether the stack is empty, if it is , push (arr[i])
3. check if top90 is not closing bracket and top(0 has higher precedence than arr[i], pop() and append it to the result
4. if it is an closing bracket push(arr[i])
5. if it is an opening bracket , pop() until top() becomes closing bracket .
6. goto step 7
7. decrement I by I and go to step 5 repeat until I = 0;
8. pop() all the elements and append to result
9. print the actual result from end of the array (right to left )

CODE

#include<iostream>

using namespace std;

class in\_to\_pre{

int n, top, bottom,count;

char \*arr, \*stack, \*result;

public:

in\_to\_pre(){

top =-1;

bottom=-1;

count = 0;

cout<<"Enter size of the array"<<endl;

cin>>n;

arr = new char[n];

stack = new char[n];

result = new char[n];

}

void input(){

cout<<"Give the input"<<endl;

for(int i = 0; i<n; i++) {

cin>>arr[i];

}

}

int preced(char ch){

switch(ch){

case '-':

case '+': return 1;break;

case '\*':

case '/': return 2; break;

case '^': return 3; break;

}

}

void push(int a){

stack[++top]= a;

}

void pop(){

if(stack[top]!=')'){

result[count]=stack[top];

count++;

top--;

}

else

top--;

}

bool check\_cat(char ch){

switch(ch){

case '+':

case '-':

case '/':

case '\*':

case '^':

case ')':

case '(':return 0;break;

default : return 1;break;

}

}

void output(){

while(top>-1)

pop();

top =-1;

bottom = -1;

int i = n-1;

while(i>=0){

cout<<result[i];

i--;

}

cout<<endl;

}

void check(){

int i = n-1 ;

while(i>=0){

if(check\_cat(arr[i])){

result[count]=arr[i];

count++;

}

else{

if(top==-1){

push(arr[i]);

bottom =0;

}

else {

if(arr[i]=='('){

if(arr[i]=='(')

pop();

while(stack[top]!=')')

pop();

}

else if((preced(arr[i]))<=(preced(stack[top]))&&(stack[top]!=')')){

pop();

push(arr[i]);

}

else

push(arr[i]);

}

}

i--;

}

}

};

int main(){

in\_to\_pre pt;

pt.input();

pt.check();

pt.output();

return 0;

}

TESTING

Enter size of the array

11

Give the input

a-b/(c+d\*e)

-a/b+c\*de

Process returned 0 (0x0) execution time : 17.939 s

Press ENTER to continue.

**PROBLEM 5**

Implement a program to evaluate Postfix expression

ALGORITHM

1. Create a class called evaluation with data members exp, res toop, stack pointer and n(for size).
2. It consists of member functions, input(), push(), pop(), size(), chech() and output().
3. A constructor initializes top to -1.
4. Input()
5. Takes an postfix expression as an input and stores it in exp.
6. Size of the expression is calculated and stored in n .
7. stack is created using dynamic memory.
8. Push(string, int)

returns the number ( since it is a char, it subtracts the ascii value of 0 )

which gets stored in the stack.

Pop(int ,int)

takes in stack and index and returns the element in that index

size(string)

takes in string and returns the length of the string.

Initialize counter to 0, till '\0' is reached counter is incremented and counter Is returned.

9.Check()

till the counter is less than the n(size)

if top is less than n

* 1. if the exp[i] lies between 48 and 58
  2. push the element in the stack and increment top
  3. if exp[i] is '+' then pop the two elements from the stack and perform addition and store is stack[top-1] and decrement the top
  4. if exp[i] is '-' then pop the two elements from the stack and perform subtract first popped from the second popped and store is stack[top-1] and decrement the top
  5. if exp[i] is '\*' then pop the two elements from the stack and perform multiplication and store is stack[top-1] and decrement the top
  6. if exp[i] is '/' then pop the two elements from the stack and perform division , ie. Divide second popped by first popped and store is stack[top-1] and decrement the top

else show error message of overflow

output()

top element of the stack + 0 is returned.

10.In main()

An object of class evaluation is created input(), check() and output() is called.

CODE

#include<iostream>

#include<cstring>

using namespace std;

class evaluation {

string exp,res;

int top, \*stack,n;

public:

evaluation():top(-1){}

void input() {

cout<<"Enter any postfix expression (eg: abc+-) : ";

cin>>exp;

n = size(exp);

stack = new int[n];

}

int push(string exp,int t)

{

return ((int)(exp[t]-48));

}

int pop(int stack[],int t )

{

return (stack[t]);

}

int size(string exp)

{

int i;

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

return i;

}

void check() {

for(int i=0;i<n;i++)

{

if(top<n)

{

if( (exp[i]>=48) && (exp[i]<58) )

{

stack[top+1]=push(exp,i);

top++;

}

if(exp[i]=='+')

{

stack[top-1]=(pop(stack,top-1))+(pop(stack,top));

top--;

}

if(exp[i]=='-')

{

stack[top-1]=(pop(stack,top-1))-(pop(stack,top));

top--;

}

if(exp[i]=='\*')

{

stack[top-1]=(pop(stack,top-1))\*(pop(stack,top));

top--;

}

if(exp[i]=='/')

{

stack[top-1]=(pop(stack,top-1))/(pop(stack,top));

top--;

}

}

else

cout<<"Stack overflow\n";

}

}

void output() {

cout<<"Result = "<<(stack[top]+0);

}

};

int main() {

evaluation e;

e.input();

e.check();

e.output();

return 0;

}

TESTING

Enter any postfix expression (eg: abc+-) : 123+\*

Result = 5

**PROBLEM 6**

Implement a program to evaluate Prefix expression

ALGORITHM

1. Create a class called evaluation with data members exp, res toop, stack pointer and n(for size).
2. It consists of member functions, input(), push(), pop(), size(), chech() and output().
3. A constructor initializes top to -1.
4. Input()
5. Takes an postfix expression as an input and stores it in exp.
6. Size of the expression is calculated and stored in n .
7. stack is created using dynamic memory.
8. Push(string, int)
9. returns the number ( since it is a char, it subtracts the ascii value of 0 )
10. which gets stored in the stack.
11. Pop(int ,int)
12. takes in stack and index and returns the element in that index
13. size(string)
14. takes in string and returns the length of the string.
15. Initialize counter to 0, till '\0' is reached counter is incremented and counter Is returned.
16. Check()
17. counter is initialized to size of the exp
18. till the counter becomes zero
19. if top is less than n
    1. if the exp[i] lies between 48 and 58
    2. push the element in the stack and increment top
    3. if exp[i] is '+' then pop the two elements from the stack and perform addition and store is stack[top-1] and decrement the top
    4. if exp[i] is '-' then pop the two elements from the stack and perform subtract second popped from the first popped and store is stack[top-1] and decrement the top
    5. if exp[i] is '\*' then pop the two elements from the stack and perform multiplication and store is stack[top-1] and decrement the top
    6. if exp[i] is '/' then pop the two elements from the stack and perform division , ie. Divide first popped by second popped and store is stack[top-1] and decrement the top
20. else show error message of overflow
21. output()
22. top element of the stack + 0 is returned.
23. In main()
24. An object of class evaluation is created e
25. input(), check() and output() is called.

CODE

#include<iostream>

#include<cstring>

using namespace std;

class evaluation {

string exp,res;

int top, n, \*stack;

public:

evaluation():top(-1){}

void input() {

cout<<"Enter any prefix expression (eg: +-abc) : ";

cin>>exp;

n = size(exp);

stack = new int [n];

}

int push(string exp,int t) {

return ((int)(exp[t]-48));

}

int pop(int stack[],int t ) {

return (stack[t]);

}

int size(string exp) {

int i;

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

return i;

}

void check() {

for(int i=n-1;i>=0;i--) {

if(top<n) {

if( (exp[i]>=48) && (exp[i]<58) ) {

stack[top+1]=push(exp,i);

top++;

}

if(exp[i]=='+') {

stack[top-1]=(pop(stack,top-1))+(pop(stack,top));

top--;

}

if(exp[i]=='-') {

stack[top-1]=(pop(stack,top-1))-(pop(stack,top));

top--;

}

if(exp[i]=='\*') {

stack[top-1]=(pop(stack,top-1))\*(pop(stack,top));

top--;

}

if(exp[i]=='/') {

stack[top-1]=(pop(stack,top-1))/(pop(stack,top));

top--;

}

}

else

cout<<"Stack overflow\n";

}

}

void output() {

cout<<"Result = "<<(stack[top]+0);

}

};

int main() {

evaluation e;

e.input();

e.check();

e.output();

return 0;

}

TESTING

Enter any prefix expression (eg: +-abc) : +-999\*3

Result = 9