Compiler Design (18CSC304J)

Experiment 10

Intermediate code Generation for Postfix and Prefix expression

Name: Rahul Goel
Reg No: RA1911030010094

Aim: A program to implement Intermediate code generation – Postfix, Prefix.

Language: C+

+ Procedure:

1. Declare set of operators.
2. Initialize an empty stack.
3. To convert INFIX to POSTFIX follow the following steps
4. Scan the infix expression from left to right.
5. If the scanned character is an operand, output it.
6. Else, If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.
7. Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack.
8. If the scanned character is an '(', push it to the stack.
9. If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis.
10. Pop and output from the stack until it is not empty.
11. To convert INFIX to PREFIX follow the following steps
12. First, reverse the infix expression given in the problem.
13. Scan the expression from left to right.
14. Whenever the operands arrive, print them.
15. If the operator arrives and the stack is found to be empty, then simply push the operator into the stack.
16. Repeat steps 6 to 9 until the stack is empty.

Code Snippet:

```
#include<iostream>

#include<string>

using namespace

std;


template<class

T> class stack

{

        int top;

        int Size;

        T *STACK;

        bool overFlow()

        {

                if (top == this->Size - 1)

                        return

                true; return false;

        }

public
:
        stack(
        )

        {       Size=10;

                STACK=new

                T[Size]; top = -1; //

                initial value

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

                        STACK[i] = 0;

        }



        stack(int Size)
```

{

```
        this->Size=Size;
        STACK=new T[this->Size];
        top = -1; //initial value
        for (int i = 0; i < this->Size; i++)
                STACK[i] = 0;
}

int getSize()
{
        return (top + 1);
}

bool isEmpty()
{
        if (top == -1)
                return
        true; return false;
}

void topp(T& assignTopValue)
{
        if (!isEmpty())
        {
                assignTopValue = STACK[top];
        }
}

void push(T element)
{
        top++;
        STACK[top] = element;
}
```

```
          void pop()
          {
                    if (top == -1)
                    {
                              cout << "cannot pop out element from empty stack.\n";

                    }
                    els
                    e
                    {         STACK[top] = 0;
                              top--;


                    }
          }
          ~stack(
          )
          {
                    delete[] STACK;
          }
};


bool isBalanced(string exp)
{
          stack<char> s;
          if (exp[0] == ')' || exp[0] == ']' || exp[0] ==
                    '}') return false;
          s.push(exp[0]);
          for (int i = 1; i < exp.length(); i++)
          {
                    char check;
                    s.topp(check);


                    if (check == '(' && exp[i] == (char)(check +
                              1)) s.pop();
```

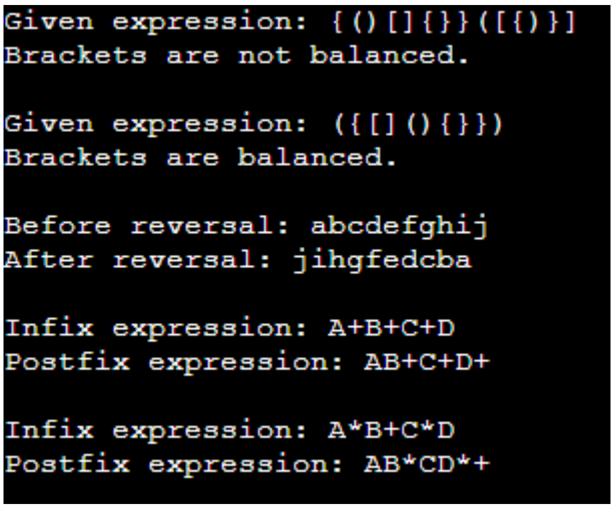
```
                else if ((check == '[' || check == '{') && exp[i] == (char)(check +
                        2)) s.pop();

               else
                        s.push(exp[i]);
        }
        if (s.isEmpty())
                return true;
        return false;
}

string reverse(string const str)
{
        s  t  r  i  n  g
        r e v S t r i n g = " " ;
        revString.resize(1
        0); stack<char> s;
        for (int i = 0; i < str.length(); i++)
        {
                s.push(str[i]);
        }
        for (int i = 0; i < str.length(); i++)
        {
                char val;
                s.topp(val);
                s.pop();
                revString[i] = val;
        }
        return revString;
}

bool precedence(char o1, char o2)
{
        if ((o1 == '*' || o1 == '/') && (o2 == '+' || o2 == '-'))//o1>o2
```

```
                return true;
        else if (((o1 == '*' || o2 == '/') && (o1 == '/' || o2 == '*')) || ((o1 == '+' || o2 == '-') && (o1 ==
        '-'
|| o2 == '+')))//o1==o2
                return true;
        else
                return false;
}


string INtoPOST(string exp)
{
        stack<char>
        s; string post
        = ""; int j = 0;
        for (int i = 0; i < exp.length(); i++)
        {
                char symbol =
                0; symbol =
                exp[i];
                int symb= static_cast<int>(symbol);
                if ((symb >= 65 && symb <= 90) || (symb >= 97 && symb <= 122))
                {
                        post.resize(j +
                        1); post[j] =
                        exp[i]; j++;
                }
                els
                e
                {       char check=0;
                        s.topp(check);

                        while (!s.isEmpty() && precedence(check, exp[i]))
                        {
                                s.topp(check);
```

```
post.resize(j + 1);
```

```
                    post[j] = check; j+
                    +;
                    s.pop();
                }
                s.push(exp[i]);
            }

        }

        while (!s.isEmpty())
        {
            post.resize(j +
            1);
            s.topp(post[j]);
            j++;
            s.pop();
        }
        return post;
}

int main()
{
        cout<<"Given expression: {()[]{}}([{)}]\n";
        bool ff=isBalanced("{()[]{}}([{)}]"); if
        (ff == true)
            cout << "Brackets are balanced.\n\n";
        else
            cout << "Brackets are not balanced.\n\n";

        cout<<"Given expression: ({[]()}})\n";
        bool f=isBalanced("({[]()}})");
        if (f == true)
            cout << "Brackets are balanced.\n\n";
        else
                cout << "Brackets are not balanced.\n\n";


        cout<<"Before reversal: abcdefghij\n";
        cout<<"After reversal: "<<reverse("abcdefghij")<<endl<<endl;


        cout<<"Infix expression: A+B+C+D\n";
        cout<<"Postfix expression: "<<INtoPOST("A+B+C+D")<<endl<<endl;


        cout<<"Infix expression: A*B+C*D\n";
        cout<<"Postfix expression: "<<INtoPOST("A*B+C*D")<<endl<<endl;
        return 0;
}
```

Output:



```
Given expression: {()[]{}}([{)}]
Brackets are not balanced.

Given expression: ({[](){}})
Brackets are balanced.

Before reversal: abcdefghij
After reversal: jihgfedcba

Infix expression: A+B+C+D
Postfix expression: AB+C+D+

Infix expression: A*B+C*D
Postfix expression: AB*CD*+
```

Result: The code was successfully implemented and output was recorded.