Ex No:10
Date:21-04-2021

Intermediate Code Generation- Postfix, Prefix

AIM: To generate intermediate code for a given expression.

PROGRAM:

1) Three Address Code Generator

```
#include <stdio.h>
#include <string.h>
void pm();
void plus();
void div();
void pm()
{
strrev(exp);
j = l - i - 1;
strncat(exp1, exp, j);
strrev(exp1);
printf("Three address code:\ntemp=%s\ntemp1=%c%ctemp\n", exp1, exp[j + 1], exp[j]);
}
void div()
{
strncat(exp1, exp, i + 2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n", exp1, exp[i + 2], exp[i + 3]);
}
void plus()
{
strncat(exp1, exp, i + 2);
printf("Three address code:\ntemp=%s\ntemp1=temp%c%c\n", exp1, exp[i + 2], exp[i + 3]);
}
int i, ch, j, l, addr = 100;
char ex[10], exp[10], exp1[10], exp2[10], id1[5], op[5], id2[5];
int main()
{
while (1)
{
printf("\n1.Assignment\n2.Arithmetic\n3.Relational\n4.Exit\nEnter the choice:");
scanf("%d", &ch);
switch (ch)
{
case 1:
```

```
printf("\nEnter the expression with assignment operator:");
scanf("%s", exp);
l = strlen(exp);
exp2[0] = '\0';
i = 0;
while (exp[i] != '=')
{
i++;
}
strncat(exp2, exp, i);
strrev(exp);
exp1[0] = '\0';
strncat(exp1, exp, l - (i + 1));
strrev(exp1);
printf("Three address code:\ntemp=%s\n%s=temp\n", exp1, exp2);
break;
case 2:
printf("\nEnter the expression with arithmetic operator:");
scanf("%s", ex);
strcpy(exp, ex);
l = strlen(exp);
exp1[0] = '\0';
for (i = 0; i < l; i++)
{
if (exp[i] == '+' || exp[i] == '-')
{
if (exp[i + 2] == '/' || exp[i + 2] == '*')
{
pm();
break;
}
else
{
plus();
break;
}
}
else if (exp[i] == '/' || exp[i] == '*')
{
div();
break;
}
}
break;
```

```
case 3:
printf("Enter the expression with relational operator:");
scanf("%s%s%s", &id1, &op, &id2);
if (((strcmp(op, "<") == 0) || (strcmp(op, ">") == 0) || (strcmp(op, "<=") == 0) || (strcmp(op,
">=")
== 0) || (strcmp(op, "==") == 0) || (strcmp(op, "!=") == 0)) == 0)
printf("Expression is error");
else
{
printf("\n%d\tif %s%s%s goto %d", addr, id1, op, id2, addr + 3);
addr++;
printf("\n%d\t T:=0", addr);
addr++;
printf("\n%d\t goto %d", addr, addr + 2);
addr++;
printf("\n%d\t T:=1", addr);
}
break;
case 4:
break;
}
}
return 0;
}
```

2) Infix to Prefix and Postfix

```
#include<bits/stdc++.h>
using namespace std;

//Function to return precedence of operators
int prec(char c)
{
        if(c == '^')
        return 3;
        else if(c == '*' || c == '/')
        return 2;
        else if(c == '+' || c == '-')
        return 1;
        else
        return -1;
}
```

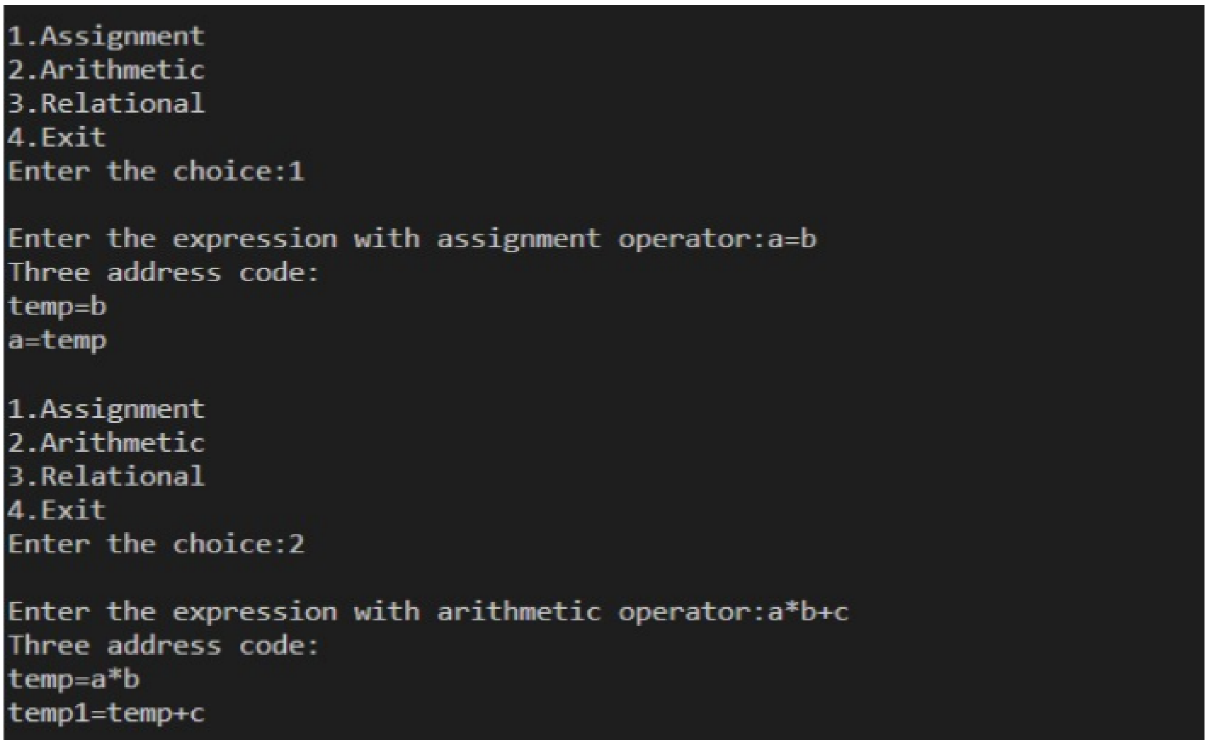
```
//to postfix expression
string infixToPostfix(string s)
{
        std::stack<char> st;
        st.push('N');
        int l = s.length();
        string ns;
        for(int i = 0; i < l; i++)
        {

                // If the scanned character is
     // an operand, add it to output string.

                if((s[i] >= 'a' && s[i] <= 'z') ||
                (s[i] >= 'A' && s[i] <= 'Z'))
                ns+=s[i];

 // If the scanned character is an
     // '(', push it to the stack.

                else if(s[i] == '(')
                 // If the scanned character is an ')',
     // pop and to output string from the stack
     // until an '(' is encountered.

                st.push('(');


                else if(s[i] == ')')
                {
                        while(st.top() != 'N' && st.top() != '(')
                        {
                                char c = st.top();
                                st.pop();
                        ns += c;
                        }
                        if(st.top() == '(')
                        {
                                char c = st.top();
                                st.pop();
                        }
                }

                //If an operator is scanned
```

```
		else{
			while(st.top() != 'N' && prec(s[i]) <=prec(st.top()))
			{
				char c = st.top();
				st.pop();
				ns += c;
			}
			st.push(s[i]);
		}

	}


	while(st.top() != 'N')
	{
		char c = st.top();
		st.pop();
		ns += c;
	}


//	cout << ns << endl;
	return ns;

}


//infix to prefix

bool isOperator(char c)
{
	return (!isalpha(c) && !isdigit(c));
}


string infixToPrefix(string infix)
{

	int l = infix.size();


	reverse(infix.begin(), infix.end());
```

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

                if (infix[i] == '(') {
                        infix[i] = ')';
                        i++;
                }
                else if (infix[i] == ')') {
                        infix[i] = '(';
                        i++;
                }
        }

        string prefix = infixToPostfix(infix);

        // Reverse postfix
        reverse(prefix.begin(), prefix.end());

        return prefix;
}

int main()
{
        string exp = "a+b*(c^d-e)^(f+g*h)-i";
        cout<<"postfix : "<<infixToPostfix(exp)<<endl;
        cout<<"prefix : "<<infixToPrefix(exp);
        return 0;
}
```

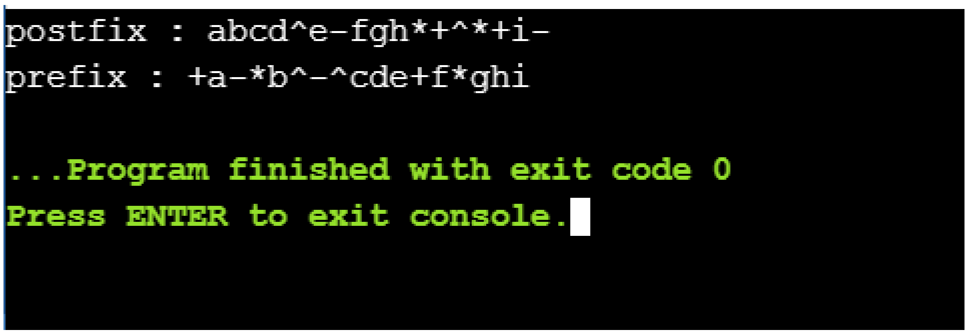
OUTPUT:

1) Three Address Code Generator



```
1.Assignment
2.Arithmetic
3.Relational
4.Exit
Enter the choice:1

Enter the expression with assignment operator:a=b
Three address code:
temp=b
a=temp

1.Assignment
2.Arithmetic
3.Relational
4.Exit
Enter the choice:2

Enter the expression with arithmetic operator:a*b+c
Three address code:
temp=a*b
temp1=temp+c
```

2) Infix to Prefix and Postfix



```
postfix : abcd^e-fgh*+^*+i-
prefix : +a-*b^-^cde+f*ghi

...Program finished with exit code 0
Press ENTER to exit console.
```

Result: The intermediate code generation is implemented successfully.