BINUS UNIVERSITY
BINUS INTERNATIONAL

Students Information :

First Name :	Last Name :	Student ID :
Fiqhy	Bismadhika	2101714824
Jerdy	Tjandra	2101718450
Jose	Valenchee	2101718482

Course Code : COMP6048 **Course Name :** Data Structures

Class : L2BC/B2BC **Lecturer(s) :** 1. Maria Seraphina
2. Raymond Kosala

Major : Computer Science **Teacher Assistant** : Vincent Seliang

Assignment's Title : Infix to Postfix Converter **Type of Assignment** : Final Project

Due Date : --/06/2018 **Submission Date** : 04/06/2018

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student	Signature of Student	Signature of Student
Fiqhy Bismadhika	Jerdy Tjandra	Jose Valenchee

“Infix to Postfix Calculator”

C++ Language Program

Team:

- Jose Vallenchee
- Fiqhy Bismadhika
- Jerdy Tjandra

This documentation describes the details of an Infix to Postfix application.

Problem description

Implementing infix to postfix converter in c++ and shows the detailed process of each step or changes of the Postfix output

Alternative Data Structures

The program uses dynamic container vector from the library to store the data instead of stack which Postfix is linked with, And use basic String type to make the final output of the algorithm.

Theoretical Analysis

- Array implements a compile-time non-resizeable array.Vector implements an array with fast random access and an ability to automatically resize when appending elements.

- Array does not support element insertion or removal. Vector supports fast element insertion or removal at the end. Any insertion or removal of an element not at the end of the vector needs elements between the insertion position and the end of the vector to be copied. The iterators to the affected elements are thus invalidated. In fact, any insertion can potentially invalidate all iterators. Also, if the allocated storage in the vector is too small to insert elements, a new array is allocated, all elements are copied or moved to the new array, and the old array is freed.

- Once a chunk of memory is allocated for an array , its size will remain fixed unless you reallocate or free the memory. Size of vector is not fixed .When new elements are inserted, if the new size of the vector becomes larger than its capacity, *reallocation* occurs. This typically causes the vector to allocate a new region of storage, move the previously held elements to the new region of storage, and free the old region.

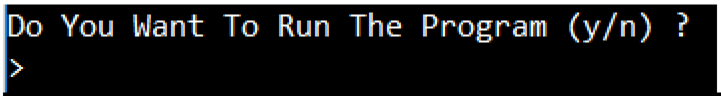
- The vector maintains a certain order of its elements, so that when a new element is inserted at the beginning or in the middle of the vector, subsequent elements are moved backwards in terms of their assignment operator or copy constructor. Consequently, references and iterators to elements after the insertion point become invalidated.

- Memory accesses are similar in both (constant time).

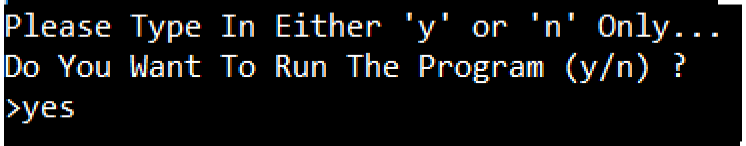
Solution

Scan the infix expression from left to right. If the scanned character is an operand, output it; otherwise, push it if the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty). Otherwise, pop the operator from the stack until the precedence of the scanned operator is less-equal to the precedence of the operator residing on the top of the stack. Push the scanned operator to the stack. If the scanned character is an '(', push it to the stack. If the scanned character is an ')', pop and output from the stack until an '(' is encountered. Repeat steps 2-6 until infix expression is scanned. Pop and output from the stack until it is not empty.

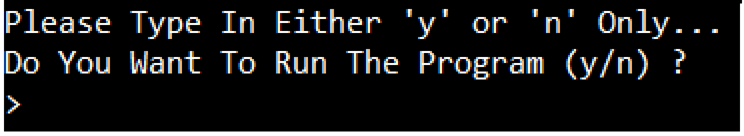
Results of the execution



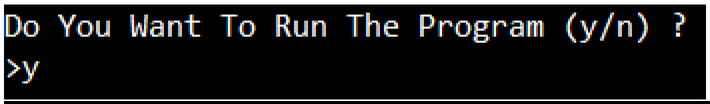
```
Do You Want To Run The Program (y/n) ?
>
```



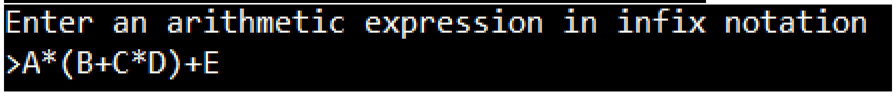
```
Please Type In Either 'y' or 'n' Only...
Do You Want To Run The Program (y/n) ?
>yes
```



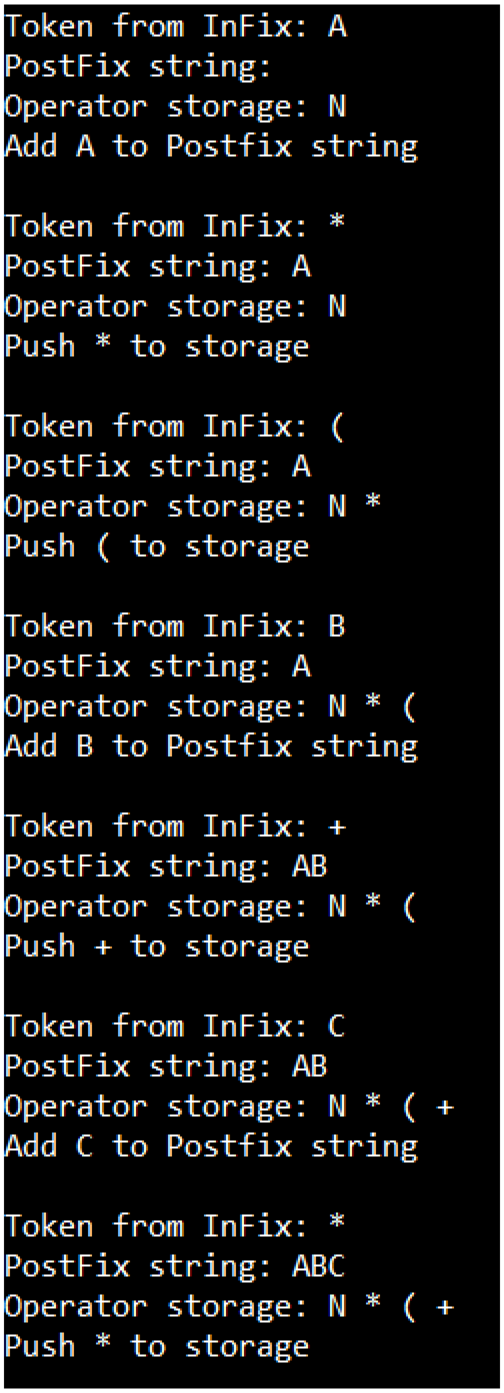
```
Please Type In Either 'y' or 'n' Only...
Do You Want To Run The Program (y/n) ?
>
```

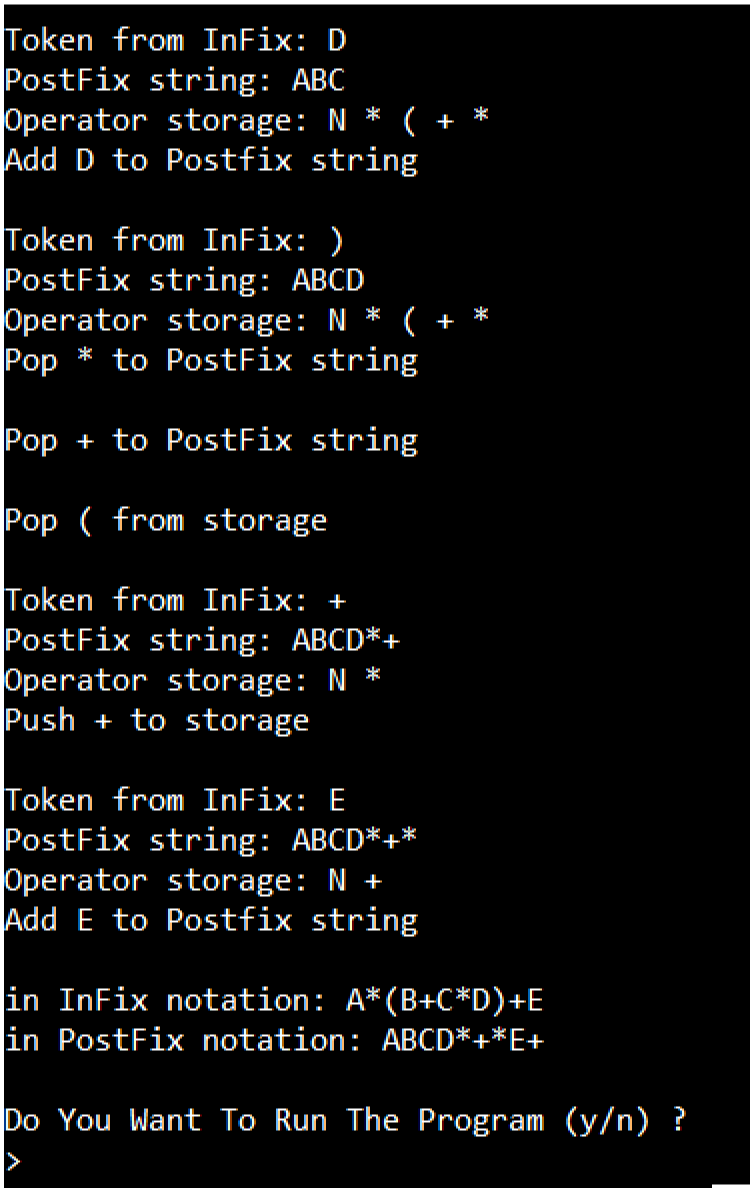


```
Do You Want To Run The Program (y/n) ?
>y
```

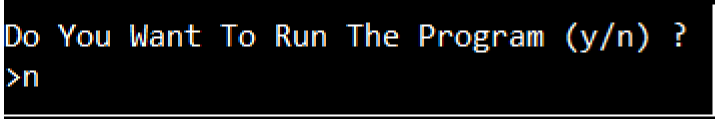


```
Enter an arithmetic expression in infix notation
>A*(B+C*D)+E
```

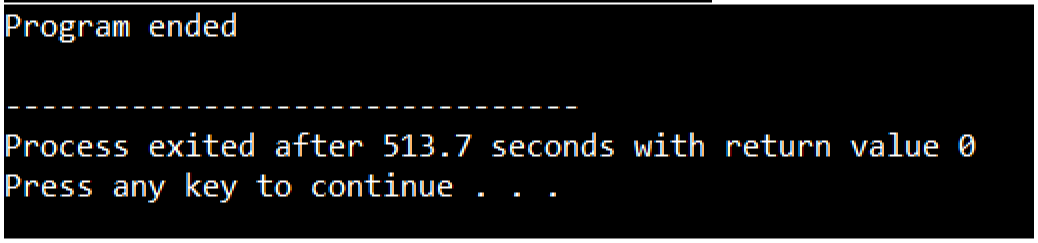

```
Token from InFix: A
PostFix string:
Operator storage: N
Add A to Postfix string

Token from InFix: *
PostFix string: A
Operator storage: N
Push * to storage

Token from InFix: (
PostFix string: A
Operator storage: N *
Push ( to storage

Token from InFix: B
PostFix string: A
Operator storage: N * (
Add B to Postfix string

Token from InFix: +
PostFix string: AB
Operator storage: N * (
Push + to storage

Token from InFix: C
PostFix string: AB
Operator storage: N * ( +
Add C to Postfix string

Token from InFix: *
PostFix string: ABC
Operator storage: N * ( +
Push * to storage
```



```
Token from InFix: D
PostFix string: ABC
Operator storage: N * ( + *
Add D to Postfix string

Token from InFix: )
PostFix string: ABCD
Operator storage: N * ( + *
Pop * to PostFix string

Pop + to PostFix string

Pop ( from storage

Token from InFix: +
PostFix string: ABCD*+
Operator storage: N *
Push + to storage

Token from InFix: E
PostFix string: ABCD*+*
Operator storage: N +
Add E to Postfix string

in InFix notation: A*(B+C*D)+E
in PostFix notation: ABCD*+*E+

Do You Want To Run The Program (y/n) ?
>
```



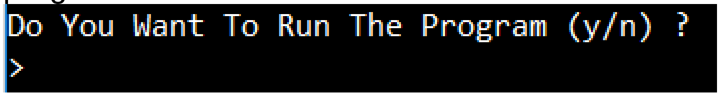
```
Do You Want To Run The Program (y/n) ?
>n
```



```
Program ended


--------------------------------
Process exited after 513.7 seconds with return value 0
Press any key to continue . . .
```

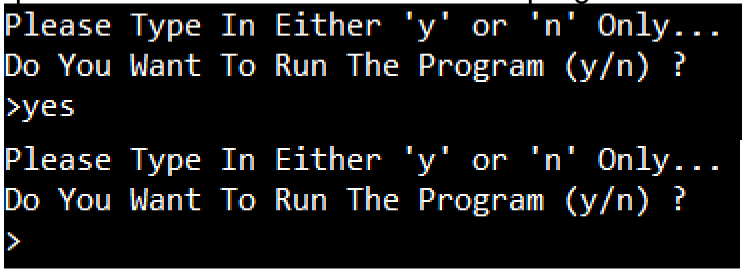
The User Manual

When we run the program, the user will be asked a question whether we want to run the program or not.



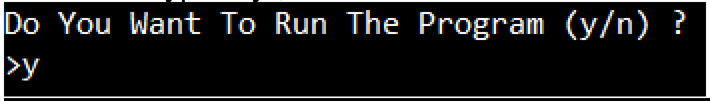
```
Do You Want To Run The Program (y/n) ?
>
```

The user must only answer “y” or “n”, otherwise the program will keep on asking a question whether we want to run the program or not.

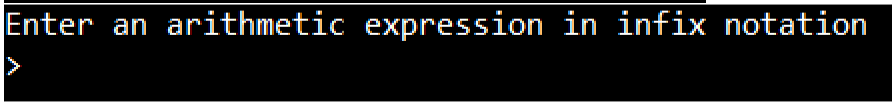


```
Please Type In Either 'y' or 'n' Only...
Do You Want To Run The Program (y/n) ?
>yes
Please Type In Either 'y' or 'n' Only...
Do You Want To Run The Program (y/n) ?
>
```

If the user types “y”, the user will be asked to enter an arithmetic expression.



```
Do You Want To Run The Program (y/n) ?
>y
```



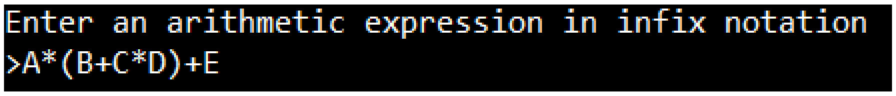
```
Enter an arithmetic expression in infix notation
>
```

Here are some examples of an infix expressions:

1. A+B
2. A+B*C
3. (A+B)*C
4. A+B*C+D
5. (A+B)*(C+D)
6. A*B+C*D
7. A+B+C+D

And, these are the postfix expressions:

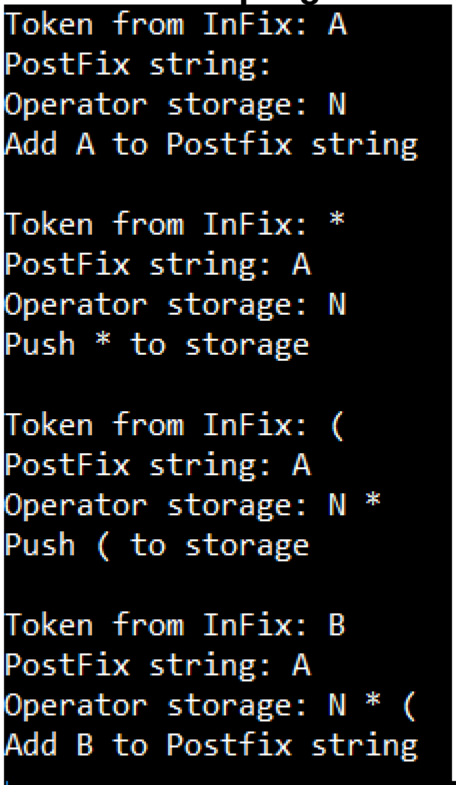
1. AB+
2. ABC*+
3. AB+C*
4. ABC*+D+
5. AB+CD+*
6. AB*CD*+
7. AB+C+D+



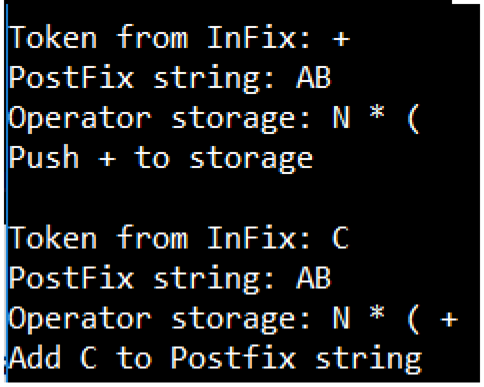
```
Enter an arithmetic expression in infix notation
>A*(B+C*D)+E
```

Postfix: A*(B+C*D)+E
Infix   : ABCD*+*E+
A * (B + C * D) + E becomes A B C D * + * E

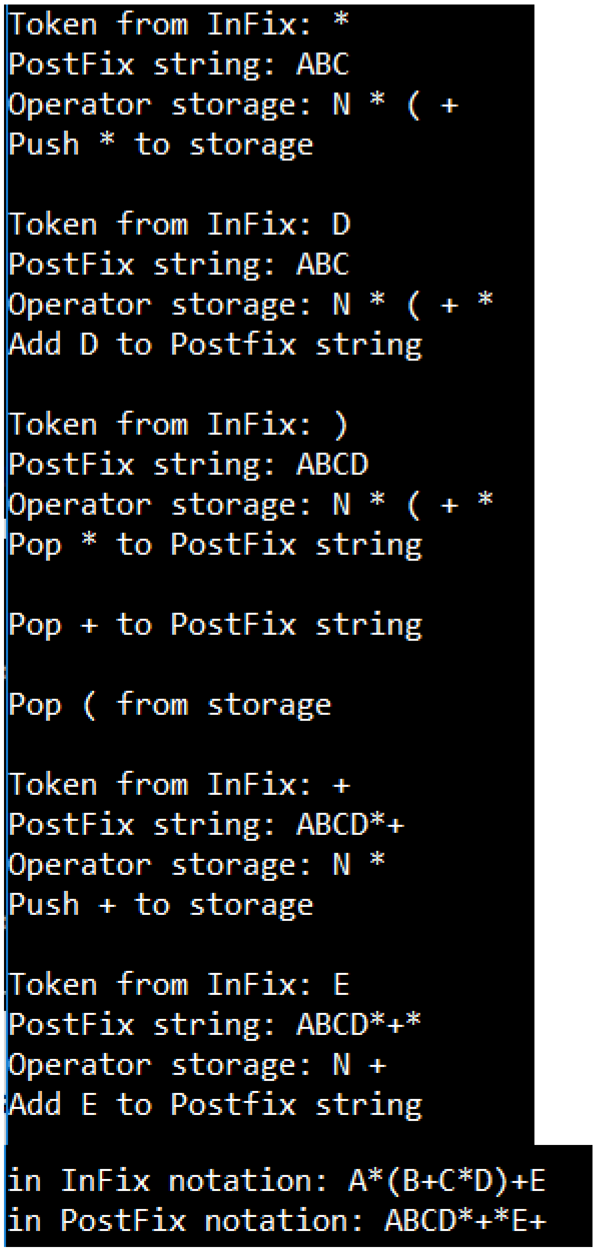
What did the program do?



```
Token from InFix: A
PostFix string:
Operator storage: N
Add A to Postfix string

Token from InFix: *
PostFix string: A
Operator storage: N
Push * to storage

Token from InFix: (
PostFix string: A
Operator storage: N *
Push ( to storage

Token from InFix: B
PostFix string: A
Operator storage: N * (
Add B to Postfix string
```

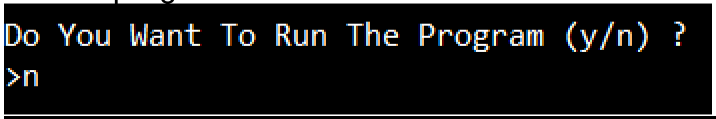


```
Token from InFix: +
PostFix string: AB
Operator storage: N * (
Push + to storage

Token from InFix: C
PostFix string: AB
Operator storage: N * ( +
Add C to Postfix string
```

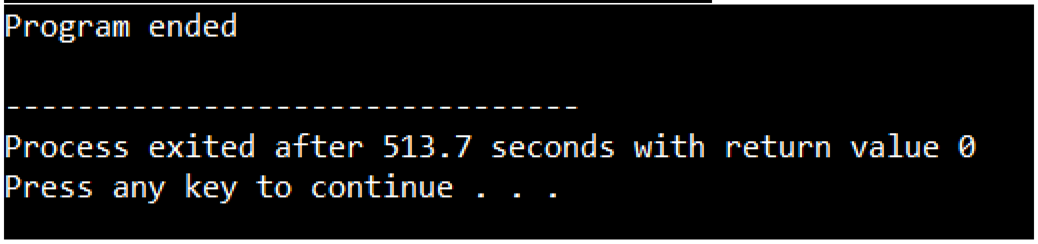


```
Token from InFix: *
PostFix string: ABC
Operator storage: N * ( +
Push * to storage

Token from InFix: D
PostFix string: ABC
Operator storage: N * ( + *
Add D to Postfix string

Token from InFix: )
PostFix string: ABCD
Operator storage: N * ( + *
Pop * to PostFix string

Pop + to PostFix string

Pop ( from storage

Token from InFix: +
PostFix string: ABCD*+
Operator storage: N *
Push + to storage

Token from InFix: E
PostFix string: ABCD*+*
Operator storage: N +
Add E to Postfix string

in InFix notation: A*(B+C*D)+E
in PostFix notation: ABCD*+*E+
```

After the result was being shown, the user will be asked a question whether we want to run the program or not.



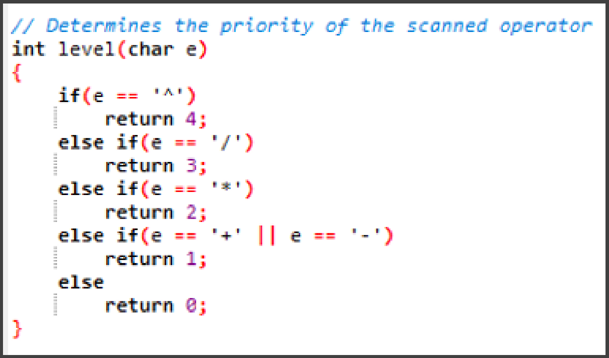
```
Do You Want To Run The Program (y/n) ?
>n
```



```
Program ended

--------------------------------
Process exited after 513.7 seconds with return value 0
Press any key to continue . . .
```

If the user types “n”, the program stops.

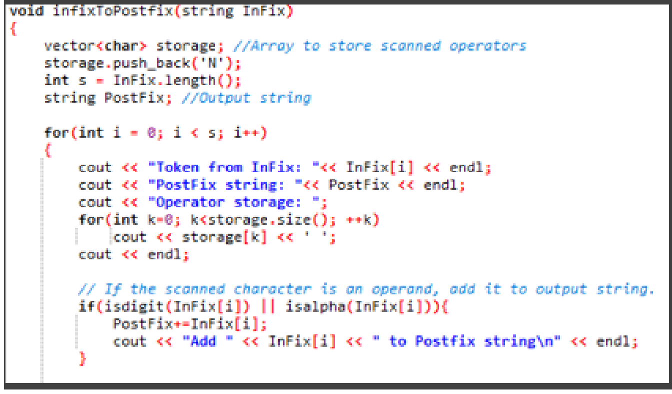
CODE OF THE PROGRAM



```cpp
// Determines the priority of the scanned operator
int level(char e)
{
    if(e == '^')
        return 4;
    else if(e == '/')
        return 3;
    else if(e == '*')
        return 2;
    else if(e == '+' || e == '-')
        return 1;
    else
        return 0;
}
```

Function to determines the precedence or priority of currently processed element (char e).

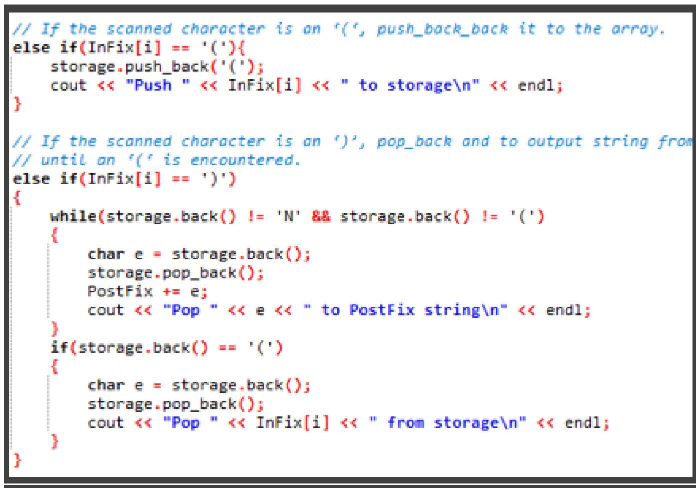
The operand returns the lowest value while ‘power’ returns the highest.



```cpp
void infixToPostfix(string InFix)
{
    vector<char> storage; //Array to store scanned operators
    storage.push_back('N');
    int s = InFix.length();
    string PostFix; //Output string

    for(int i = 0; i < s; i++)
    {
        cout << "Token from InFix: "<< InFix[i] << endl;
        cout << "PostFix string: "<< PostFix << endl;
        cout << "Operator storage: ";
        for(int k=0; k<storage.size(); ++k)
            cout << storage[k] << ' ';
        cout << endl;

        // If the scanned character is an operand, add it to output string.
        if(isdigit(InFix[i]) || isalpha(InFix[i])){
            PostFix+=InFix[i];
            cout << "Add " << InFix[i] << " to Postfix string\n" << endl;
        }
```

Function to convert Infix to Postfix, each element of the input will be scanned.

If the function scanned is an operand, it will be stored in the final Postfix output.

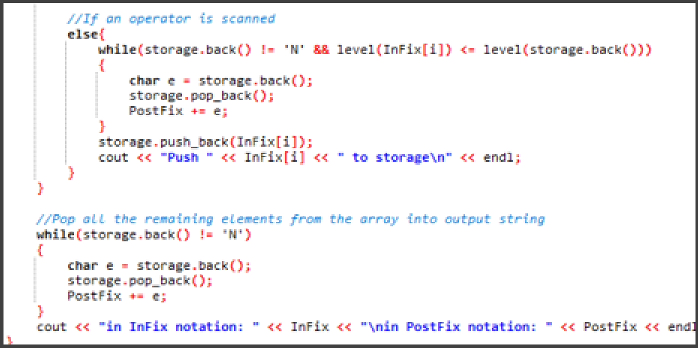


```
// If the scanned character is an '(', push_back_back it to the array.
else if(InFix[i] == '('){
    storage.push_back('(');
    cout << "Push " << InFix[i] << " to storage\n" << endl;
}

// If the scanned character is an ')', pop_back and to output string from
// until an '(' is encountered.
else if(InFix[i] == ')')
{
    while(storage.back() != 'N' && storage.back() != '(')
    {
        char e = storage.back();
        storage.pop_back();
        PostFix += e;
        cout << "Pop " << e << " to PostFix string\n" << endl;
    }
    if(storage.back() == '(')
    {
        char e = storage.back();
        storage.pop_back();
        cout << "Pop " << InFix[i] << " from storage\n" << endl;
    }
}
```

Part of the convert function that focuses on bracket input.

If open bracket is scanned, it will be pushed into the vector.

In case close bracket is scanned, If last element in vector is not an open bracket, the function will pop the last element into Postfix output. While last is open bracket, last element in vector will be popped.

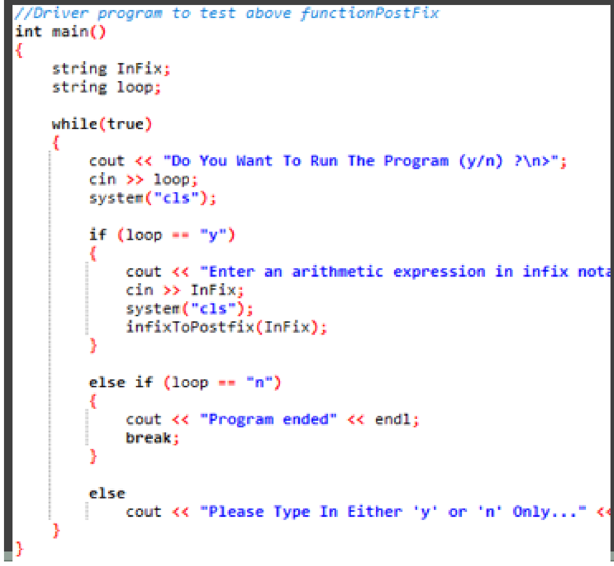


```
        //If an operator is scanned
        else{
            while(storage.back() != 'N' && level(InFix[i]) <= level(storage.back()))
            {
                char e = storage.back();
                storage.pop_back();
                PostFix += e;
            }
            storage.push_back(InFix[i]);
            cout << "Push " << InFix[i] << " to storage\n" << endl;
        }
    }

    //Pop all the remaining elements from the array into output string
    while(storage.back() != 'N')
    {
        char e = storage.back();
        storage.pop_back();
        PostFix += e;
    }
    cout << "in InFix notation: " << InFix << "\nin PostFix notation: " << PostFix << endl
}
```

Else if operator is scanned, last element of storage will be popped and pushed into String output.

And current element will be pushed into storage vector.

And if there are still remaining data in storage vector, all will be popped and pushed into Postfix output.



```
//Driver program to test above functionPostFix
int main()
{
    string InFix;
    string loop;

    while(true)
    {
        cout << "Do You Want To Run The Program (y/n) ?\n>";
        cin >> loop;
        system("cls");

        if (loop == "y")
        {
            cout << "Enter an arithmetic expression in infix nota
            cin >> InFix;
            system("cls");
            infixToPostfix(InFix);
        }

        else if (loop == "n")
        {
            cout << "Program ended" << endl;
            break;
        }

        else
            cout << "Please Type In Either 'y' or 'n' Only..." <<
    }
}
```

The main function of the program, controls the usage of each function and controls the flow of the main program.

Link to the application demo video (with max. length of 2 minutes)
https://youtu.be/F8JvoSxOiBk

Link to the GIT website:
https://github.com/fiqhyb/COMP_6048-FINAL_PROJECT