Data Structures 2012-2013 Fall

Practice Session 3

Contents

- Infix to Postfix Conversion

Infix vs Postfix

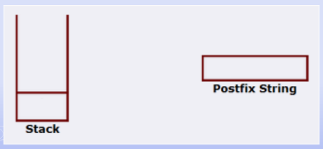
- **Infix Expression:** Any expression in the standard form like "2*3-4/5" is an Infix(Inorder) expression.
- **Postfix Expression:** The Postfix(Postorder) form of the above expression is "23*45/-". In a Postfix expression, operators are written after operands.

Infix to Postfix Conversion by Using a Stack

- Initialize an empty stack.
- Scan the Infix string from left to right. For all the characters,
  - If the scanned character is an **operand**, add it to the Postfix string.
  - If the scanned character is an **operator** and
    - If the stack is empty or the character is '(', ***push*** the character to the stack.
    - If the stack is not empty, compare the precedence of the character on top of the stack(topStack). If topStack has higher precedence over the scanned character ***pop*** it from the stack to the Postfix string else ***push*** the scanned character to stack. Repeat this step as long as the stack is not empty and topStack has precedence over the character.
    - If the character is ')' ***pop*** all the characters to the Postfix string until reaching '(' in the stack.
- After all the characters are scanned, add any remaining character in the stack to the Postfix string.

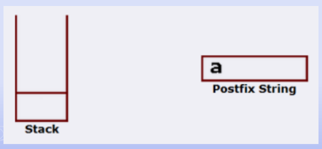
Example

- **Infix String:** a+b*(c-d)
- Initially the Stack is empty and Postfix string has no characters.



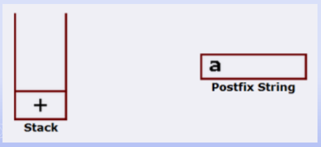
Example

- **Infix String:** a+b*(c-d)
- The first character scanned is 'a'. As 'a' is an operand, it is added to the Postfix string.



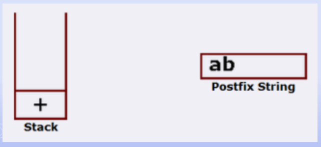
Example

- **Infix String:** a+b*(c-d)
- The next character scanned is '+'. As it is an operator and the stack is empty, it is pushed to the stack.



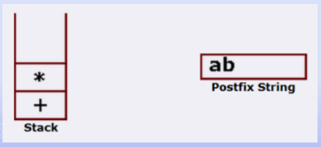
Example

- **Infix String:** a+b*(c-d)
- The next character scanned is 'b' which is placed in the Postfix string as it is an operand.



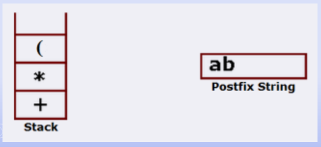
Example

- **Infix String:** a+b*(c-d)
- The next character scanned is '*'. As it is an operator and it has more precedence than '+' on the top of the stack , it is pushed to the stack.



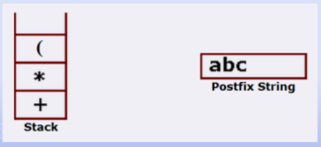
Example

- **Infix String:** a+b*(c-d)
- The next character scanned is '(' and it is directly pushed to the stack.



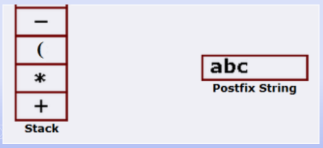
Example

- **Infix String:** a+b*(c-d)
- The next character scanned is 'c' which is placed in the Postfix string as it is an operand.



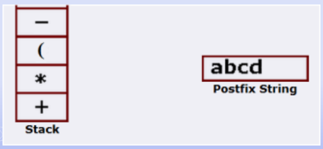
Example

- **Infix String:** a+b*(c-d)
- The next character scanned is '-'. As '(' does not have any precedence problem, scanned '-' is pushed in the stack.



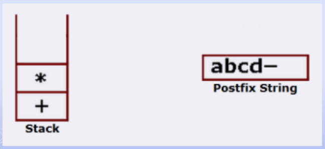
Example

- **Infix String:** a+b*(c-d)
- The next character scanned is 'd' which is placed in the Postfix string as it is an operand.



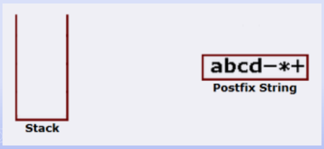
Example

- **Infix String:** a+b*(c-d)
- The last character of the infix string is ')'. All the characters (i.e. only '-' here) are popped into postfix string until reaching '('.
- Then '(' is popped without appending to postfix string.



Example

- **Infix String:** a+b*(c-d)
- As the infix string is scanned to the end, the remaining characters in the stack are appended to the end of the postfix string.



Stack Structure

```
#ifndef STACK_H
#define STACK_H

struct Node{
    char data;
    Node *next;
};

struct Stack{
    Node *head;
    void create();
    void makeEmpty();
    void push(char);
    char pop();
    // to read the top element without popping
    char getHeadChar();
    bool isEmpty();
    // to print the elements in the stack
    void print();
};

#endif
```

Stack Methods

```
void Stack::create(){
    head = NULL;
}
```

```
bool Stack::isEmpty(){
    return head==NULL;
}
```

```
void Stack::makeEmpty(){
    Node *p;
    while (head){
        p = head;
        head = head->next
        delete p;
    }
}
```

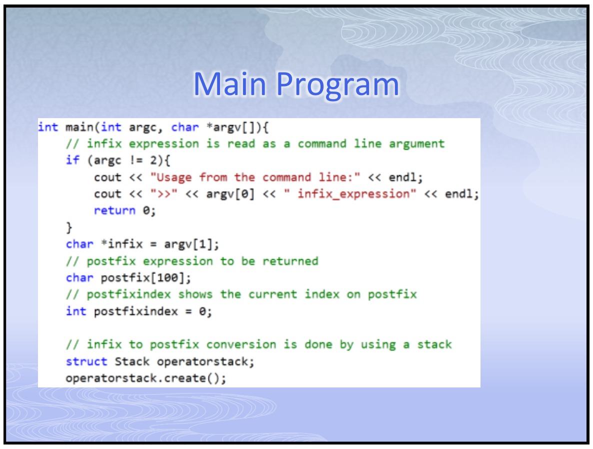
```
void Stack::print(){
    cout<<"Stack contents : ";
    Node *p = head;
    while (p){
        cout<<p->data<<" ";
        p = p->next;
    }
    cout<<endl;
}
```

Stack Methods

```
void Stack::push(char toPush){
    Node *newNode = new Node;
    newNode->data = toPush;
    newNode->next = head;
    head = newNode;
}
```

```
char Stack::getHeadChar(){
    if (head == NULL)
        return '\0';
    return head->data;
}
```

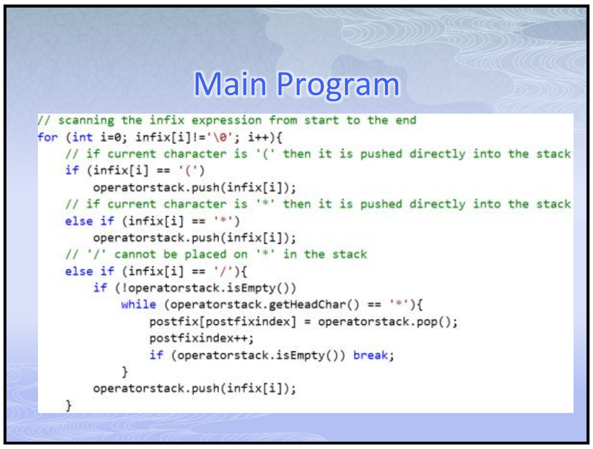
```
char Stack::pop(){
    if (head == NULL)
        return '\0';
    Node *topStack;
    char toReturn;
    topStack = head;
    head = head->next;
    toReturn = topStack->data;
    delete topStack;
    return toReturn;
}
```



Main Program

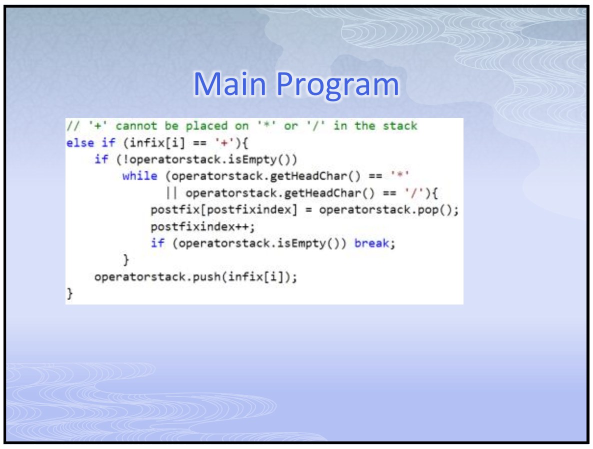
```
int main(int argc, char *argv[]){
    // infix expression is read as a command line argument
    if (argc != 2){
        cout << "Usage from the command line:" << endl;
        cout << ">>" << argv[0] << " infix_expression" << endl;
        return 0;
    }
    char *infix = argv[1];
    // postfix expression to be returned
    char postfix[100];
    // postfixindex shows the current index on postfix
    int postfixindex = 0;

    // infix to postfix conversion is done by using a stack
    struct Stack operatorstack;
    operatorstack.create();
```



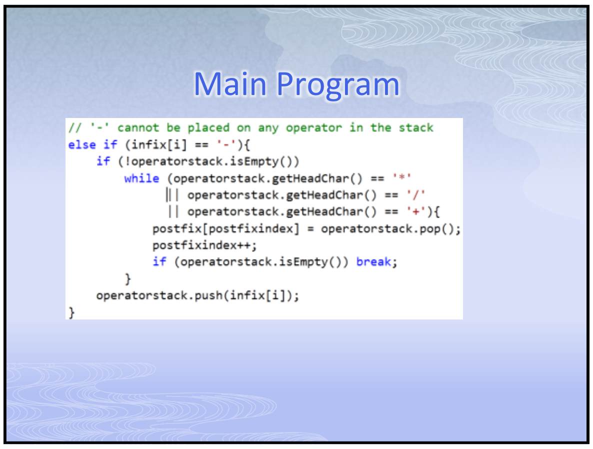
Main Program

```
// scanning the infix expression from start to the end
for (int i=0; infix[i]!='\0'; i++){
    // if current character is '(' then it is pushed directly into the stack
    if (infix[i] == '(')
        operatorstack.push(infix[i]);
    // if current character is '*' then it is pushed directly into the stack
    else if (infix[i] == '*')
        operatorstack.push(infix[i]);
    // '/' cannot be placed on '*' in the stack
    else if (infix[i] == '/'){
        if (!operatorstack.isEmpty())
            while (operatorstack.getHeadChar() == '*'){
                postfix[postfixindex] = operatorstack.pop();
                postfixindex++;
                if (operatorstack.isEmpty()) break;
            }
        operatorstack.push(infix[i]);
    }
```



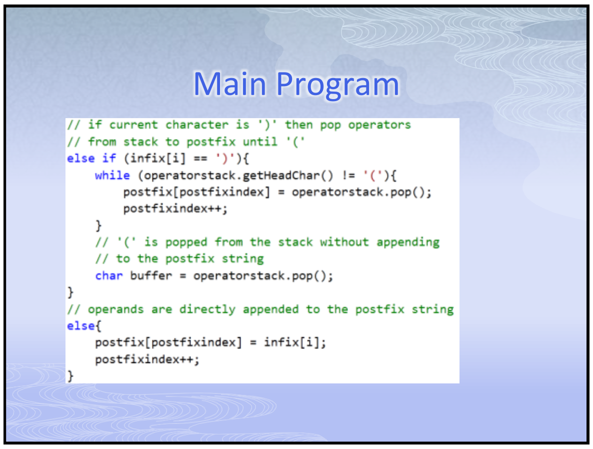
Main Program

```
// '+' cannot be placed on '*' or '/' in the stack
else if (infix[i] == '+'){
    if (!operatorstack.isEmpty())
        while (operatorstack.getHeadChar() == '*'
              || operatorstack.getHeadChar() == '/'){
            postfix[postfixindex] = operatorstack.pop();
            postfixindex++;
            if (operatorstack.isEmpty()) break;
        }
    operatorstack.push(infix[i]);
}
```



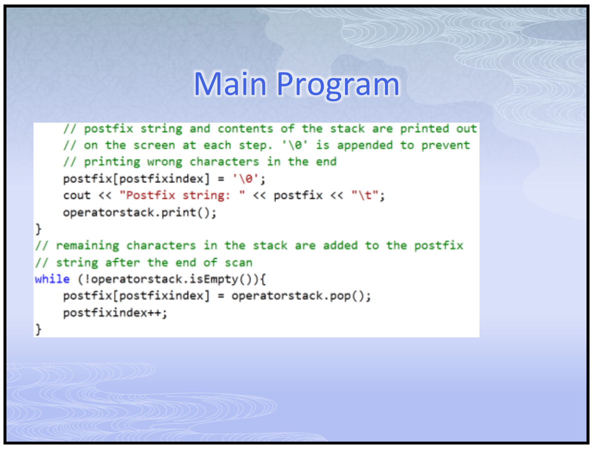
Main Program

```
// '-' cannot be placed on any operator in the stack
else if (infix[i] == '-'){
    if (!operatorstack.isEmpty())
        while (operatorstack.getHeadChar() == '*'
              || operatorstack.getHeadChar() == '/'
              || operatorstack.getHeadChar() == '+'){
            postfix[postfixindex] = operatorstack.pop();
            postfixindex++;
            if (operatorstack.isEmpty()) break;
        }
    operatorstack.push(infix[i]);
}
```



Main Program

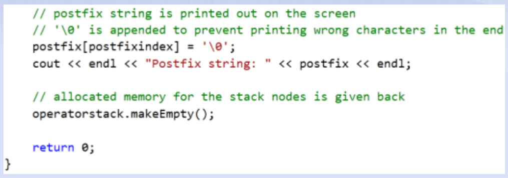
```
// if current character is ')' then pop operators
// from stack to postfix until '('
else if (infix[i] == ')'){
    while (operatorstack.getHeadChar() != '('){
        postfix[postfixindex] = operatorstack.pop();
        postfixindex++;
    }
    // '(' is popped from the stack without appending
    // to the postfix string
    char buffer = operatorstack.pop();
}
// operands are directly appended to the postfix string
else{
    postfix[postfixindex] = infix[i];
    postfixindex++;
}
```



Main Program

```
    // postfix string and contents of the stack are printed out
    // on the screen at each step. '\0' is appended to prevent
    // printing wrong characters in the end
    postfix[postfixindex] = '\0';
    cout << "Postfix string: " << postfix << "\t";
    operatorstack.print();
}
// remaining characters in the stack are added to the postfix
// string after the end of scan
while (!operatorstack.isEmpty()){
    postfix[postfixindex] = operatorstack.pop();
    postfixindex++;
}
```

Main Program

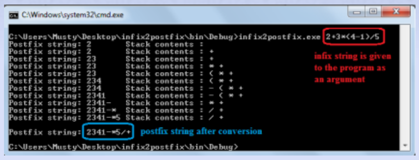


```
    // postfix string is printed out on the screen
    // '\0' is appended to prevent printing wrong characters in the end
    postfix[postfixindex] = '\0';
    cout << endl << "Postfix string: " << postfix << endl;

    // allocated memory for the stack nodes is given back
    operatorstack.makeEmpty();

    return 0;
}
```

Screenshot of the Program



C:\Windows\system32\cmd.exe
infix string is given to the program as an argument
postfix string after conversion

```
C:\Users\Musty\Desktop\infix2postfix\bin\Debug>infix2postfix.exe 2+3*(4-1)/5
Postfix string: 2       Stack contents :
Postfix string: 2       Stack contents : +
Postfix string: 23      Stack contents : +
Postfix string: 23      Stack contents : * +
Postfix string: 23      Stack contents : ( * +
Postfix string: 234     Stack contents : ( * +
Postfix string: 234     Stack contents : - ( * +
Postfix string: 2341    Stack contents : - ( * +
Postfix string: 2341-   Stack contents : * +
Postfix string: 2341-*  Stack contents : / +
Postfix string: 2341-*5 Stack contents : / +

Postfix string: 2341-*5/+

C:\Users\Musty\Desktop\infix2postfix\bin\Debug>
```

References

- http://scriptasylum.com/tutorials/infix_postfix/algorithms/infix-postfix/index.htm