4.2

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXSIZE 130
struct node {
 char * value;
 struct node * next;
};
void push(struct node ** top,char * val){
 struct node * n = (struct node
*)malloc(sizeof(struct node));
 if (n){
  n->value = (char
*)malloc(sizeof(char)*MAXSIZE);
  strcpy(n->value, val);
  n->next = *top;
  *top = n;
 }
}
struct node * pop (struct node **top){
 struct node * t = *top;
 *top = (*top)->next;
 t->next = NULL;
 return t;
}
void display(struct node ** start){
 struct node * p;
 printf("[  ");
 for (p = *start;p!=NULL;p = p->next)
  printf("%s  ",p->value);
 printf(" ]\n");
}
int isWhiteSpace(char c ){
 if (c==' ' || c=='\t'||c=='\n')
  return 1;
 return 0;
}
 char * pfToInfix(char * input){
 struct node * Stack = NULL;
 struct node * w, * t, * y;
 int nStack = 0,i;
 char p,q;
 char * s1, *s2, *s4;
 for (i = 0;i < strlen(input);i++){
  p = input[i];
  if (!isWhiteSpace(p)){
   switch(p){
    case '+':
    case '-':
    case '*':
    case '/':
    case '^':
    case '%':
     //if more than 2 values on stack
     if (nStack<2){
      printf("Error\n");
     }else{
      w = pop(&Stack);
      t = pop(&Stack);
      //operands nodes in w and t
      s1 = (char
*)malloc(sizeof(char)*MAXSIZE);
      s2 = (char
*)malloc(sizeof(char)*MAXSIZE);
      s4 = (char
*)malloc(sizeof(char)*MAXSIZE);
      strcpy(s1,w->value);
      strcpy(s2,t->value);
      //now operand strings in s1, s2
      char s3[4];
      s3[0] = p;
      s3[1] = '\0';
      strcpy(s4,"(");
      strcat(s4,s2);
      strcat(s4,s3);
      strcat(s4, s1);
      strcat(s4,")");
      push(&Stack,s4);
      free(s1);
      free(s2);
      free(s4);
```

```
        free(w->value);
        free(t->value);
        free(w);
        free(t);
        nStack--;
       }
       break;
     default:
      //operands shud be pushed to stack
      char s5[3];
      s5[0] = p;
      s5[1] = '\0';
      push(&Stack,s5);
      nStack++;
    }
   }
  }
  char * exp = (char
*)malloc(sizeof(char)*MAXSIZE);
  if (nStack == 1){
   y = pop(&Stack);
   strcpy(exp,y->value);
   free(y->value);
   free(y);
  }else{
   strcpy(exp,"\nThe input expression is
invalid\n");
  }
  return exp;
}
int main(){
  char * fExp,*input;
  int m;
  input = (char *)malloc(sizeof(char)*
MAXSIZE);
  printf("------Postfix to infix convertor------");
  do {
   printf("\nEnter\n\t1 to convert
expression\n\t2 to exit:");
   scanf("%d",&m);
   switch(m){
    case 1:
     getchar();
     printf("Enter posfix expression to be
evaluated: \n");
     fgets(input,MAXSIZE,stdin);
     input[strlen(input)-1] = '\0';
     fExp = pfToInfix(input);
     printf("The evaluated expression
is:%s\n",fExp);
     break;
    case 2:
     break;
    default:
     printf("Invalid input :(1/2 only)\n");
   }
  }while(m!=2);
  return 0;
}
```

Output:

```
C:\Users\Vincent\Code\c-cpp\SE\Labs\DS\LabS\exp4>.\stack.exe
------Postfix to infix convertor------
Enter
        1 to convert expression
        2 to exit:1
Enter posfix expression to be evaluated:
93-4+3*
The evaluated expression is:(((9-3)+4)*3)

Enter
        1 to convert expression
        2 to exit:1
Enter posfix expression to be evaluated:
3-2+4
Error
The evaluated expression is:
The input expression is invalid
Unable to generate needed output


Enter
        1 to convert expression
        2 to exit:3
Invalid input (1/2 only)

Enter
        1 to convert expression
        2 to exit:2
```