Problem 1.

CODE

```
#include<stdio.h>

#include<string.h>

#define SIZE 25

int top = -1, low = 0;

int stack[SIZE];

void push(char c);

char pop();

int isEmpty(int stack[]);

int isFull(int stack[]);

char p(char c);

void display();

void checkExp (char exp[]);

int main()

{

    char exp[SIZE];

    int choice=0;


    while(1)

    {

        printf("Please Enter 1 to input or 2 to exit : ");

        scanf("%d",&choice);

        fflush(stdin);
        switch(choice)
        {
        case 1:

            printf("Enter the Expression : ");
            gets(exp);
            checkExp (exp);
            display();
            printf("\n");
            break;
        case 2:

            exit(0);
            break;
        default:
            printf("Invalid Input\n");
        }
    }

    return 0;
}

void push(char c){
    if(isFull(stack))

    {

        printf("Stack overflow");

    }

    else

    {

        stack[++top] = c;

    }


}


char pop(){



    if(isEmpty(stack))

    {

        printf("Stack is Empty");

    }

    else

    {

         char e = stack[top];

         --top;

         printf("%c",e);
    }

}

int isEmpty(int stack[])
{
    return (top == -1);
}

int isFull(int stack[])
{
    return (top == SIZE - 1);
}

void display()
{
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        return;
    }
    else {
        int count=0;
        char x;
        while (!isEmpty(stack)) {
```

```
x=pop();
 if (x == '(' || x== '[' || x== '{'){
        count++;
        if ( count == 1){
        printf("%c\n\t",x);
        }
        if (count == 2){
                                    printf("%c\n\t\t",x);
                                    }
                                    if (count == 3){
                                    printf("%c\n\t\t\t",x);
                                    }
 }
                  if (x == ')' || x== ']' || x== '}'){
        count--;
    if ( count == 1){
        printf("%c\n\t",x);
    }
    else if (count == 2){
        printf("%c\n\t\t",x);
    }
    else if (count == 3){
        printf("%c\n\t\t\t",x);
    }
```

```
            }
        }

        }
}
char p(char c)
{
    if (c == ')')
        return '(';
    else if (c == '}')
        return '{';
    else
        return '[';
}

void checkExp (char exp[])
{
    int i,j,count=0;
    int cl=0, cr=0;
    char ch, x;
    for (i = 0; i < strlen(exp); i++) {
        if (exp[i] == '(' || exp[i] == '[' || exp[i] == '{') {
            push(exp[i]);
            cl++;
    }
    else if (exp[i] == ')' || exp[i] == '}' || exp[i] == ']') {
          cr++;
        if (isEmpty(stack)) {
            printf("NO OPENING BRACKET...INVALID EXPRESSION\n");
            return -1;
        }
        else {
            x = p(exp[i]);
            if (pop() != x) {
                printf("NO MATCHING..INVALID EXPRESSION\n");
                return -1;
            }
        }
    }
}
if (cl>cr) {
    printf("LEFT PARENTHESIS ARE >THAN RIGHT.!!INVALID EXPRESSION\n");
}
else{
       printf("\n%s expression contains %d Matching Groups\n \n",exp,cl);
       for (j=strlen(exp) ; j > -1; j--){
                 push(exp[j]);
                 }
        }
}
```

Problem 2.

CODE

```
#include<stdio.h>
#include<stdlib.h>

#define SIZE 100

void push(char );
char pop();
int is_operator(char );
int precedence(char );
void InfixToPostfix(char [], char []);

char stack[SIZE];
int top = -1;

int main()
{
        char infix[SIZE], postfix[SIZE];

        printf("ASSUMPTION: The infix expression contains single letter variables
and single digit constants only.\n");
```

```
	printf("\nEnter Infix expression : ");
	gets(infix);

	InfixToPostfix(infix,postfix);
	printf("Postfix Expression: ");
	puts(postfix);

	return 0;
}

void push(char val)
{
	if(top >= SIZE-1)
	{
		printf("\nStack Overflow...!!!");
	}
	else
	{
		top = top+1;
		stack[top] = val;
	}
}

char pop()
```

```
{
        char item ;

        if(top <0)
        {
                printf("Stack Underflow...!!!");
                getchar();
                exit(1);
        }
        else
        {
                item = stack[top];
                top = top-1;
                return(item);
        }
}

int is_operator(char op)
{
        if(op == '^' || op == '*' || op == '/' || op == '+' || op =='-')
        {
                return 1;
        }
        else
```

```
          {
          return 0;
          }
}

int precedence(char symbol)
{
          if(symbol == '^')
          {
                    return(3);
          }
          else if(symbol == '*' || symbol == '/')
          {
                    return(2);
          }
          else if(symbol == '+' || symbol == '-')
          {
                    return(1);
          }
          else
          {
                    return(0);
          }
}
```

```
void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
        int i, j;
        char item;
        char x;

        push('(');
        strcat(infix_exp,")");

        i=0;
        j=0;
        item=infix_exp[i];

        while(item != '\0')
        {

                if(item == '(')
                {
                        push(item);
                }
                else if( isdigit(item) || isalpha(item))
                {
                        postfix_exp[j] = item;
```

```
            j++;
        }
        else if(is_operator(item) == 1)
        {
            x=pop();
            while(is_operator(x) == 1 && precedence(x)>=
precedence(item))
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
            push(x);

            push(item);
        }
        else if(item == ')')
        {
            x = pop();
            while(x != '(')
            {
                postfix_exp[j] = x;
                j++;
                x = pop();
            }
        }
        else
        {
                printf("\nInvalid infix Expression.\n");
                getchar();
                exit(1);
        }
        i++;


        item = infix_exp[i];
}
if(top>0)
{
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
}
if(top>0)
{
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
}
```

```
        postfix_exp[j] = '\0';

    }
```