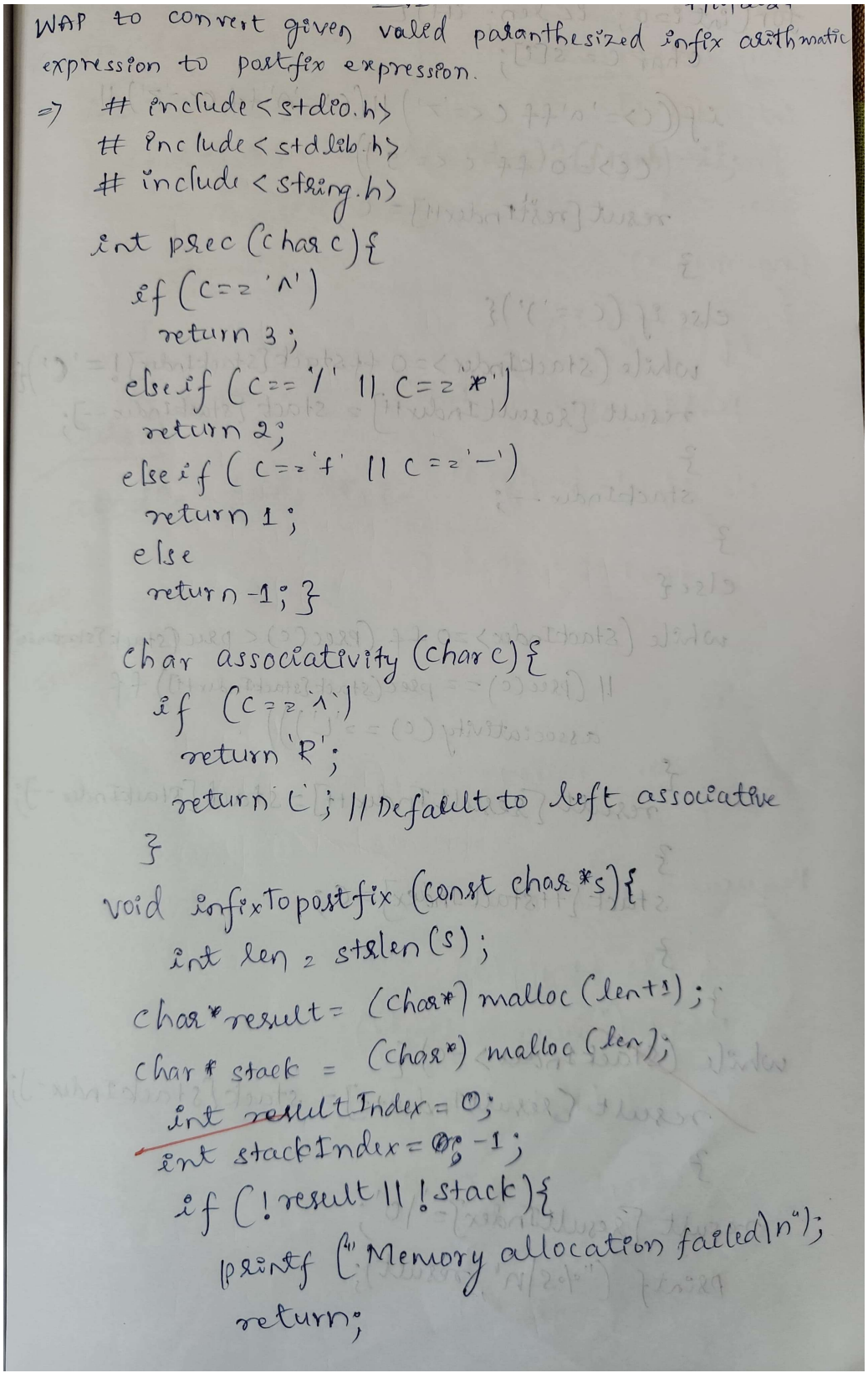
Program 2

Write a program to convert a given valid parenthesized in-fix arithmetic expression to post-fix expression.The expression consists of single character operands and the binary operators +,-,*,/.

Observation:



```
WAP to convert given valid paranthesized infix arithmatic
expression to postfix expression.
=> #include<stdio.h>
   #include<stdlib.h>
   #include<string.h>
   int prec(char c){
     if(c=='^')
       return 3;
     else if(c=='/'||c=='*')
       return 2;
     else if(c=='+'||c=='-')
       return 1;
     else
       return -1;}
   char associativity(char c){
     if(c=='^')
       return 'R';
       return 'L'; // Default to left associative
     }
   void infixtopostfix(const char *s){
       int len = strlen(s);
     char*result=(char*)malloc(len+1);
     char* stack = (char*) malloc(len);
       int resultIndex=0;
       int stackIndex=-1;
       if(!result||!stack){
         printf("Memory allocation failed\n");
         return;
```

```
for(int i=0; i<len; i++){
    char c = s[i];
    if((c>='a' && c<='z') || (c>='A' && c<='Z') ||
       (c>='0' && c<='9'))
        result[resultIndex++] = c;
    }
    else if (c==')'){
    while (stackIndex>=0 && stack[stackIndex]!='('){
    result [resultIndex++] = stack [stackIndex--];
    }
    stackIndex--;
}
else{
while (stackIndex>=0 && (prec(c) < prec(stack[stackIndex]
        || (prec(c)==prec(stack[stackIndex+1]) &&
        associativity(c)=='L')))
    {
    result [resultIndex++] = stack[stackIndex--];
    }
    stack[++stackIndex]=c;
    }
}
while (stackIndex>=0){
    result [resultIndex++]= stack[stackIndex--];
    }
result [resultIndex]='\0';
printf ("%s\n", result);
```

```
free(result);
free(stack); }
int main() {
    char exp[] = "a+b*(c^d-e)^(f+g*h)-i";
    infixToPostfix(exp);
    return 0;
}
```

Output

```
abcd^e-fgh*+^*+i-
```

Seen
Gl
28/10/24

Code:

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

int prec(char c){
    if (c == '+' || c == '-') {
        return 1;
    }
    if (c == '*' || c == '/') {
        return 2;
    }
    if (c == '^') {
        return 3;
    }
    return 0;
}

char associativity(char c){
    if(c=='^')
        return 'R';
    return 'L';
}

void infixtopostfix(const char *s){
    int len = strlen(s);
    char result[len+1];
    char stack[len];
    int resultIndex=0;
    int StackIndex=-1;

    if(!result || !stack){
        printf("Memory Allocation Failed \n");
        return;
    }

    for (int i = 0; i < len; i++)
    {
        char c=s[i];
        if ((c>='a' && c<='z') || (c>='A' && c<='Z'))
        {
            result[resultIndex++] = c;
        }
        else if(c=='('){
            stack[++StackIndex]=c;
        }
        else if(c==')'){
            while (StackIndex>=0 && stack[StackIndex]!='('
```

```
            {
                result[resultIndex++]=stack[StackIndex--];
            }
           StackIndex--;
        }
        else{
            while (StackIndex>=0 && (prec(c))<prec(stack[StackIndex]) ||
(prec(c)==prec(stack[StackIndex]) && associativity(c)=='L'))
            {
                result[resultIndex++]=stack[StackIndex--];
            }
            stack[++StackIndex]=c;
        }
    }

    while (StackIndex>=0)
    {
        result[resultIndex++]=stack[StackIndex--];
    }
    result[resultIndex++]='\0';
    printf("%s\n",result);
}

int main(){
    char exp[] = "a+b*(c^d-e)^(f+g*h)-i";
    infixtopostfix(exp);
    return 0;
}
```

Output: