

ABHINAV RANJAN  
RA1911003010003  
CSE A1 SECTION  
SRMIST , KTR

## **COMPILER DESIGN LAB**

***EXP 10 - INTERMEDIATE CODE GENERATOR***

## EXP 10 - INTERMEDIATE CODE GENERATOR

AIM

To implement a C program for intermediate code generation.

REQUIREMENTS

1. Knowledge of the concepts of compiler design and intermediate code generation.
2. Online GDB compiler to run and execute code.

THEORY

In the analysis-synthesis model of a compiler, the front end of a compiler translates a source program into an independent intermediate code, then the back end of the compiler uses this intermediate code to generate the target code which can be understood by a machine.

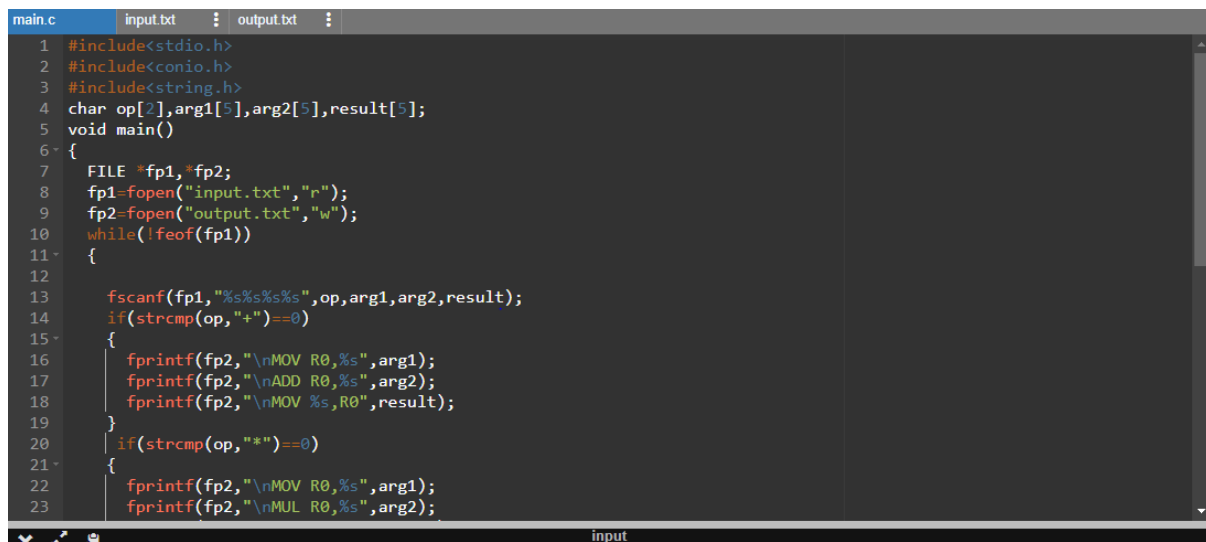
Benefits

- \* enhanced portability of the code
- \* retargeting is facilitated
- \* easier to apply source code modification to improve the performance of source code by optimising the intermediate code.

## ALGORITHM

1. Start
2. Get address code sequence
3. Determine current location of using address [for 1st operand]
4. If current location not already exist generate move (B, 0).
5. Update address of A [for 2nd operand]
6. If current value of B and C) is null, exist.
7. If they generate operator C) A, 3 ADPR.
8. Store the move instruction in memory.
9. Stop.

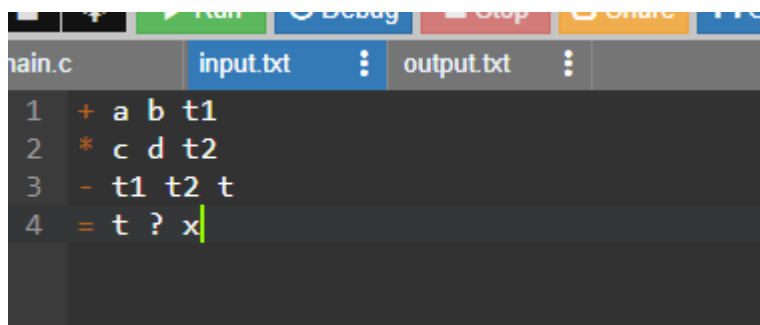
## SCREENSHOTS :



A screenshot of a code editor showing a C program. The editor has tabs for 'main.c', 'input.txt', and 'output.txt'. The code in 'main.c' is as follows:

```
1 #include<stdio.h>
2 #include<conio.h>
3 #include<string.h>
4 char op[2],arg1[5],arg2[5],result[5];
5 void main()
6 {
7     FILE *fp1,*fp2;
8     fp1=fopen("input.txt","r");
9     fp2=fopen("output.txt","w");
10    while(!feof(fp1))
11    {
12
13        fscanf(fp1,"%s%s%s%s",op,arg1,arg2,result);
14        if(strcmp(op,"+")==0)
15        {
16            fprintf(fp2,"\nMOV R0,%s",arg1);
17            fprintf(fp2,"\nADD R0,%s",arg2);
18            fprintf(fp2,"\nMOV %s,R0",result);
19        }
20        if(strcmp(op,"*")==0)
21        {
22            fprintf(fp2,"\nMOV R0,%s",arg1);
23            fprintf(fp2,"\nMUL R0,%s",arg2);
```

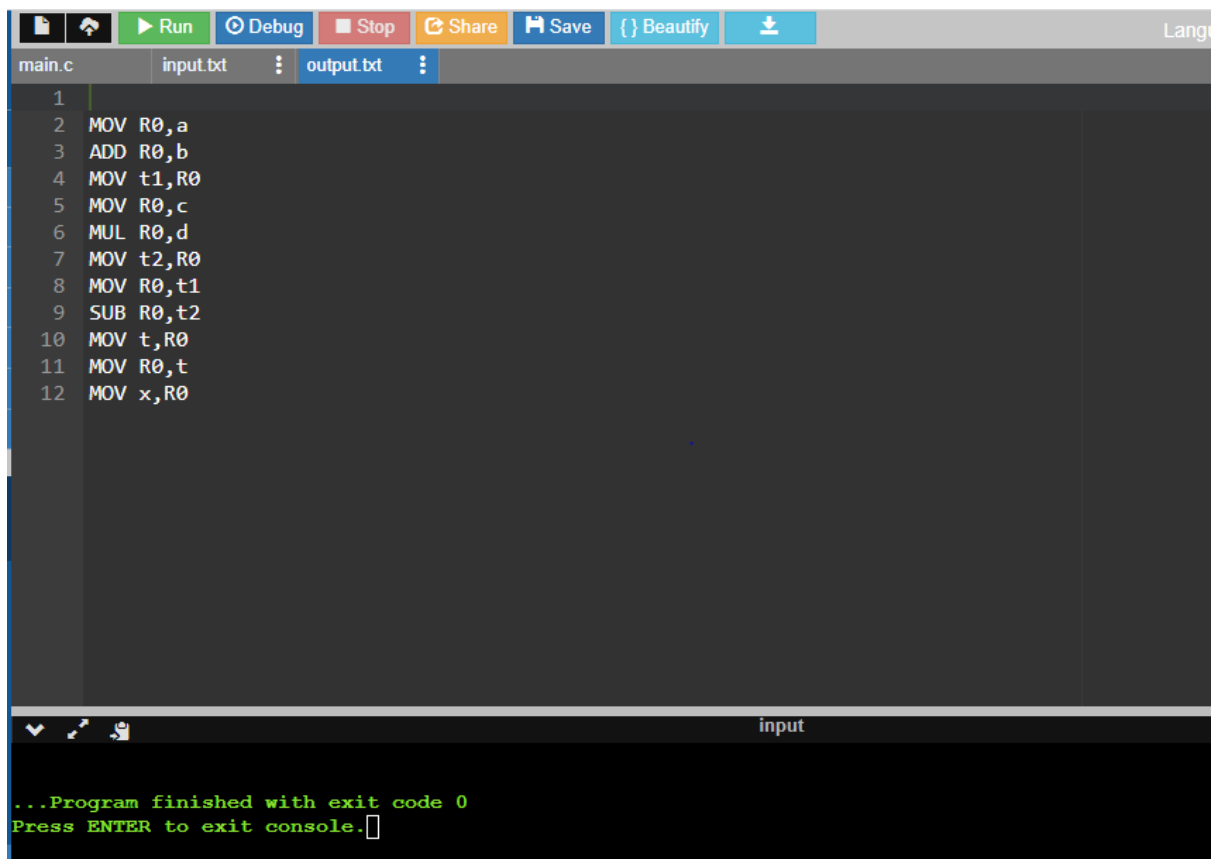
## INPUT FILE :



A screenshot of a code editor showing the content of the 'input.txt' file. The editor has tabs for 'main.c', 'input.txt', and 'output.txt'. The content of 'input.txt' is as follows:

```
1 + a b t1
2 * c d t2
3 - t1 t2 t
4 = t ? x
```

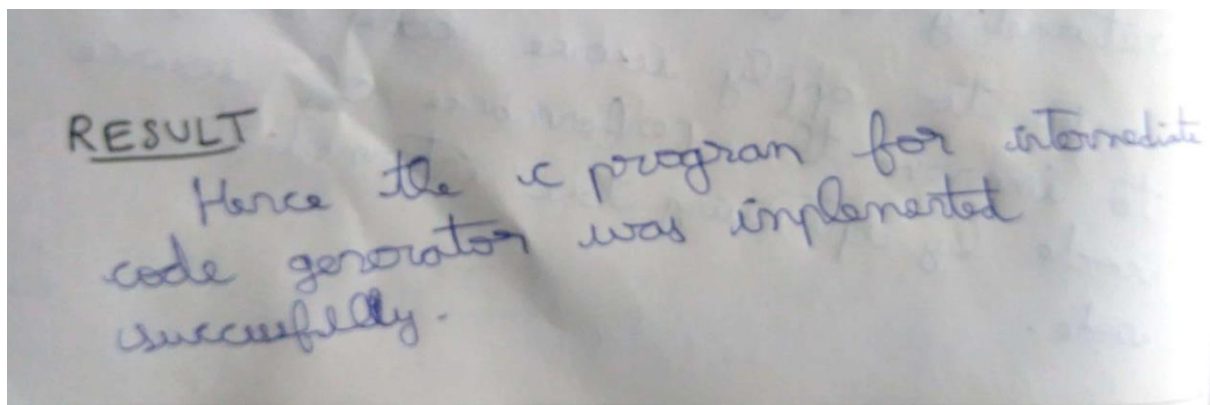
## OUTPUT FILE:



The image shows a screenshot of a code editor interface. At the top, there is a toolbar with buttons for 'Run', 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. Below the toolbar, there are tabs for 'main.c', 'input.txt', and 'output.txt'. The 'main.c' tab is active, displaying the following assembly code:

```
1  
2 MOV R0,a  
3 ADD R0,b  
4 MOV t1,R0  
5 MOV R0,c  
6 MUL R0,d  
7 MOV t2,R0  
8 MOV R0,t1  
9 SUB R0,t2  
10 MOV t,R0  
11 MOV R0,t  
12 MOV x,R0
```

Below the code editor, there is a terminal window with the text: "...Program finished with exit code 0" and "Press ENTER to exit console." followed by a cursor.



RESULT  
Hence the c program for intermediate code generator was implemented successfully.