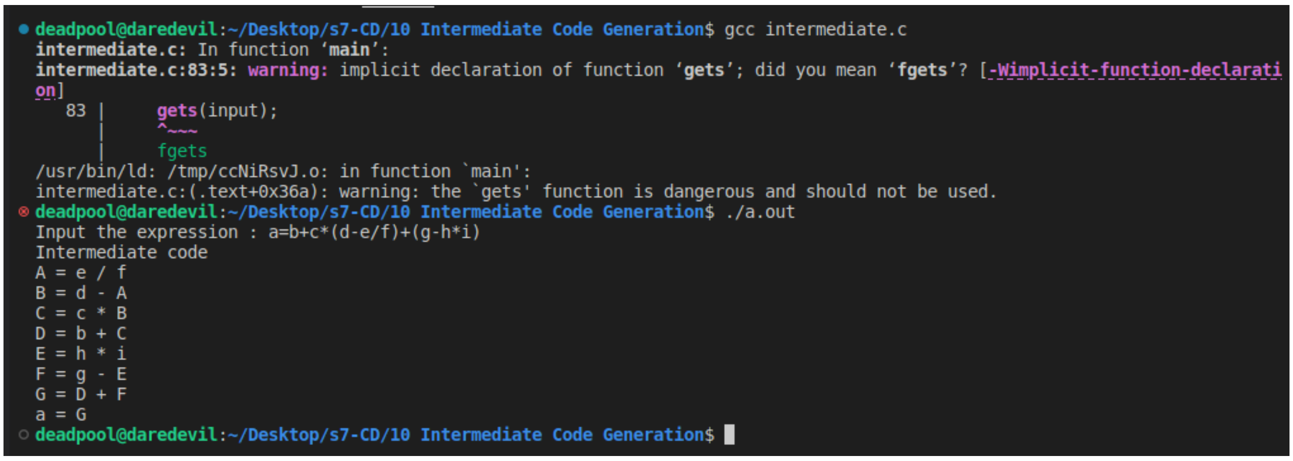
Exp15 : Intermediate Code Generation

```
1. Start
2. Read the code in array – input
3. infix_to_postfix()
4. top = -1
5. intermediate()

6. void push(c)
    1. top++
    2. stack[top] = c

7. int pop()
    1. return (stack[top--])

8. int priority(c)
    1. switch(c)
        1. case # : return 0
        2. case ( : return 1
        3. case + : case - : return 2
        4. case * : case / : return 3

9. void infix_to_postfix()
    1. i = 2, k = 0
    2. push( # )
    3. while ( ch = input[i-2] ) != '\0'
        1. if ch = ( then
            1. push( ch )
        2. else if ch = )
            1. while stack[top] != ( do
                1. postfix[k] = pop()
                2. k++
            2. d = pop()
        3. else if isalnum( ch )
            1. postfix[k] = ch
            2. k++
        4. else
            1. while pr(stack[top]>=pr(ch) do
                1. postfix[k] = pop()
                2. k++
            2. push ( ch )

        5. i++
    4. while( stack[top] != # )
        1. postfix[k] = pop()
        2. k++
    5. postfix[k] = '\0'

10. void intermediate()
    1. i = 0 , num = A
    2. while ( ch = postfix[i] ) != '\0' do
        1. if ( isalnum(ch) ) then
            1. push( ch )
        2. else
            1. op1 = pop()
            2. op2 = pop()
            3. print num = op1 ch op2
            4. push( num )
            5. num++
        3. i++
    3. print input[0] = stack[top]
11. Stop
```

Intermediate Code Generation (C)

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

char stack[50];
int top = -1;
char input[50];
char postfix[50];

void push(char elem) {
   top++;
   stack[top] = elem;
}

char pop() {
   return (stack[top--]);
}


int pr(char elem) {
   switch (elem) {
      case '#' : return 0;
      case '(' : return 1;
      case '+' : case '-' : return 2;
      case '*' : case '/' : return 3;
   }
}

void infix_to_postfix() {
   char ch, d;
   int i = 2, k = 0;

   push('#');

   while ((ch = input[i]) != '\0') {
      if (ch == '('){
         push(ch);
      } else if (isalnum(ch)){
         postfix[k] = ch;
         k++;
      } else if (ch == ')') {
         while (stack[top] != '('){
             postfix[k] = pop();
             k++;
         }
         d = pop();
      } else {
         while (pr(stack[top]) >= pr(ch)){
             postfix[k] = pop();
             k++;
         }
         push(ch);
      }
      i++;
   }

   while (stack[top] != '#'){
      postfix[k] = pop();
      k++;
   }
   postfix[k] = '\0';
}

void intermediate() {
   char ch,op1,op2,num = 'A';
   int i = 0;
   while((ch = postfix[i]) != '\0') {
      if(isalnum(ch)){
         push(ch);
      }else{
         op2 = pop();
         op1 = pop();
         printf("%c = %c %c %c\n",num,op1,ch,op2);
         push(num);
         num++;
      }
      i++;
   }
   printf("%c = %c\n",input[0],stack[top]);
}

void main() {
   printf("Input the expression : ");
   gets(input);
   infix_to_postfix();
   top = -1;
   printf("Intermediate code\n");
   intermediate();
}
```

output



```
● deadpool@daredevil:~/Desktop/s7-CD/10 Intermediate Code Generation$ gcc intermediate.c
  intermediate.c: In function ‘main’:
  intermediate.c:83:5: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’? [-Wimplicit-function-declarati
  on]
     83 |     gets(input);
        |     ^~~~
        |     fgets
  /usr/bin/ld: /tmp/ccNiRsvJ.o: in function `main':
  intermediate.c:(.text+0x36a): warning: the `gets' function is dangerous and should not be used.
⊗ deadpool@daredevil:~/Desktop/s7-CD/10 Intermediate Code Generation$ ./a.out
  Input the expression : a=b+c*(d-e/f)+(g-h*i)
  Intermediate code
  A = e / f
  B = d - A
  C = c * B
  D = b + C
  E = h * i
  F = g - E
  G = D + F
  a = G
○ deadpool@daredevil:~/Desktop/s7-CD/10 Intermediate Code Generation$
```