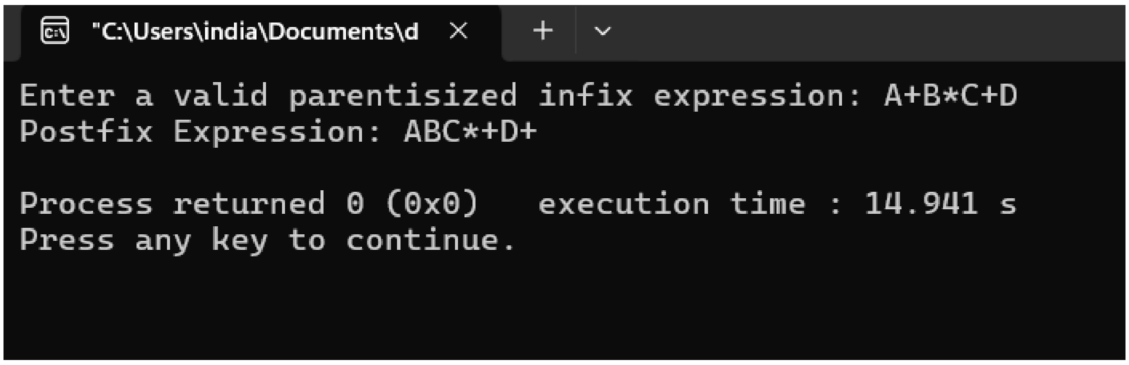
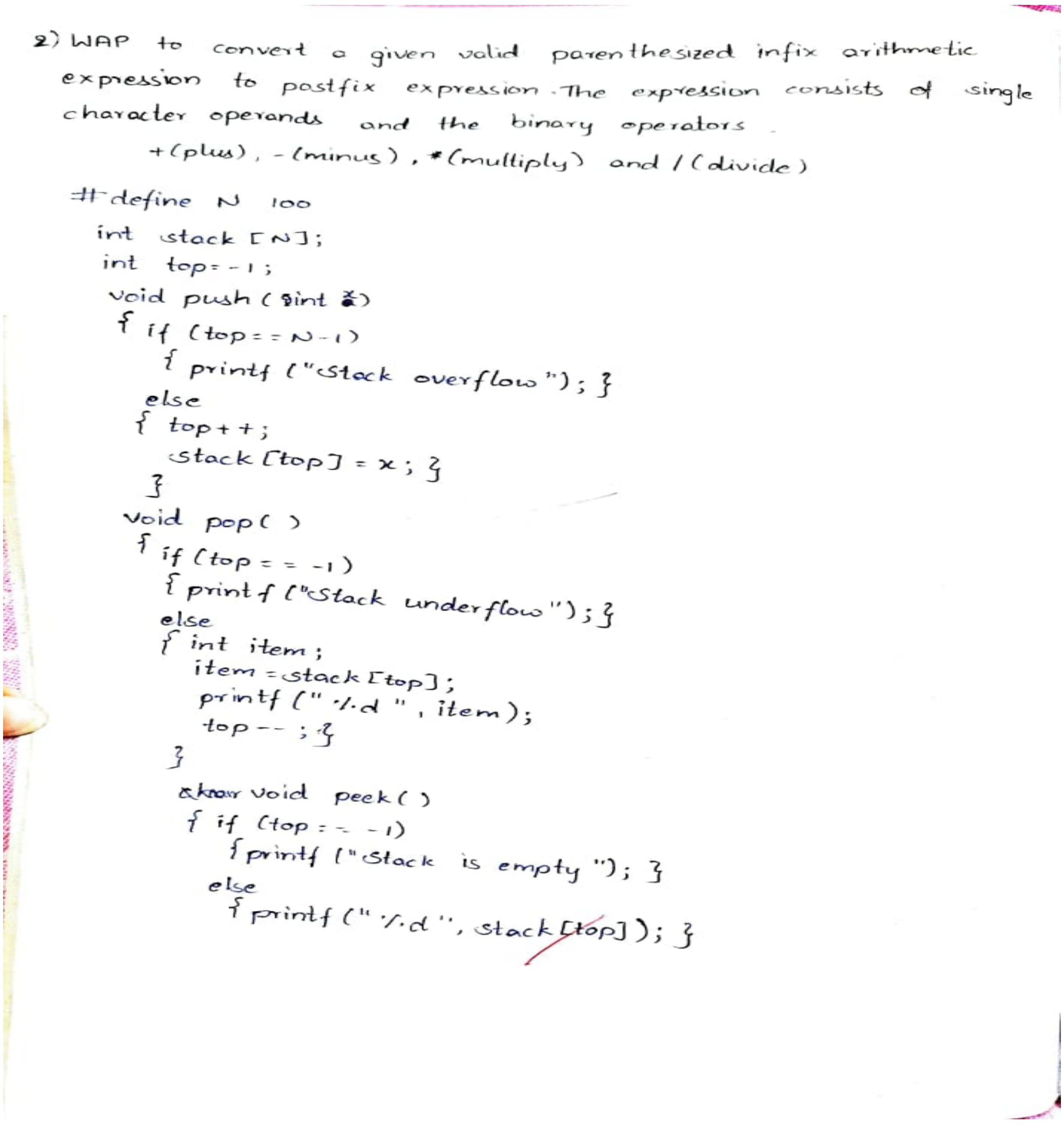
LAB 2



```
Enter a valid parentisized infix expression: A+B*C+D
Postfix Expression: ABC*+D+

Process returned 0 (0x0)   execution time : 14.941 s
Press any key to continue.
```



2) WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators . +(plus), -(minus), *(multiply) and /(divide)

```
#define N 100
 int stack [N];
 int top=-1;
 void push ( int x)
 { if (top==N-1)
    { printf ("Stack overflow"); }
   else
   { top++;
     stack [top] = x; }
  }
 void pop( )
 { if (top == -1)
   { printf ("Stack underflow"); }
   else
   { int item;
      item =stack [top];
      printf (" %d ", item);
      top-- ; }
   }
   ~~char~~ void peek( )
   { if (top == -1)
      {printf ("Stack is empty"); }
     else
      {printf (" %d ", stack [top]); }
```

```
int precedence (char op)
{ switch (op)
  { case '+':
   case '-' : ~~push~~ return 1;
   case '*' :
   case '/' : return 2;
   case '^' : return 3;
   case '(' : return 0; }
    return -1;
    } return 0; }

    int associativity (int op)
    { if (op == '^')
    { return 1; }
      return 0; }

   ~~int precedence(~~
   void infix to postfix (int infix[], int postfix[])
   { int i,k=0;
    char c;
    for (i=0; infix [i] != '\0' ; i++)
     { c = infix [i];
       if (isalnum(c)) {
       postfix [k++] = c; }
      else if (c == '(')
      {push (c);
```

else if it is an operator then check the highest precidence and the push it into the stack if it is of ~~highest~~ lowest precedence.

if it is of highest precidence then pop the highest ~~prec~~ operator and print it in the output.

If it is of equal precedence then check the associative rule

```
Void infixTopostfix (char infix[], char postfix[])
{
 int i, k = 0;
 char c;
 for (i=0; infix[i] != '\0' ; i++)
  { c = infix[i];
    if (isalnum(c))
     {
      postfix[k++]=c;
     }
    else if (c == '(')
     { push(c) ; }
    else if (c == ')')
     {
      while peek() != '(')
       {
        postfix[k++] = pop();
       }
       pop();
     }
    else
     { while (top != -1 && (precedence (peek()) > (precedence (c)) ||
           ((precedence (peek())) == (precedence(peek())) &&
              associativity (c) = = 0 ))
     {
      postfix[k++] = pop();
      }
     push(c);
     }
  }
  while (top != -1)
   {
    postfix [k++] = pop();
   }
   postfix [k] = '\0';
  }
```

```
int main ()
 {
  char infix [N], postfix [N];
  printf ("Enter a parentisized infix expression : ");
  scanf ("%s", infix);
  infixtopostfix (infix, postfix);
  printf ("Postfix Expression : %s \n", postfix);
  return 0;
  }
```

Output:

```
Enter a parentisized infix expression : A+B*C+D
Postfix Expression : ABC*+D+
```

14/10/25

[illegible]