```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lexical.h"
#include "nextInputChar.h"
#include "tokenStack.h"

/*
        FAHAD AHMED KHAN
        214468888

        This code simulates a RPN calculator as it takes an input from the user,
        and decodes it to perform a calculation.
*/

static int popInt(struct tokenStack * s) {
  if (s -> top <= 0) {
    fprintf(stderr, " popInt : popping an empty stack,aborting \n ");
    exit(1);
  }

  struct lexToken * topToken = popTokenStack(s);
  char number = topToken -> symbol[0];

  int intnum = number - '0';
  freeToken(topToken);
  return intnum;
}

static void pushInt(struct tokenStack * s, int v) {
  struct lexToken * newtoken = allocToken();
  newtoken -> kind = LEX_TOKEN_NUMBER;
  char numchar;

  /* Handles negative numbers */
  if (v < 0) {
    v = v * -1;
    newtoken -> symbol[0] = '-';
    numchar = v + '0';
    newtoken -> symbol[1] = numchar;
    newtoken -> symbol[2] = '\0';
  } else {
    numchar = v + '0';
    newtoken -> symbol[0] = numchar;
    newtoken -> symbol[1] = '\0';
  }
  pushTokenStack(s, newtoken);
}

static void doOperator(struct tokenStack * s, char * op) {

  if (!strcmp(op, "quit")) {
    exit(0);
  } else if (!strcmp(op, "print")) {
    struct lexToken * t = popTokenStack(s);

    dumpToken(stdout, t);
    freeToken(t);
  } else {
    fprintf(stderr, "don't know |%s|\n", op);
    exit(1);
  }
}
```

```
int main(int argc, char *argv[]) {
  setFile(stdin);
  struct tokenStack * stack = createTokenStack();
  /* For tokens of type LEX_TOKEN_EOF your code should quit
     For tokens of type LEX_TOKEN_IDENTIFIER your code should call doOperator
     For tokens of type LEX_TOKEN_NUMBER your code should push the token on the stack *
/
  int type = 0;
  do {
    struct lexToken * next = nextToken();
    int type = next -> kind;
    char input = next -> symbol[0];
    char *identString = next -> symbol;
    double op2;
    int x;

    switch (type) {
    case LEX_TOKEN_IDENTIFIER:
      doOperator(stack, identString);
      break;
    case LEX_TOKEN_NUMBER:
      pushTokenStack(stack, next);
      break;
    case LEX_TOKEN_OPERATOR:
      switch (input) {
      case '+':
        x = popInt(stack) + popInt(stack);
        pushInt(stack, x);
        break;
      case '*':
        pushInt(stack, popInt(stack) * popInt(stack));
        break;
      case '-':
        op2 = popInt(stack);
        pushInt(stack, popInt(stack) - op2);
        break;
      case '/':
        op2 = popInt(stack);
        if (op2 != 0.0)
          pushInt(stack, popInt(stack) / op2);
        else
          fprintf(stderr, " zero divisor \n ");
        break;
      }
    }
  }
  while (type != LEX_TOKEN_EOF);

  fprintf(stdout, "%d \n", popInt(stack));

}
```