```
#include <stdio.h>
#include <string.h>
#include "stack.h"

/
*****************************************************************************
************
    D E C L A R A T I O N S
*****************************************************************************
************/

struct record {
    char name [32];
    int declare;
    char type[5];
    };

#define TID_SIZE 1000

record TID[TID_SIZE]; //identifiers table

struct Lex{
   int ltype;/*type of lexeme*/
   int value;/*index of the string in the corresponding table*/
};

Lex c;//current lexeme

//to fill the declare and type fields
void dec_id(int i, const char* t){
   if (TID[i].declare==1) throw "Redeclaration"
   TID[i].declare =1;
   strcpy(TID[i].type, t)
}

Stack<int> Si(TID_SIZE);

void check_types(const char* t){
   int i;
   while ((i=Si.pop())!= -1)
    dec_id(i,t);
}

/
*****************************************************************************
********
    E X P R E S S I O N S
*****************************************************************************
********/
const char* gettype (const char * op, const char * t1, const char* t2);

Stack<const char*> Scc (TID_SIZE);
const char * TD[13] = {"+","-","*","/","or","and","not","<","<=",">",">=","!
=","="} //should be extended

void checkid(){
  int i;
  i=c.value;
  if (TID[i].declare) Scc.push(TID[i].type);
   else throw "Undeclared identifier";
}


void checkop(){
```

```
  const char* op,*t1,*t2, *res;
  t2=Scc.pop();
  op=Scc.pop();
  t1=Scc.pop();
  res = gettype(op,t1,t2);
  if (res) Scc.push(res);
   else throw "Incompatible types";
}

void checknot(){
  if (!strcmp(Scc.pop(),"bool")) Scc.push("bool");
  else throw "Type bool is expected"
}
```