```
#ifndef SYMBOLTABLE_H
#define SYMBOLTABLE_H

#include <string>
#include <vector>
#include <sstream>
#include <iomanip>
#include <stack>
#include "Lexer.h"
#include "Globals.h"

// Begin instruction address at 1
static int instr_address = 1;

class SymbolTable
{
public:
  struct Instr
  {
    int address;
    std::string op;
    int operand;

    Instr(std::string op, int operand) : address(instr_address++), op(op), operand(operand) {}
  };

  struct Symbol
  {
    Lexer::Token token;
    int address;
    std::string type;

    Symbol(Lexer::Token t, int address, std::string type) : token(t), address(address), type(type) {}
  };

  SymbolTable();

  ~SymbolTable();

  // Accessors

        // Symbol Table
  int lookup(Lexer::Token t);
  std::string get_type(Lexer::Token token) const;
  int get_mem();
  int get_address(Lexer::Token token);

        // Output
  std::string list();
  std::string list_instr();

        // Instruction table
  int get_instr_address() const;

        // Typestack
  std::string top_typestack() const;
  bool typestack_empty() const;

  // Mutators

        // Symbol Table
  bool insert(Lexer::Token t, std::string type);
  bool remove(Lexer::Token t);

        // Instruction table
  void gen_instr(std::string op, int operand);

        // Jumpstack
  void push_jumpstack(int address);
  void back_patch(int jump_addr);

        // Typestack
  void push_typestack(std::string type);
  bool pop_typestack();

private:

  void incrementMem();

  std::vector<Symbol> table;
  std::vector<Instr> instructions;
  std::vector<int> jumpstack;
  std::stack <std::string> typestack;
  int memaddress;
  std::ostringstream error;
};

#endif // SYMBOLTABLE_H
```