Help

```
#ifndef PARSER_H
#define PARSER_H


#include "pnl/pnl_matrix.h"
#include <iostream>
#include <map>
#include <string>
#include <vector>
#include <cstring>
#include <boost/variant.hpp>
#include <boost/serialization/map.hpp>
#include <boost/serialization/string.hpp>
#include <boost/serialization/vector.hpp>
#include <boost/serialization/variant.hpp>
#include <boost/serialization/version.hpp>
#include <boost/static_assert.hpp>
#include <boost/type_traits.hpp>

#define MAX_CHAR_LINE 1024
// #define DEBUG

/* list of possible types */
typedef enum
{
  T_NULL,
  T_INT,
  T_LONG,
  T_DOUBLE,
  T_VECTOR,
  T_STRING,
  T_PTR,
} T_type;

class TypeVal
{
public:
  T_type type;
  boost::variant<int, size_t, double, std::vector<double>, std::string, void *>
```

```
  TypeVal();
  TypeVal(const TypeVal &);
  // Be sure not to delete anything because we rely on copy by address.
  ~TypeVal();
  TypeVal& operator= (const TypeVal &v);
  void print(const std::string &s) const;

private:
  friend class boost::serialization::access;
  template<class Archive> void serialize(Archive& ar, const unsigned int version
  {
    ar & type;
    ar & Val;
  }

};

struct comp
{
  bool operator() (const std::string& lhs, const std::string& rhs) const {
    return strcasecmp(lhs.c_str(), rhs.c_str()) < 0;
  }
};


typedef std::map<std::string, TypeVal, comp> Hash;

class Param
{
public:
  Hash M;
  Param() { }
  Param(const Param&);
  ~Param();
  Param& operator=(const Param &P);


  template <typename T> bool extract(const std::string &key, T &out, bool go_on
  {
    BOOST_STATIC_ASSERT(!(boost::is_same<T, PnlVect*>::value));
    Hash::const_iterator it;
```

```
  if (check_if_key(it, key) == false)
    {
      if (!go_on)
        {
          std::cout << "Key " << key << " not found." << std::endl;
          abort();
        }
      else return false;
    }
  try
    {
      out = boost::get<T>(it->second.Val);
      return true;
    }
  catch (boost::bad_get e)
    {
      std::cout << "Boost bad get for " << key << std::endl;
      abort();
    }
}

bool extract(const std::string &key, PnlVect  *&out, int size, bool go_on = fa

template <typename T> bool set(const std::string &key, const T &in)
{
  Hash::iterator it;
  if ((it = M.find(key)) == M.end()) return false;
  try
    {
      boost::get<T>(it->second.Val) = in;
      return true;
    }
  catch (boost::bad_get e)
    {
      std::cout << "Boost bad get for " << key << std::endl;
      abort();
    }
}

/**
 * Insert a new pair in the map or set M[key] to the new value if the key alre
```

```
   *
   * @tparam T the template type of the element to be inserted
   * @param key the key
   * @param t the type of the elements as an integer
   * @param in the element itself
   */
  template <typename T> void insert(const std::string &key, const T_type &t, con
  {
    if (M.find(key) != M.end())
      {
        set<T>(key, in);
        return;
      }
    TypeVal V;
    V.type = t;
    V.Val = in;
    M[key] = V;
  }

  void print() const
  {
    Hash::const_iterator it;
    for (it = M.begin() ; it != M.end() ; it++) it->second.print(it->first);
  }
private:
  bool check_if_key(Hash::const_iterator &it, const std::string &key) const;

  friend class boost::serialization::access;
  template<class Archive> void serialize(Archive& ar, const unsigned int version
  {
      ar & BOOST_SERIALIZATION_NVP(M);
  }
};

class Parser : public Param
{
public:
  Parser();
  Parser(const char *InputFile);
  ~Parser();
  void add(char RedLine[]);
```

```
private:
  void ReadInputFile(const char *InputFile);
  char type_ldelim;
  char type_rdelim;
};

#endif
```