C++ Programming - Intermediate - Assignment

Author: Peter Tse (mcreng)

Introduction

In a low level perspective, programming is about calling different addresses and changing the corresponding values. In this assignment, you will be handling variables and addresses, on how to allocate memory for them and modify their contents.

Provided Header

```
/* address.h */
/* DO NOT CHANGE ANYTHING */
#include <vector>
#include <string>
#include <iostream>
#include <sstream>

#ifndef __ADDRESS_H_
#define __ADDRESS_H_

struct Entry{
  Entry(std::string name, int* addr) :
        name_(name),
        addr_(addr){}
  std::string name_{};
  int* addr_ = nullptr;
};

class Address{
  public:

  /**
   * @brief Parse the arguments input by user
   *
   * Three commands supported:
   *
   * >add <name> <init value>
   * Corresponds to add_entry(std::string name, int value)
   *
   * >del <address>
   * Corresponds to del_entry(int* address)
```

```
 * If del * is entered, it is treated as deleting all addresses
 *
 * >chg <address> <new value>
 * Corresponds to chg_entry(int* address, int value)
 *
 * @param arguments Arguement input
 * @return Whether the arguements are valid
 */
static bool parse_arg(const std::vector<std::string>& arguments);

/**
 * @brief Add a variable to our memory
 * @param name Variable name
 * @param value Initialzed value
 * @return Whether the action is successful
 */
static bool add_entry(const std::string& name, const int& value);

/**
 * @brief Delete a variable from our memory
 * @param address Variable address
 * @return Whether the action is successful
 */
static bool del_entry(int* address);

/**
 * @brief Change the value of a variable in our memory
 * @param address Variable address
 * @param value New value
 * @return Whether the action is successful
 */
static bool chg_entry(int* address, const int& value);

/**
 * @brief Print entered entries in our memory
 * In format of:
 * <address> "\t" <name> "\t" <value>
 * as specified in main.cpp
 */
static void print_data();

/**
 * @brief Convert a string to an integer
 * @param str Input string
 * @param isHex Whether the integer is considered a hexadecimal
 * @return int Converted integer
```

```
   * @return bool Whether the action is successful
   */
  static std::pair<int, bool> strtoint(const std::string& str, bool isHex = false){
        int result;
        std::istringstream convert(str);
        if (!isHex){
          if (!(convert >> result)) return std::make_pair(0, false);
        } else {
       if (!(convert >> std::hex >> result)) return std::make_pair(0, false);
        }
        return std::make_pair(result, true);
  }

  private:
  static std::vector<Entry> entries;
};

#endif
```

Provided Program Entry

```
/* main.cpp */
/* DO NOT CHANGE ANYTHING */
#include "address.h"
#include <vector>
#include <string>
#include <iostream>

int main(){
  std::vector<std::string> arguments;
  std::string temp{};
  while (true){
        arguments.clear();
        std::string line{};
        std::getline(std::cin, line);
        std::istringstream iss(line);
    while (!iss.eof()){
      iss >> temp;
      arguments.push_back(temp);
    };
    if (arguments.size() == 1 && arguments.at(0) == "exit") return 0;
    if (!Address::parse_arg(arguments)) std::cout << "Invalid input" << std::endl;
    Address::print_data();
  };
}
```

Explanation

In the header file, you would see something in a format of

```
/**
 *
 *
 */
```

This is the syntax of C++ documentations. Inside the documentations, there are informations that you may find useful. The `@brief` tag tells you about the brief information of the functions, the `@param` tag tells you the parameters the functions take and the `@return` tag tells you the return value of the functions. Note that the documentations are located **before** the prototypes, Read them yourselves in the code.

This program contains a user input/output interface and the input interface has already provided in `main.cpp`. However, you still need to parse the arguments yourself. There are only three commands supported, namely

- `add`

  which can add an entry to the memory with certain variable name and variable type;

- `del`

  which can remove an entry based on the address provided; and

- `chg`

  which can change the value of an entry based on the address provided.

After any user inputs, the program in `main.cpp` would put the arguments and parameters input to `arguments` with type `std::vector<std::string>`, and output the current entries entered with formatting (using `"\t"` tab characters). Your job is to parse the arguments, finish the `add`, `del` and `chg` functions and the output function in this assignment. You need to make sure there will be no potential crashes such as trying to access invalid addresses and incorrect inputs. You may find the function `std::pair<int, bool> strtoint(const std::string&, bool)` useful. Note that the `exit` command is handled by `main.cpp` already.

Expected I/O Results

This is to provide a set of correct input/output result of the program. Lines with `>` at the start indicate an user input.

```
>add var1 10
Address                 Name        Value
```

```
0x9f1ff8        var1        10
>add var2 50
Address                 Name        Value
0x9f1ff8        var1        10
0x9f2008        var2        50
>chg 0x9f1ff8 205
Address                 Name        Value
0x9f1ff8        var1        205
0x9f2008        var2        50
>del 0x09f2007
Invalid input
Address                 Name        Value
0x9f1ff8        var1        205
0x9f2008        var2        50
>chg 0x9f2008 d
Invalid input
Address                 Name        Value
0x9f1ff8        var1        205
0x9f2008        var2        50
>del 0x9f2008
Address                 Name        Value
0x9f1ff8        var1        205
>del
Invalid input
Address                 Name        Value
0x9f1ff8        var1        205
>del *
Address                 Name        Value
>exit
```

Submission

Another C++ program which includes `address.h` is used to check whether your functions have been implemented correctly. Please, therefore, make sure you do not change the filename and function names when you submit your files. Finish your implementation in `address.cpp` and submit it with a comment containing your name. Submission deadline is ????/??/??.