Week 4

Exercise 32

../32/stringManip.h

```
1  #ifndef INCLUDED_STRINGMANIP_
2  #define INCLUDED_STRINGMANIP_
3
4  class StringManip
5  {
6    std::string d_source;
7
8    public:
9      StringManip(std::string const &source);
10
11   private:
12
13
14     std::string lc() const;               // return a copy of d_source in
15                                           // lower-case chars
16     std::string uc() const;               // return a copy in upper-case
17                                           // chars
18
19     int compare(std::string &rhs) const;  // -1: d_source first, 0: equal
20                                           // 1: rhs first, case insensitive
21                                           // comparison.
22
23     std::string copy() const;       // return a copy of d_source
24 };
25
26 // Changes
27 // - Safer make member functions constant, to prevent changes to member data
28 // -- No const return types, as all return types are copies or new variables
29 // - Include guards
30 // - Safer to keep those functions private, except for constructor,
31 //   which needs to be publicly accessible.
32
33 #endif
```

Exercise 33

../33–34/person/person.h

```
1  // Person class: interface header
2
3  #ifndef INCLUDED_PERSON_
4  #define INCLUDED_PERSON_
5
6  #include <string>
7  #include <iostream>
8
9  class Person
10 {
11   std::string d_name;     // name of person
12   std::string d_address;  // address field
13   std::string d_phone;    // telephone number
14   size_t      d_mass;     // the mass in kg.
15
16   public:
17     std::string const &name()     const;
18     std::string const &address()  const;
19     std::string const &phone()    const;
20     size_t mass()                 const;
21     // Getters
22
23     void insert(std::ostream &outputStream);  // Storing data
24     void extract(std::istream &inputStream);  // Extracting data
25
26   private:
27     void setName(std::string const &name);
28     void setAddress(std::string const &address);
29     void setPhone(std::string const &phone);
30     void setMass(size_t mass);
31     // Setters
32 };
33
34 // Basic inline functions
35
36 inline void Person::setName(std::string const &name)
37 {
38   d_name = name;
39 }
40
41 inline std::string const &Person::name() const
42 {
43   return d_name;
44 }
45
46 inline void Person::setAddress(std::string const &address)
47 {
48   d_address = address;
49 }
50
51 inline std::string const &Person::address() const
52 {
53   return d_address;
54 }
55
56 // Phone number setter defined seperately
57
58 inline std::string const &Person::phone() const
59 {
60   return d_phone;
61 }
62
```

```
63 inline void Person::setMass(size_t mass)
64 {
65   d_mass = mass;
66 }
67
68 inline size_t Person::mass() const
69 {
70   return d_mass;
71 }
72
73 #endif
```

../33–34/person/person.ih

```
1 // Person class: implementation internal header
2
3 #include "person.h"
4
5 #define CERR std::cerr << __FILE__": "
6
7 using namespace std;
```

../33–34/person/extract.cc

```
1  // Person member function: extract person data from istream
2
3  #include "person.ih"
4
5  void Person::extract(istream &inputStream)
6  {
7    string varValue; // Initialise string to be used to populate variables
8    for (size_t index = 0; index != 3; ++index) // Loop through first three vars
9    {
10     if (!getline(inputStream, varValue, ',')) // Read until a comma is found,
11       break;
12     switch (index)                            // assign that to the string
13     {                                         // and then assign the string
14       case 0:                                 // to the vars in order
15         setName(varValue);
16         break;
17       case 1:
18         setAddress(varValue);
19         break;
20       case 2:
21         setPhone(varValue);
22         break;
23       default:                                // Should never happen, but
24         break;                                // it's good practice to include it,
25     }                                         // right?
26   }
27   if (!getline(inputStream, varValue))        // For the last var, read until
28     return;                                   // new-line char and assign it to mass
29   setMass(stoi(varValue));
30 }
```

../33–34/person/insert.cc

```
1 // Person member function: insert data into ostream
2
3 #include "person.ih"
4
5 void Person::insert(ostream &outputStream)
6 {
7   outputStream  << "NAME:    " << name()     << '\n'
8                 << "ADDRESS: " << address()  << '\n'
```

```
9                 << "PHONE:    " << phone()     << '\n'
10                << "MASS:     " << mass()      << '\n';
11  }
12  // Inserts all object characteristics into ostream. It was assumed that the
13  // variable identifiers were also desirable.
```

../33–34/person/setPhone.cc

```
1   // Person member function: set phone number after verification
2
3   #include "person.ih"
4
5   void Person::setPhone(string const &phone)
6   {
7     if (phone.empty())
8       d_phone = " - not available -";
9     else if (phone.find_first_not_of("0123456789") == string::npos)
10      d_phone = phone;
11    // Switched the two options above around from the example, as an empty string
12    // will also not contain any non-numerical characters.
13    else
14      cout << "A phone number may only contain digits\n";
15  }
```

Exercise 35

../35/main.ih

```
#include <iostream>
#include <string>
#include "user/user.h"

using namespace std;
```

../35/main.cc

```
#include "main.ih"


int main(int argc, char **argv)
{

  User me ; //constructing object me

  cout << "valid\t\t" << (me.valid() ? "true" : "false") << '\n';

  if (me.valid())
    cout << "name\t\t" << me.name() << '\n'
     << "groupId\t\t" << me.groupId() << '\n'
     << "homeDir\t\t" << me.homeDir() << "/\n"
     << "realName\t" << me.realName() << '\n'
     << "shell\t\t" << me.shell() << '\n'
     << "userId\t\t" << me.userId() << '\n';


}
```

../35/user/user.h

```
#ifndef INCLUDED_USER_
#define INCLUDED_USER_

#include <string>
#include <sys/types.h>
#include <pwd.h>

class User
{
    std::string   d_pw_name;       // username
    std::string   d_pw_passwd;     // user password
    uid_t         d_pw_uid;        // user ID
    gid_t         d_pw_gid;        // group ID
    std::string   d_pw_gecos;      // user information
    std::string   d_pw_dir;        // home directory
    std::string   d_pw_shell;      // shell program

    public:
      User();
      bool valid() const;
      size_t groupId() const;
      bool inGroup(size_t gid) const;
      std::string const &name() const;
      std::string const &homeDir() const;
      std::string const &realName() const;
      std::string const &shell() const;
      size_t userId() const;

};

```

```
31  inline std::string const &User::name() const
32  {
33    return d_pw_name;
34  }
35
36  inline size_t User::groupId() const
37  {
38    return d_pw_gid;
39  }
40
41  inline std::string const &User::homeDir() const
42  {
43    return d_pw_dir;
44  }
45
46  inline std::string const &User::realName() const
47  {
48    return d_pw_gecos;
49  }
50
51  inline std::string const &User::shell() const
52  {
53    return d_pw_shell;
54  }
55
56  inline size_t User::userId() const
57  {
58    return d_pw_uid;
59  }
60
61
62
63  #endif
```

../35/user/user.ih

```
1  #include "user.h"
2
3  using namespace std;
```

../35/user/constructor.cc

```
1  // User: constructor
2
3  #include "user.ih"
4
5  User::User()
6  {
7    struct passwd *pwd = getpwuid(1000); //Putting the user data in the struct
8
9    if  ((*pwd).pw_name == 0)  //checking if this user exists
10     return;                  // and we dont have a null deref
11
12   d_pw_name = (*pwd).pw_name;         //Setting the values of the
13   d_pw_passwd = (*pwd).pw_passwd;     //newly constructed object.
14   d_pw_uid = (*pwd).pw_uid;
15   d_pw_gid = (*pwd).pw_gid;
16   d_pw_gecos =  (*pwd).pw_gecos;
17   d_pw_dir =  (*pwd).pw_dir;
18   d_pw_shell = (*pwd).pw_shell;
19 };
```

../35/user/inGroup.cc

```
1  #include "user.ih"
```

```
2
3  bool User::inGroup(size_t gid) const
4  {
5    return gid == d_pw_gid; //checking whether gid is equal to the pw_gid field
6  }
```

../35/user/valid.cc

```
1  #include "user.ih"
2
3  bool User::valid() const
4  {
5    return  !User::name().empty()  && User::groupId() != 0  &&
6     !User::homeDir().empty() &&  !User::realName().empty() &&
7     !User::shell().empty()  && User::userId() != 0 ;
8  }
9
10 //Checks whether the object is properly constructed and all the fields
11 //are filled in.
```

Exercise 36

../36/enums/enums.h

```
1  // ENUMS class: header file
2
3  #ifndef INCLUDED_ENUMS_
4  #define INCLUDED_ENUMS_
5
6  #include <cstddef>
7
8  enum class RAM: size_t
9  {
10   SIZE = 20
11 };
12
13 enum class Opcode: size_t
14 {
15   ERR,
16   MOV,
17   ADD,
18   SUB,
19   MUL,
20   DIV,
21   NEG,
22   DSP,
23   STOP
24 };
25
26 enum class OperandType: size_t
27 {
28   SYNTAX,
29   VALUE,
30   REGISTER,
31   MEMORY
32 };
33
34 #endif
```

../36/enums/enums.ih

```
1  // Enums class: implementation internal header
2
3  #include "enums.h"
4
5  #define CERR std::cerr << __FILE__": "
6
7  using namespace std;
```

../36/memory/memory.h

```
1  // Memory class: header file
2
3  #ifndef INCLUDED_MEMORY_
4  #define INCLUDED_MEMORY_
5
6  #include <cstddef>
7
8  class Memory
9  {
10   public:
11     size_t const &load(size_t *valueAddress) const;
12     void store(size_t value, size_t *address);
13 };
14
```

```
15 #endif
```

../36/memory/memory.ih

```
1  // Memory class: internal header file
2
3  #include "memory.h"
4  //#define CERR std::cerr << __FILE__": "
5
6  using namespace std;
```

../36/memory/load.cc

```
1  // Memory function: load
2
3  #include "memory.ih"
4
5  size_t const &Memory::load(size_t *valueAddress) const
6  {
7    return *valueAddress;
8  };
```

../36/memory/store.cc

```
1  // Memory function: store
2
3  #include "memory.ih"
4
5  void Memory::store(size_t value, size_t *address)
6  {
7    *address = value;
8  }
```