Regular: Jaap van der Leest
Attendee: Timon van der Berg
October 11, 2017

C++ Course
Assignment 4

Exercise 26

Problem statement. Describe in your own words what 'encapsulation' and 'data hiding' means, and why these concepts are important when designing classes. Provide a small example of a self-defined class illustrating your explanation. Why is the implementation of a class irrelevant to submit?
Solution. Encapsulation means that the data in a class is not directly available outside the class. Instead, member functions provide a way to access the data of the class; the class *encapsulates* the data. The data can only be used through the interface, the header file of the class and the functions defined within the header. In this sense the data is hidden from the outside of the class. "Data hiding" means that no data from within the class can be seen by other code. Not accessed nor altered.

An example of a class interface

```
class Vector{
    int d_xval;
    int d_yval;
    public:
        void setXval(int xval);                 // sets d_xval
        void setYval(int yval);                 // sets d_yval

        int const &xval()  const;              // reads d_xval
        int const &yval()  const;              // reads d_yval

        size_t length(int xval, int yval);     // compute length of vector (a,b)
};
```

Here the data `int d_xval` and `int d_yval` is not accessable from the outside of the class. The member functions `setXval` and `setYval` will assign values to `d_xval` and `d_yval`. The member functions `xval` and `yval` can read the values of `d_xval` and `d_yval`. Finally, a member function `length` is declared, capable of reading and writing of the value. A user who calls `lenth()` need not worry about how the data is saved or how length is computed. In fact, the programmer may change this implementation at any time, for example to make the code faster. If the interface stays the same, the user should not notice. The data variables are also not available to the user, other than by calling the functions that read them. So everything regarding `d_xval`and `d_yval` will happen inside the class.

The implementation of a class is irrelevant because only what is described in the public part of the header is available to implement in other code. So the header defines what data and functions of a class can be used by others.

Exercise 27

main.cc:

```
#include "user.ih"

int main()
{
    User currentUser;
                                    // test all member functions except inGroup
```

```
    cout << "valid: " << currentUser.valid() << '\n';
    cout << "user id: " << currentUser.userId() << '\n';
    cout << "group id: " << currentUser.groupId() << '\n';
    cout << "homedir: " << currentUser.homeDir() << '\n';
    cout << "username: " << currentUser.name() << '\n';
    cout << "real name: " << currentUser.realName() << '\n';
    cout << "shell: " << currentUser.shell() << '\n';
}
```

user.ih:

```
#include <iostream>                // cin, cout
#include <pwd.h>                   // getpwuid
#include <sys/types.h>             // getuid
#include <unistd.h>                // getuid
#include "user.h"

using std::cout;    using std::cin;     using std::string;
```

user.h::

```
#ifndef INCLUDED_USER_
#define INCLUDED_USER_

#include <string>
#include "user.ih"

using std::string;

class User
{
    bool    d_valid;                            // construction succes
    size_t  d_groupId;
    string  d_homeDir;
    string  d_name;
    string  d_realName;
    string  d_shell;
    size_t  d_userId;

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

inline bool   User::valid()    const { return d_valid; }
```

```
inline size_t User::groupId()  const { return d_groupId; }
inline string User::homeDir()  const { return d_homeDir; }
inline string User::name()     const { return d_name; }
inline string User::realName() const { return d_realName; }
inline string User::shell()    const { return d_shell; }
inline size_t User::userId()   const { return d_userId; }

#endif
```

ingroup.cc:

```
#include "user.ih"

bool User::inGroup(size_t gid) const
{
    return gid == d_groupId;
}

```

user1.cc:

```
#include "user.ih"

User::User()
    :
        d_valid(false)
{
    d_userId = geteuid();                   // current user
    passwd *pw = getpwuid(d_userId);        // passwd file

    d_name = string(pw->pw_name);
    d_groupId  = pw->pw_gid;
    d_homeDir = string(pw->pw_dir);
    d_realName = string(pw->pw_name);
    d_shell = string(pw->pw_shell);

    d_valid = true;                         // succes
}
```

Exercise 28

enums/enums.h:

```
#ifndef INCLUDED_ENUMS_H
#define INCLUDED_ENUMS_H

#include <cstddef>          // size_t

enum RAM : size_t {
    SIZE = 20
};

```

```
enum class Opcode{
    ERR,
    MOV,
    ADD,
    SUB,
    MUL,
    DIV,
    NEG,
    DSP,
    STOP
};

enum class OperandType {
    SYNTAX,
    VALUE,
    REGISTER,
    MEMORY
};

#endif
```

memory/memory.ih:

```
#include "memory.h"
using namespace std;
```

memory/memory.h:

```
#ifndef INCLUDED_MEMORY_H
#define INCLUDED_MEMORY_H

#include <cstddef>
#include "../enums/enums.h"

class Memory
{
    int d_mem[RAM::SIZE];                // array of 20 ints

    public:
        Memory();
        void store(int value, size_t adress);
        int load(size_t adress) const;
    private:
};

#endif
```

memory/load.cc:

```
#include "memory.ih"

int Memory::load(size_t adress) const
{
```

```
    return (adress < RAM::SIZE ? d_mem[adress] : 0);
}


```

memory/memory1.cc:

```
#include "memory.ih"

Memory::Memory()
{
}
```

memory/store.cc:

```
#include "memory.ih"

void Memory::store(int value, size_t adress)
{
    if (adress <= RAM::SIZE)
        d_mem[adress] = value;
}
```

Exercise 29

cpu/cpu.ih:

```
#include "cpu.h"
#include "../tokenizer/tokenizer.h"
#include <iostream>

using namespace std;
```

cpu/cpu.h:

```
#ifndef INCLUDED_CPU_H
#define INCLUDED_CPU_H

#include "../enums/enums.h"
#include "../memory/memory.h"
#include "../tokenizer/tokenizer.h"

class Cpu
{
    int d_NREGISTERS[5] = {0, 0, 0, 0, 0};
    Memory d_memory;

    public:
        Cpu(Memory const &memory1);
        void start();

```

```
17          struct Operand
18          {
19              OperandType typeOfOperand;
20              int returnValue;
21          };
22
23      private:
24          bool err();
25          void mov();
26          void add();
27          void sub();
28          void mul();
29          void div();
30          void neg();
31          void dsp();
32
33  };
34
35  #endif
```

cpu/cpu1.cc:

```
1   #include "cpu.ih"
2
3
4   Cpu::Cpu(Memory const &memory1)
5   //:
6   {
7       d_memory = memory1;
8   }
```

cpu/err.cc:

```
1   #include "cpu.ih"
2
3   bool Cpu::err()
4   {
5       cout << "syntax error /n";
6       return false;
7   }
```

cpu/start.cc:

```
1   #include "cpu.ih"
2
3   void Cpu::start()
4   {
5       Tokenizer tokans;
6
7       while (true)
8       {
9           Opcode returnValue = tokans.opcode();  // request opcode from tokenizer
10
```

```
        switch (returnValue)
        {
            case Opcode::ERR: err();
            case Opcode::MOV: mov();
            case Opcode::ADD: add();
            case Opcode::SUB: sub();
            case Opcode::MUL: mul();
            case Opcode::DIV: div();
            case Opcode::NEG: neg();
            case Opcode::DSP: dsp();
            case Opcode::STOP: return;
        }

        tokans.reset();
    }

}
```

Exercise 30

tokenizer/tokenizer.h:

```
#ifndef INCLUDED_TOKENIZER_
#define INCLUDED_TOKENIZER_

#include "../enums/enums.h"

class Tokenizer
{
    int d_value = 0;

    public:
        OperandType token();
        void reset()            const;
        Opcode opcode()         const;
        int value()             const;
    private:
};

#endif
```

tokenizer/tokenizer.ih:

```
#include "tokenizer.h"
#include <string>
#include <iostream>
#include <limits>
using namespace std;
```

tokenizer/opcode.cc:

```
#include "tokenizer.ih"

```

```
Opcode Tokenizer::opcode() const
{
    string word;
    cin >> word;
    if (word.empty()) return Opcode::ERR;
    if (word == "mov") return Opcode::MOV;
    if (word == "add") return Opcode::ADD;
    if (word == "sub") return Opcode::SUB;
    if (word == "mul") return Opcode::MUL;
    if (word == "div") return Opcode::DIV;
    if (word == "neg") return Opcode::NEG;
    if (word == "dsp") return Opcode::DSP;
    if (word == "stop") return Opcode::STOP;
    return Opcode::ERR;
}
```

tokenizer/reset.cc:

```
#include "tokenizer.ih"

void Tokenizer::reset() const
{
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}
```

tokenizer/token.cc:

```
#include "tokenizer.ih"

OperandType Tokenizer::token()
{
    string word;
    cin >> word;

    if (word.empty()) return OperandType::SYNTAX;

    if (word[0] == '@')
    {
        d_value = stoi(word.erase(0,1));
        return OperandType::MEMORY;
    }
    if (word[0] <= 'z' && word[0] >= 'a' && word.size() == 1)
    {
        d_value = word[0] - '0';
        return OperandType::REGISTER;
    }
    if (word.find_first_not_of("0123456789") == string::npos)
    {
        d_value = stoi(word);
        return OperandType::VALUE;
    }

```

```
26      return OperandType::SYNTAX;
27  }
```

tokenizer/value.cc:

```
1   #include "tokenizer.ih"
2
3   int Tokenizer::value() const
4   {
5       return d_value;
6   }
```