Programming report
Week 3 Assignments C++

Jaime Betancor Valado
Christiaan Steenkist
Remco Bos
Diego Ribas Gomes

October 5, 2016

Assignment 21, Key Concepts

Encapsulation and data hiding

Encapsulation: Is the act of putting together all the data and functions as a class. Data hiding: In the making of the encapsulation you can try to implement the data hiding, i.e., making private the data members to not be accesible for the user or other functions outside the class.

Why are they important to designing classes?

How you design the interface of the class (how it is seen by the user) it is important. Furthermore, making data only accesible by member functions prevents some design errors. Another feature implemented by encapsulation is that you can modify data members without worrying to much of external functions since only member functions interacts with them.

Why only interface and not implementation?

To learn to use an object of a class you only need to learn how it is his interface declared. The user doesn't need to know how it is implemented, i.e., how it works, that is the beauty of the objects.

Code Listings

Listing 1: bignum.h

```
//Small example of self-defined class

class person
{
        //Member data
        std::string d_name;
        size_t d_weight;
        size_t d_height;
        size_t d_age;
    public://Constructor and member functions
        person();
        //Manipulators
        void setName(std::string const &name);
        void setWeight(size_t weight);
        void setHeight(size_t height);
        void setAge(size_t age);
        //Accessors
        std::string &name() const;
        size_t weight() const;
        size_t height() const;
        size_t age() const;
};
```

Assignment 22, Class member modifiers

We were tasked to ”implement” a class with several overloaded members, each of them with specific tasks. The implementation is not required, only the declaration.

Code Listings

Listing 2: header.h

```
#ifndef EX22_H_
#define EX22_H_

class Demo
{
    public:
        void run() &; // 1
```

```cpp
    void run() const &; // 2
    void run() &&; // 3
};

#endif
```

Listing 3: `inthead.ih`

```cpp
#include "ex22.h"
#include <iostream>

using namespace std;

void caller(Demo &demo);        // 1
void caller(Demo const &demo); // 2
void caller(Demo&& demo);       // 3
```

Listing 4: `main.cc`

```cpp
#include "ex22.ih"

int main()
{
    Demo demo;
    demo.run();
    caller(demo);         // 1

    const Demo constdemo;
    constdemo.run();
    caller(constdemo);    // 2

    Demo().run();
    caller(Demo());   // 3
}
```

Listing 5: `caller1.cc`

```cpp
#include "ex22.ih"

void caller(Demo &demo)
{
    demo.run();
}
```

Listing 6: `caller2.cc`

```
#include "ex22.ih"

void caller(Demo const &demo)
{
    demo.run();
}
```

Listing 7: `caller3.cc`

```
#include "ex22.ih"

void caller(Demo&& demo)
{
    Demo().run();
}
```

Assignment 23, CPU - MEMORY

This is the first of a series of exercises to design a CPU. In this one we are tasked to implement some enum elements from which all the classes will read and finally we need to implement the memory class.

Code Listings

Listing 8: `enum.h`

```
#ifndef INCLUDED_ENUM_H
#define INCLUDED_ENUM_H

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

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

enum RAM
{
    UNDEFINED = 0,
    SIZE = 20
};

#endif
```

Listing 9: `memory.h`

```
#ifndef INCLUDED_MEMORY_H_
#define INCLUDED_MEMORY_H_

#include <iostream>
#include "../enum.h"

struct Data
{
    signed int d_values[RAM::SIZE];
};

class Memory
{
    private:
        size_t const s_SIZE = RAM::SIZE;
        Data data;

    public:
        Memory();

        void store(size_t address,
            signed int value);

        signed int load(size_t address) const;
};

#endif
```

Listing 10: `inhead.ih`

```
#include "memory.h"
#include <iostream>
#include "../enum.h"

using namespace std;
```

Listing 11: `load.cc`

```
#include "memory.ih"

int Memory::load(size_t address) const
{
    return address < SIZE ? data.d_values[address] :
        UNDEFINED;
}
```

Listing 12: `constr1.cc`

```
#include "memory.ih"

Memory::Memory()
//:
{
    for (size_t index = 0; index != s_SIZE; ++index)
    {
        store(index, UNDEFINED);
    }
}
```

Listing 13: `store.cc`

```
#include "memory.ih"

void Memory::store(size_t address,
    int value)
{
    if (address < SIZE)
        data.d_values[address] = value;
}
```

Assignment 24, CPU - CPU

This is the second of a series of exercises to design a CPU. In this one we are tasked to implement the cpu class and in addition, to define main and start the cpu.

Code Listings

Listing 14: `cpu.h`

```
#ifndef EX24_H_
#define EX24_H_

#include "../memory/memory.h"
#include "../enum.h"

enum registers
{
    NREGISTERS = 5
};

class CPU
{
    struct Operand
    {
        OperandType type;
        int value;
    };

    char d_REGISTER[NREGISTERS];
    Memory d_memory;

    private:
        bool error();
        void mov();
        void add();
        void sub();
        void mul();
        void div();
        void neg();
        void dsp();
        void stop();

    public:
        CPU(Memory &memory);
        void start();
};

#endif
```

Listing 15: `inthead.ih`

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

using namespace std;
```

Listing 16: `main.cc`

```
#include "cpu.h"

int main()
{
    Memory memory;
    CPU cpu(memory);
    cpu.start();
}
```

Listing 17: `cpu.cc`

```
#include "cpu.h"

CPU::CPU(Memory &memory)
:
    d_memory(memory)
{
}
```

Listing 18: `error.cc`

```
#include "cpu.h"

CPU::error()
{
    cout << "syntax error\n";
    return false;
}
```

Listing 19: `start.cc`

```
#include "cpu.ih"

void CPU::start()
{
    while (true)
    {
        int opCode = Tokenizer.opcode();
        switch (opCode)
        {
            case (ERR):
                //return an error if invalid
                //instruction
                error();
            break;
            case (MOV):
                //value of rhs operand
                //asigned to lhs operand
                mov();
            break;
            case (ADD):
                //value of rhs operand
                //added to lhs operand
                add();
            break;
            case (SUB):
                //value of rhs operand
                //substracted from lhs operand
                sub();
            break;
```

```
                case (MUL):
                    //lhs operand is multiplied
                    //by the value of lhs operand
                    mul();
                break;
                case (DIV):
                    //lhs operand is divided
                    //by the value of lhs operand
                    div();
                break;
                case (NEG):
                    //value negated
                    neg();
                break;
                case (DSP):
                    //value inserted into cout
                    dsp();
                break;
                case (STOP):
                    //the program ends
                    stop();
                break;
            }
            Tokenizer.reset();
        }
}
```

Assignment 25, CPU - TOKENIZER

This is the third of a series of exercises to design a CPU. In this one we are tasked to implement the tokenizer class.

Code Listings

Listing 20: `tokenizer.h`

```
#ifndef TOKENIZER_H_
#define TOKENIZER_H_

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

```
class Tokenizer
{
    Tokenizer();
    static int d_value;

    public:

    static void reset();
    static int value();
    static Opcode opcode();
    static OperandType token();

};

#endif
```

Listing 21: `tokenizer.ih`

```
#include "tokenizer.h"
#include <string>
#include <iostream>

using namespace std;
```

Listing 22: `opcode.cc`

```
#include "tokenizer.ih"
#include "../enums/enums.h"

Opcode Tokenizer::opcode()
{
    string input;
    cin >> input;

    if (input.compare("mov") == 0)
        return MOV;
    if (input.compare("add") == 0)
        return ADD;
    if (input.compare("sub") == 0)
        return SUB;
    if (input.compare("mul") == 0)
```

```
        return MUL;
    if (input.compare("div") == 0)
        return DIV;
    if (input.compare("neg") == 0)
        return NEG;
    if (input.compare("dsp") == 0)
        return DSP;
    if (input.compare("stop") == 0)
        return STOP;
    return ERR;
}
```

Listing 23: `reset.cc`

```
#include "tokenizer.ih"

void Tokenizer::reset()
{
    cin.clear();

    string input;
    getline(cin, input);
}
```

Listing 24: `token.cc`

```
#include "tokenizer.ih"
#include "../enums/enums.h"

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

    char firstChar = input[0];
    if (firstChar >= 'a' && firstChar <= 'z' && input.
   length() == 2)
    {
        d_value = firstChar - 'a';
        return REGISTER;
    }
    OperandType output = SYNTAX;
    if (firstChar == '-' || (firstChar >= '0' &&
   firstChar <= '9'))
        output = VALUE;
    if (firstChar == '@')
        output = MEMORY;
    if (output == SYNTAX)
        return output;

    for (int index = 1; index != input.length(); ++
   index)
    {
        if (input[index] == 0)
            break;
        if (input[index] < '0' || input[index] > '9')
            return SYNTAX;
    }
    if (output == MEMORY)
    {
        string address = input.substr(1, input.length
   () - 1);
        d_value = stoi(address);
    }
    else
        d_value = stoi(input);
    return output;
}
```

Listing 25: `tokenizer1.cc`

```
#include "tokenizer.ih"

Tokenizer::Tokenizer()
//:
{
}
```

Listing 26: `value.cc`

```
#include "tokenizer.ih"
```

```
int Tokenizer::value()
{
    return d_value;
}
```