Programming report
Week 2 Assignments C++

Jaime Betancor Valado
Christiaan Steenkist
Remco Bos
Diego Ribas Gomes

September 30, 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 design 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.

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
{
    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;
    private://Member data
        std::string d_name;
        size_t d_weight;
        size_t d_height;
        size_t d_age;
};
```

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:
        Demo();
        void run();        // 1
        void run() const;  // 2
        void run() &&;     // 3
};

#endif
```

Listing 3: `inthead.ih`

```
#include "ex22.h"

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

Listing 4: `main.cc`

```
#include "ex22.ih"

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

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

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

Listing 5: `caller1.cc`

```
#include "ex22.ih"

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

Listing 6: `caller2.cc`

```
#include "ex22.ih"
```

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

Listing 7: caller3.cc

```
#include "ex22.ih"

void caller(Demo d&&)
{
    d.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
{
    SIZE = 20
};

#endif
```

Listing 9: `memory.h`

```
#ifndef INCLUDED_MEMORY_H_
#define INCLUDED_MEMORY_H_

#include <iostream>

class Memory
{
    private:
        size_t const s_SIZE = 20;

        unsigned int d_values[];

    public:
        Memory();
        Memory(unsigned int values[]);

        void store(size_t address,
            unsigned int value);

        unsigned int const load(size_t address) const;
};

#endif
```

Listing 10: `inhead.ih`

```
#include <iostream>

using namespace std;
```

Listing 11: `load.cc`

```
#include "memory.ih"

unsigned int const Memory::load(size_t address) const;
{
    if (address >= 0 && address < SIZE)
        return d_values[address];
    return UNDEFINED;
}
```

Listing 12: `constr1.cc`

```
#include "memory.ih"

Memory::Memory()
:
    Memory::Memory(unsigned int values[] {UNDEFINED})
{
}
```

Listing 13: `constr2.cc`

```
#include "memory.ih"

Memory::Memory(unsigned int values[])
:
    d_values(values)
{
}
```

Listing 14: `store.cc`

```
#include "memory.ih"

void Memory::store(size_t address,
    unsigned int value);
{
    if (address >= 0 && address < SIZE)
        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 15: `cpu.h`

```
#ifndef EX24_H_
#define EX24_H_

#include "enum.h"

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

    int const d_NREGISTERS = 5;
    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 16: `inthead.ih`

```
#include <iostream>

using namespace std;
```

Listing 17: `main.cc`

```
#include "cpu.h"

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

Listing 18: `cpu.cc`

```
#include "cpu.h"

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

Listing 19: `error.cc`

```
#include "cpu.h"

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

Listing 20: `start.cc`

```
#include "cpu.h"

CPU::start()
{
    while (true)
    {
        int opCode = Tokenizer.opcode();
        switch (opCode)
        {
            case (ERR):
                error();
                Tokenizer.reset();
                break;
            case (MOV):
                mov();
                Tokenizer.reset();
                break;
            case (ADD):
                add();
                Tokenizer.reset();
                break;
            case (SUB):
                sub();
                Tokenizer.reset();
                break;
            case (MUL):
                mul();
                Tokenizer.reset();
                break;
            case (DIV):
                div();
                Tokenizer.reset();
                break;
            case (NEG):
                neg();
                Tokenizer.reset();
                break;
            case (DSP):
                dsp();
                Tokenizer.reset();
                break;
            case (STOP)
                stop();
                Tokenizer.reset();
                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 21: `tokenizer.h`

```
#ifndef INCLUDED_TOKENIZER_
#define INCLUDED_TOKENIZER_

#include "enum.h"

class tokenizer
{

    public:

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

    private:

    int d_value;

};
```

```
#endif
```

Listing 22: `tokenizer.ih`

```
#include "tokenizer.h"

using namespace std;
```

Listing 23: `opcode.cc`

```
#include "tokenizer.ih"
#include "enum.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 24: `reset.cc`

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

```
void tokenizer::reset()
{
    cin.clear();

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

Listing 25: `token.cc`

```
#include "tokenizer.ih"
#include "enum.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.subtr(1,
            input.length() - 1);
        d_value = stoi(address);
    else
        d_value = stoi(input);
    return output;
}
```

Listing 26: `tokenizer1.cc`

```
#include "tokenizer.ih"

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

Listing 27: `value.cc`

```
#include "tokenizer.ih"

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