Exercise Set 4

Steffen Beudeker s2503166

Exercise 26

Data hiding is the practice of protecting the integrity of your class. In the example the class contains two coordinates, members to set the coordinates, and a member to find the distance between two points. The members "set_x()" and "set_y" are there to make sure that the values are actually correct. If the data was instead accessed directly, someone using the class could perhaps input a character instead of a number, and then the entire class would break down. Distance from (a,1) to (2,3) cannot be computed.

Encapsulation is a bigger definition and data hiding is a form of encapsulation as well. The reason for this practice is so that your code is modular. For example, if your class handles the storage method instead of the user, then it is easier to switch. This makes the code more adaptable without having to alter everything. In the example the values are stored in two integer type variables, but if the program were to store them in an array instead, the members can be easily modified to accommodate this change, while not having to change all code dependant on the class.

```
class Coordinate
{
     int d_x_coord;
     int d_y_coord;

     public:
          Coordinate();
          set_x();
          set_y();
          size_t distance() const;
};
```

Showing the implementation is not relevant, because the essence of encapsulation is that your code is adaptable. So many different implementations can be used while using the same interface.

Exercise 27

Disclaimer: in user1.cc and main.cc there is a magic number 501. That happens to be my uid. I chose to use a magic number here because it is easier and because it is not the point of the exercise. It would have been possible to do it differently but that would have required unnecessary work. Please understand.

user.h

```
#ifndef INCLUDED_USER_
#define INCLUDED_USER_

#include <string>

class User
{
     struct passwd* pw_entry;
     //this is true if the constructor has worked correctly:
     bool d_constructed = false;

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


    private:
};

#endif
```

user.ih

```
#include "user.h"
#include <sys/types.h>
#include <pwd.h>

using namespace std;
```

groupId.cc
```
#include "user.ih"

size_t User::groupId() const
{
     return pw_entry -> pw_gid;
}
```

homeDir.cc
```
#include "user.ih"

string User::homeDir() const
{
     string directory = pw_entry -> pw_dir;
     return directory.append("/");
}
```

inGroup.cc
```
#include "user.ih"

bool User::inGroup(size_t gid) const
{
     return (pw_entry -> pw_gid == gid ? true : false);
}
```

name.cc
```
#include "user.ih"

string User::name() const
{
     return pw_entry -> pw_name;
}
```

realName.cc
```
#include "user.ih"

string User::realName() const
{
     return pw_entry -> pw_gecos;
}
```

shell.cc
```
#include "user.ih"

string User::shell() const
{
     return pw_entry -> pw_shell;
}
```

user1.cc
```
#include "user.ih"

User::User()
//:
```

```
{
     pw_entry = getpwuid(501);
     d_constructed = true;
}
```

user2.cc

```
#include "user.ih"

User::User(size_t userId)
//:
{
     pw_entry = getpwuid(userId);
     d_constructed = true;
}
```

userId.cc

```
#include "user.ih"

size_t User::userId() const
{
     return pw_entry -> pw_uid;
}
```

valid.cc

```
#include "user.ih"

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

main.cc

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

using namespace std;


int main(int argc, char const *argv[])
{
     User Steffen(501);
     Steffen.groupId();
     Steffen.homeDir();
     Steffen.name();
     Steffen.realName();
     Steffen.shell();
     Steffen.userId();
     Steffen.valid();
}
```

Exercise 28

enums.h

```
#ifndef ENUMS_H_
#define ENUMS_H_

enum RAM
{
	SIZE = 20
};

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

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

#endif
```

memory.h
```
#ifndef INCLUDED_MEMORY_
#define INCLUDED_MEMORY_

#include <stdlib.h> //Included to use 'size_t'
#include "enums.h"

class Memory
{
     int d_memory[RAM::SIZE];

    public:
        Memory();
        void store(size_t const address, int const &value);
        int load(size_t const address) const;

    private:
};

#endif
```

memory.ih
```
#include "memory.h"

using namespace std;
```

memory1.cc
```
#include "memory.ih"

Memory::Memory()
{}
```

memory_load.cc
```
#include "memory.ih"

int Memory::load(size_t address) const
{
     if (address > RAM::SIZE)
          return 0;
     else
          return d_memory[address];
}
```

memory_store.cc
```
#include "memory.ih"

void Memory::store(size_t const address, int const &value)
{
     d_memory[address] = value;
}
```

Exercise 29

cpu.h
```
#ifndef INCLUDED_CPU_
#define INCLUDED CPU
```

```
//To access member functions the class must be defined
#include "memory.h"

//Defined here because its the only class using it
enum NREGISTERS
{
     NREGISTERS = 5
};

struct Operand
{
     OperandType type;
     int value;
};

class CPU
{
     Memory d_memory;
     int d_registers [NREGISTERS];
    public:
        CPU();
        CPU(Memory memory);
        void start();
    private:
     bool error();
};

#endif
```

cpu.ih

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

using namespace std;
```

cpu1.cc

```
#include "cpu.ih"

CPU::CPU()
//:this constructor does not look for an existing memory
{

}
```

cpu2.cc

```
#include "cpu.ih"

CPU::CPU(Memory memory)
//:this constructor makes the passed memory his own
{
     d_memory = memory;
}
```

cpu_error.cc

```
#include "cpu.ih"

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

cpu_start.cc

```
#include "cpu.ih"

void CPU::start()
{

     while (true)
     {
          switch (Tokenizer::opcode())
          {
               case OpCode::ERR:
                    CPU::error();
                    break;

               case OpCode::MOV:
                    CPU::mov();
                    break;

               case OpCode::ADD:
                    CPU::add();
                    break;

               case OpCode::SUB:
                    CPU::sub();
                    break;

               case OpCode::MUL:
                    CPU::mul();
                    break;

               case OpCode::DIV:
                    CPU::div();
                    break;

               case OpCode::NEG:
                    CPU::neg();
                    break;

               case OpCode::DSP:
                    CPU::dsp();
                    break;

               case OpCode::STOP:
                    return;
          }
          Tokenizer::reset();
```

```
    }
}
```

main.cc

```
//while some headers are already included in other headers due
//to dependancies, I write them here again for clarity so that
//it is immediately clear which headers are used. The include //guards
make sure there is no extra work

#include "enums.h"
#include "memory.h"
#include "cpu.h"

using namespace std;

int main(int argc, char* argv[])
{
    Memory memory;
    CPU cpu(memory);
    cpu.start();
    return 0;
}
```

Exercise 30

tokenizer.h

```
#ifndef INCLUDED_TOKENIZER_
#define INCLUDED_TOKENIZER_

#include "cpu.h"

class Tokenizer
{
    public:
        Tokenizer();
        void reset();
        OpCode opcode();
        Operand token();
    private:
};

#endif
```

tokenizer.ih

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

using namespace std;
```

tokenizer1.cc

```
#include "tokenizer.ih"

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

tokenizer_reset.cc

```
#include "tokenizer.ih"
#include <limits> //for infinite cin.ignore()

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

tokenizer_opcode.cc

```
#include "tokenizer.ih"

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

     switch (input[2])
     {
```

```
        case 'v':
            return (input == "mov" ? MOV :
                    input == "div" ? DIV : ERR);

        case 'd':
            return (input == "add" ? ADD : ERR);

        case 'b':
            return (input == "sub" ? SUB : ERR);

        case 'l':
            return (input == "mul" ? MUL : ERR);

        case 'g':
            return (input == "neg" ? NEG : ERR);

        case 'p':
            return (input == "dsp" ? DSP : ERR);

        case 'o':
            return (input == "top" ? STOP : ERR);

        default:
            return ERR;

    }
}
```

tokenizer token.cc

I could not get this to work correctly. I put it in regardless. Maybe you have something interesting to say, but I seriously doubt it. I understand if you don't read this part

```
#include "tokenizer.ih"

Operand Tokenizer::token()
{
    Operand operand;
    string input;
    cin >> input;

    if (input.find_first_not_of("0123456789") == std::string::npos)
    {
        operand.type = VALUE;
        operand.value = input;
        return operand;
    }


    if (input[0] == '@')
    {
        size_t pos;
        if (input[1] == '-')
            pos = 2;
        else
```

```
                pos = 1;
          if (input.find_first_not_of("0123456789", pos) ==
std::string::npos)
          {
                operand.type = MEMORY;
                operand.value = input.substr(input.length() - 1);
                return operand;
          }

     }

     if (isalpha(input[0]) && input.length() == 1)
     {
          operand.type = REGISTER;
          operand.value = input;
          return operand;
     }


     operand.type = SYNTAX;
     return operand;

}
```