Regular: Yuqin Cui (s3215040)
Regular: Jaap van der Leest
Attendee: Timon van der Berg (s1925873)
October 3, 2017

C++ Course
Assignment 3

Exercise 18

Solution.

- It is appropriate to use an int-type parameter when only whole numbers are used. In the code example this is illustrated. The price of pizzas is calculated based on the number of pizzas ordered.

```
    int nrPizzas = 7;
    cout << "Give the pizzadeliverguy: " << nrPizzas * 4.99 << " euros.\n";
```

- It is appropriate to use an std::string value parameter when pieces of text need to be stored. In the code example this is illustrated because some text is predefined and later shown.

```
    std::string welcome = "Hello to all of you!";
    cout << welcome;
```

- It is appropriate to use a const reference to an int-type parameter... Well it is not really appropriate to define a const reference. Because a reference to a variable is always a const and the compiler won't accept it either.

- It is appropriate to use a const reference to a std::string value parameter... Well it is not really appropriate to define a const reference. Because a reference to a variable is always a const and the compiler won't accept it either.

- It is appropriate to use a non-const reference to an int-type parameter if we want to make a reference to an int-type parameter. A reference is always constant so no need to declare const or non const. A reference to an int-type parameter would make sense if we want to pass the value of the int-type parameter to a function but not copy it. This can be for reasons of e.g. memory or speed, or if we want to access the int-type parameter (which lives somewhere else locally) from within the function.

```
void ShowNumber(int &number)
{
    cout << number << '\n';
}

int main()
{
    int value = 5;
    ShowNumber(value);
}
```

- It is appropriate to use a non-const reference to a std::string value parameter if we want to make a reference to a std::string value parameter. A reference is always constant so no need to declare const or non const. A reference to a std::string value parameter would make sense if we want to pass the value of the std::string value parameter to a function but not copy it. This can be for reasons of e.g. memory or speed, or if we want to access the std::string value parameter (which lives somewhere else locally) from within the function.

```
void ShowText(std::string &text)
{
    cout << text << '\n';
}

int main()
{
    std::string string = "Hello world!";
```

```
    ShowText(string);
}
```

- It is appropriate to use a const rvalue reference to a int type parameter if... no. A const rvalue is nonsense. a rvalue by definition is temporary and will cease to exist after it is used.
- It is appropriate to use a const rvalue reference to a std::string parameter if... no. A const rvalue is nonsense. a rvalue by definition is temporary and will cease to exist after it is used.
- It is appropriate to use a rvalue reference to a int type parameter if we need to use (and probably modify) an int only within a function which value is passed to it when calling the function. It ceases to exist after the function ends.

```
void myFun(int &&number)
```

- It is appropriate to use a rvalue reference to a std::string parameter if we need to use (and probably modify) a string only within a function which value is passed to it when calling the function. It ceases to exist after the function ends.

```
void myFun(string &&myString)
```

- It is appropriate to return an int-type value if a function returns a whole number.

```
int multiply(int first, int second)
{
    return (first * second);
}
```

- It is appropriate to return a std::string value if a function returns a piece of text.

```
std::string helloWorld(void)
{
    std::string hello = "Hello World!\n";
    return(hello);
}
```

- It is not appropriate to return something like a reference or rvalue reference (const or non-const) Because the values being returned are not accessible anymore when the function ends.

Exercise 19

makefile:

```
exercise_19 : exercise_19.o method1.o method2.o method3.o method4.o \
            method5.o method6.o
                g++ -std=c++17 exercise_19.o method1.o method2.o method3.o \
                method4.o method5.o method6.o -o exercise_19

exercise_19.o : exercise_19.cc myheader.ih
    g++ -std=c++17 -Wall -Werror -c exercise_19.cc
method1.o : method1.cc myheader.ih
    g++ -std=c++17 -Wall -Werror -c method1.cc
method2.o : method2.cc myheader.ih
    g++ -std=c++17 -Wall -Werror -c method2.cc
method3.o : method3.cc myheader.ih
    g++ -std=c++17 -Wall -Werror -c method3.cc
method4.o : method4.cc myheader.ih
    g++ -std=c++17 -Wall -Werror -c method4.cc
method5.o : method5.cc myheader.ih
    g++ -std=c++17 -Wall -Werror -c method5.cc
method6.o : method6.cc myheader.ih
    g++ -std=c++17 -Wall -Werror -c method6.cc
```

```
clean :
        rm exercise_19 exercise_19.o method1.o method2.o method3.o method4.o \
            method5.o method6.o
```

myheader.ih:

```
// myheader.ih
#include <iostream>
#include <sstream>
#include <math.h>

    #ifdef __cplusplus
        extern "C" {
    #endif

    void method1(unsigned long long int const valueToAnalyze, int nrOfTurns);
        // MSB right shift bits
    void method2(unsigned long long int const valueToAnalyze, int nrOfTurns);
        // MSB logarithm
    void method3(unsigned long long int const valueToAnalyze, int nrOfTurns);
        // MSB bit boundary search
    void method4(unsigned long long int const valueToAnalyze, int nrOfTurns);
        // LSB right shift bits
    void method5(unsigned long long int const valueToAnalyze, int nrOfTurns);
        // LSB logarithm
    void method6(unsigned long long int const valueToAnalyze, int nrOfTurns);
        // LSB bit boundary search

    extern double const ln2;

    #ifdef __cplusplus
        }
    #endif
```

exercise_19.cc:

```
// exercise_19.cc
#include "myheader.ih"

using namespace std;


int main(int argc, char* argv[])
{
    istringstream iSS(argv[1]);
        // need istringstream here. stoi can't handle long long ints

    unsigned long long int valueToAnalyze;
    iSS >> valueToAnalyze;

    size_t method = stoi(argv[2]);
    size_t nrOfTurns = 1;

    if (argc > 3)                   // if provided set nr of turns to calculate
        nrOfTurns = stoi(argv[3]);

    if (method == 1)
        method1(valueToAnalyze, nrOfTurns);
    else if (method == 2)
```

```
        method2(valueToAnalyze, nrOfTurns);
    else if (method == 3)
        method3(valueToAnalyze, nrOfTurns);
    else if (method == 4)
        method4(valueToAnalyze, nrOfTurns);
    else if (method == 5)
        method5(valueToAnalyze, nrOfTurns);
    else if (method == 6)
        method6(valueToAnalyze, nrOfTurns);
    else
        cout << "Please provide the second argument as a number 1 - 6.\n";

}

/*
The fastest method for finding the MSB depends on the input.
Method 1 is quick for small nrs. But big nrs can request up to 64 calculations.
Method 3 is relative fast when calculating big nrs. This method takes a maximum
of 6 calculations.
Method 2 seems to have a constant calc speed which is never the fastest.
*/
```

method1.cc:

```
// method1.cc
#include "myheader.ih"

using namespace std;


void method1(unsigned long long int const valueToAnalyze, int nrOfTurns)
{
    size_t counter = 0;

    while (nrOfTurns != 0)
    {
        unsigned long long int valueToCompute = valueToAnalyze;
        counter = 0;
        while (valueToCompute != 0)
        {
            valueToCompute >>= 1;    // right shift bits
            ++counter;
        }

        counter -= 1;            // calc offset not the nr of bits
        --nrOfTurns;
    }

    cout << "Method 1 right shift bits.\n";
    cout << "MSbit of " << valueToAnalyze << " is at bit offset "
        << counter << '\n';

    return;
}
```

method2.cc:

```
// method2.cc
#include "myheader.ih"
```

```

using namespace std;


double const ln2 = log (2);

void method2(unsigned long long int const valueToAnalyze, int nrOfTurns)
{
    size_t counter = 0;

    while (nrOfTurns != 0)
    {
        unsigned long long int valueToCompute = valueToAnalyze;

        counter = log (valueToCompute) / ln2;  // ln2 lives in ln2.cc

        --nrOfTurns;
    }

    cout << "Method 2 logarithm.\n";
    cout << "MSbit of " << valueToAnalyze << " is at bit offset "
        << counter << '\n';

    return;
}
```

method3.cc:

```
// method3.cc
#include "myheader.ih"

using namespace std;


void method3(unsigned long long int const valueToAnalyze, int nrOfTurns)
{
    size_t counter = 0;

    while (nrOfTurns != 0)
    {
        unsigned long long int valueToCompute = valueToAnalyze;
        size_t ttLow = 0;
        size_t ttHigh = sizeof (valueToCompute) * 8; // nr of bytes * 8 bits
        size_t ttMid;

        ttMid = (ttLow + ttHigh) / 2;

        while (true)
        {
            valueToCompute = valueToAnalyze;

                // compute if all bits are before ttMid
            size_t shiftedValue = valueToCompute >>= ttMid;
            if (shiftedValue == 0)
            {
                ttHigh = ttMid;
                ttMid = (ttLow + ttHigh) / 2;
            }

            else if (ttLow == ttMid)
```

```
            {
                counter = ttMid;
                break;
            }
            else
            {
                ttLow = ttMid;
                ttMid = (ttLow + ttHigh) / 2;
            }
        }

        --nrOfTurns;
    }

    cout << "Method 3 binary search.\n";
    cout << "MSbit of " << valueToAnalyze << " is at bit offset "
        << counter << '\n';

    return;
}
```

method4.cc:

```
// method4.cc
#include "myheader.ih"

using namespace std;


void method4(unsigned long long int const valueToAnalyze, int nrOfTurns)
{
    size_t counter = 0;

    while (nrOfTurns != 0)
    {
        unsigned long long int valueToCompute = valueToAnalyze;
        counter = 0;

        while ((valueToCompute & 1) != 1)
        {
            valueToCompute >>= 1;    // right shift bits
            ++counter;
        }

        --nrOfTurns;
    }

    cout << "Method 4 right shift bits.\n";
    cout << "LSbit of " << valueToAnalyze << " is at bit offset "
        << counter << '\n';

    return;
}
```

method5.cc:

```
// method5.cc
#include "myheader.ih"

```

```
using namespace std;


void method5(unsigned long long int const valueToAnalyze, int nrOfTurns)
{

    while (nrOfTurns != 0)
    {
        // doing nothing as many times as you ask
        --nrOfTurns;
    }

    cout << "Method 5 logarithm.\n";
    cout << "I don't think it is possible to calculate the LSbit with "
        << "logarithm \n";

    return;
}
```

method6.cc:

```
// method6.cc
#include "myheader.ih"

using namespace std;


void method6(unsigned long long int const valueToAnalyze, int nrOfTurns)
{
    size_t counter = 0;
    while (nrOfTurns != 0)
    {
        unsigned long long int valueToCompute = valueToAnalyze;
        size_t ttLow = sizeof (valueToCompute) * 8; // nr of bytes * 8 bits
        size_t ttHigh = 0;
            // ttLow and ttHigh values are swapped for LSB calc

        size_t ttMid;
        ttMid = (ttLow + ttHigh) / 2;

        while (true)
        {
            valueToCompute = valueToAnalyze;

                // compute if last bit is before ttMid
            size_t shiftedValue = valueToCompute <<= ttMid;
            if (shiftedValue == 0)
            {
                ttLow = ttMid;
                ttMid = (ttLow + ttHigh) / 2;
            }

            else if (ttHigh == ttMid)
            {
                counter = sizeof (valueToAnalyze) * 8 - (ttMid + 1);
                break;
            }
            else
            {
```

```
                ttHigh = ttMid;
                ttMid = (ttLow + ttHigh) / 2;
            }
        }

        --nrOfTurns;
    }

    cout << "Method 6 binary search.\n";
    cout << "LSbit of " << valueToAnalyze << " is at bit offset "
        << counter << '\n';

    return;
}
```

Exercise 20

Problem statement. Bla
Solution. head.ih:

```
#include <unistd.h>         // isatty
#include <iostream>         // cin, cout
#include <getopt.h>         // getopt_long

// processing type
enum class Mode {
    ERROR,
    CAPITALIZE,
    LOWER_CASE,
    VERSION,
    USAGE
};

// arguments type
struct vars_t {
    bool help;              // -h --help
    bool version;           // -v --version
    bool capitalize;        // -c --uc --capitalize
    bool lowercase;         // -l --lc --lower-case
};

// info for user
void usage();

// process input
void process(vars_t Vars);

// do stuff
vars_t arguments(int argc, char* argv[]);

// select mode from arguments
Mode selectOpt(vars_t Vars);

// cout version num
void version();
```

main.cc:

```
#include "head.ih"

int succesState = 0;                    // expections NC

int main(int argc, char* argv[])
{
    if (isatty(0))
    {
        std::cout << "no file redirection" << '\n';
        return 1;
    }
    process(arguments(argc,argv));
    return succesState;
}
```

arguments.cc:

```
#include "head.ih"

// long options and short options
struct option longOpts[] =
{
    {"capitalize",  0,    0, 'c'},
    {"uc",          0,    0, 'c'},
    {"lowercase",   0,    0, 'l'},
    {"lc",          0,    0, 'l'},
    {"version",     0,    0, 'v'},
    {"help",        0,    0, 'h'},
    { 0 }
};

vars_t arguments(int argc, char* argv[])
{
    vars_t Vars = {false, false, false, false};
    int opt;
    while ((opt = getopt_long(argc, argv, "hvcl", longOpts, &opt)) != -1)
        switch (opt)
        {
            case 'h':                   // help
            {
                Vars.help = true;
                break;
            }
            case 'v':                   // version
            {
                Vars.version = true;
                break;
            }
            case 'c':                   // capitalize
            {
                Vars.capitalize = true;
                break;
            }
            case 'l':                   // lower-case
            {
                Vars.lowercase = true;
                break;
            }
            default:
```

```
            {
                Vars.help = true;
                break;
            }
        }
    return Vars;
}
```

process.cc:

```
#include "head.ih"
#include <ctype.h>          // toupper, tolower

void process(vars_t Vars)
{
    Mode option = selectOpt(Vars);
    switch (option)
    {
        case (Mode::ERROR):
        {
            std::cout << "ERROR" << '\n';                          // is this enough?
            break;
        }
        case (Mode::USAGE):
        {
            usage();
            break;
        }
        case (Mode::VERSION):
        {
            version();
            break;
        }
        case (Mode::CAPITALIZE):
        {
            char ch;
            while (std::cin.get(ch)) std::cout << static_cast<char>(toupper(ch));
            break;
        }
        case (Mode::LOWER_CASE):
        {
            char ch;
            while (std::cin.get(ch)) std::cout << static_cast<char>(tolower(ch));
            break;
        }
    }
    return;
}
```

selectopt.cc:

```
#include "head.ih"
extern int succesState;

Mode selectOpt(vars_t Vars)
{
    if (Vars.help)
        return Mode::USAGE;
    if (Vars.version)
```

```
        return Mode::VERSION;
    if (Vars.capitalize and Vars.lowercase)         // can't do both
    {
        return Mode::ERROR;
        succesState = 1;
    }
    if (Vars.capitalize)
        return Mode::CAPITALIZE;
    if (Vars.lowercase)
        return Mode::LOWER_CASE;
    std::cout << "Invalid argument provided.";
        succesState = 1;
    return Mode::ERROR;

}
```

usage.cc:

```
// instructions for users
#include "head.ih"

char const use[]=
R"(
20 V 1>

Usage: ./main [options] < file
Where:
    --captitalize   (--uc, -u);     captitalize the letters in 'file'
    --help          (-h);           display this information
    --lowercase     (--lc, -l);     convert letters to lowercase in 'file'
    --version       (-v);           display version information

20 processes 'file' and writes the results
to the standard output stream.
)";
void usage()
{
    std::cout << use << '\n';
}
```

version.cc:

```
#include "head.ih"

void version(){
    std::cout << "Version 1.45.12c.EY RC 5" << '\n';
}
```

Exercise 21

Problem statement.
Solution. myheader.ih:

```
// myheader.ih
#include <iostream>
#include <string>
```

```
4
5   using namespace std;
6
7   // calculate the next block
8   void square(string input, string leftFactor, string rest);
9
10  // a partial calculation of a block
11  int partial(string firstBlock, string leftFactor, string& rest);
```

main.cc:

```
1   // main.cc
2   #include "myheader.ih"
3
4   using namespace std;
5
6   int main (int argc, char* argv[])
7   {
8       string nrToCalc = argv[1];
9       square(nrToCalc, "", "");   // left factor and rest are 0 at start
10  }
```

partial.cc:

```
1   // partial.cc
2   #include "myheader.ih"
3
4   using namespace std;
5
6   int partial(string firstBlock, string leftFactor, string& rest)
7   {
8       string newNr = "";
9       int digit = 9;
10
11      while(true)
12      {
13          newNr = leftFactor + to_string(digit);
14          if (stoi(newNr) * digit <= stoi(firstBlock))
15              break;
16          --digit;
17      }
18
19      rest = to_string(stoi(firstBlock) - stoi(newNr) * digit);
20      cout << digit;
21      return(digit);
22  }
```

square.cc

```
1   // square.cc
2   #include "myheader.ih"
3
4   using namespace std;
5
6   void square(string input, string leftFactor, string rest)
7   {
8       string firstBlock = "";
9       size_t length = input.length();
10
```

```
    if (length == 0)
    {
        cout << '\n';
        return;
    }

    if (length % 2 == 0)
    {
        firstBlock = input.substr(0,2);  // take first 2 digits to calculate
        input.erase(0, 2);
    }
    else
    {
        firstBlock = input.substr(0,1);  // take only the first digit
        input.erase(0, 1);
    }

    firstBlock = rest + firstBlock;
    string& remainder = rest;            // this reference can access rest from
                                         // within the function partial()

    int digit = 0;
    digit = partial(firstBlock, leftFactor, remainder);
    leftFactor = to_string(stoi(leftFactor + to_string(digit)) + digit);

    square(input, leftFactor, rest);    // the function calls itself to
                                        // calculate the rest
    return;
}
```

Exercise 22

Problem statement.
Solution. test.sh (a small shell script to test the program)

```
#!/bin/bash

./main -d < input.txt >> output.txt
./main -e < output.txt >> return.txt

echo "INPUT=============="
cat input.txt
echo "OUTPUT============="
cat output.txt
echo "INPUT?============="
cat return.txt

rm output.txt return.txt
```

header.ih:

```
#include <string>
#include <iostream>

// info for users
void usage(std::string const &programName);
```

```

// command line options
enum class EOption {ENCODE,
                    DECODE,
                    NONE};

// which command line option was provided
EOption getOpt(std::string opt);

// convert 2-digit hex to decimal
size_t hexToDec(std::string str);

// convert decimal to 2-digit hex
std::string decToHex(size_t num);

// test if ch is alphanumerical
bool isAlpha(char ch);

// test if ch is ~ . - _
bool isOther(char ch);

// url-encode the stream
void encode(std::istream &is, std::ostream &os);

// decode url-encoded stream
void decode(std::istream &is, std::ostream &os);
```

main.cc

```
#include "header.ih"

int main(int argc, char* argv[])
{
    std::string arg = (argc == 2 ? argv[1] : "");    // read if possible

    switch (getOpt(arg))
    {
        case EOption::ENCODE:
            encode(std::cin, std::cout);
            break;
        case EOption::DECODE:
            decode(std::cin, std::cout);
            break;
        default:
            usage(argv[0]);
            int FAIL = 1;
            return FAIL;
    }
}

// Exercise 22: an URL stream decoder / encoder
//
//         Usage: main [-e/-d] < input.txt
//             (or provide stdin in other way)
//         Where:
//             -e        url-encode input
//             -d        decode url-encoded input
//         input.txt
//             contains either
//                 an url-encoded string (when using -d)
```

```
//                  an url-decoded string (when using -e)
```

decode.cc

```
#include "header.ih"
// decode url-encoded stream

void decode(std::istream &is, std::ostream &os)
{
    char ch;
    while (is.get(ch))
    {
        if (ch == '%' && is.get(ch))
        {
            std::string str;
            str.push_back(ch);            // why these str methods
            if (is.get(ch))
                str.append(1, ch);
            os << static_cast<char>(hexToDec(str));
        }
        else
        os << ch;                               // skip
    }
    return;
}

Jaap: Line 9 combined the 2 IF statement into 1
```

dectohex.cc

```
#include "header.ih"
// convert decimal to 2-digit hex

std::string decToHex(size_t num)
{
    size_t radix = 16;
    std::string buff;
    while (num != 0)                                        // process digits
    {                                                       // in reverse
        size_t remainder = num % radix;
        if (remainder > 9)
            buff.insert(0, 1, 'A' + remainder - 10);        // letter
        else
            buff.insert(0, 1, '0' + remainder);             // number
        num /= radix;
    }
    if (buff.length() != 2) buff.insert(0,1,'0');           // trailing 0
    return buff;
}
```

encode.cc:

```
#include "header.ih"
// url-encode stream

void encode(std::istream &is, std::ostream &os)
{
    char ch;
    while (is.get(ch))
```

```
        if (isAlpha(ch) or isOther(ch))                 // skip
            os << ch;
        else
            os << '%' << decToHex((size_t) ch);         // encode
    return;
}
```

getopt.cc:

```
#include "header.ih"
// determine which option was provided

EOption getOpt(std::string opt)
{
    if (opt == "-e") return EOption::ENCODE;
    if (opt == "-d") return EOption::DECODE;

    return EOption::NONE;
}
```

hextodec.cc

```
#include "header.ih"
// convert 2-digit hex to decimal

size_t hexToDec(std::string str)
{
    std::string hexDigits = "0123456789ABCDEF";
    return hexDigits.find(str[0]) * 16 + hexDigits.find(str[1]);   // hex base 16

}
```

isalpha.cc

```
#include "header.ih"
// test if ch is alphanumerical

bool isAlpha(char ch)
{
    if (ch >= '0' && ch <= '9')
        return true;
    if (ch >= 'A' && ch <= 'Z')
        return true;
    if (ch >= 'a' && ch <= 'z')
        return true;
    return false;
}
```

isother.cc

```
#include "header.ih"
// test if ch is part of exlude set

bool isOther(char ch)
{
    std::string others = "-_.~";                        // to skip
    return others.find(ch) != std::string::npos;
}
```

usage.cc

```
#include "header.ih"
// instructions for users

char const use[]=
R"(
Exercise 22: an URL stream decoder / encoder

Usage: main [-e/-d] < input.txt
    (or provide stdin in other way)
Where:
    -e        url-encode input
    -d        decode url-encoded input
input.txt
    contains either
        an url-encoded string (when using -d)
        an url-decoded string (when using -e)
)";

void usage(std::string const &programName)
{
    std::cout << use << '\n';
}
```

Exercise 23

Problem statement.
Solution. header.ih:

```
#include <iostream>
#include <string>   // string, to_string

// insert value, keeping digit seperators
void printBig(std::ostream &os, long long value);

// direct method for inserting seps
void printBigDirect(std::ostream &ou, long long value);


```

main.cc:

```
#include "header.ih"

int main(int argc, char* argv[])
{
    if (argc != 2)
    {
        std::cout << "Error: this program expects a single integer." << '\n';
        return 1;
    }

    long long num = std::stoll(argv[1]);

    std::cout << "direct method:" << '\n';
    printBigDirect(std::cout, num);
```

```
    std::cout << "recursive method:" << '\n';
    printBig(std::cout, num);
    std::cout << '\n';
}
```

printbig.cc:

```
#include "header.ih"

void printBig(std::ostream &ou, long long value)
{
    std::string val = std::to_string(value);
    size_t vsize = val.size();

    if (vsize > 3)
    {
        printBig(ou, std::stoll(val.substr(0, vsize - 3)));  // recursive all but last 3
        ou << '\'' << val.substr(vsize - 3, 3);              // last 3
    } else
    {
        ou << val;
    }

}

// description
```

printbigdirect.cc

```
#include "header.ih"

void printBigDirect(std::ostream &ou, long long value)
{
    std::string val = std::to_string(value);
    size_t len = val.length();

    std::string sepval;
    for (size_t idx = len + 1; idx != 0; --idx)                          // reverse over digits of val
    {
        if ((len - idx) % 3 == 0 && idx < len)                   // seperator every 3 digits
            sepval.insert(0, 1,  '\'');

        sepval.insert(0, 1, val[idx - 1]);                                  // digit
    }
    ou << sepval << '\n';
    return;
}

// description
```

usage.cc:

```
// instructions for users
#include "header.ih"

char const use[]=
R"(
20 V 1>
```

```
7
8   Usage: ./bin number
9   Where:
10      number is an integer
11
12  <20 processes number and writes it to the output stream after
13  adding seperators every 3 digits, using two different methods:
14  direct and indirect.
15  )";
16
17  void usage()
18  {
19      std::cout << use << '\n';
20  }
```