Programming report
Week 2 Assignments C++

Jaime Betancor Valado
Christiaan Steenkist
Remco Bos
Pepijn Sietsema

September 23, 2016

Assignment 12, Big Numbers

We were tasked with making a function that prints unsigned long longs with separators. We are proud to announce the recursive `printBig` function. The non-recursive variant of this amazing function will be revealed at a later date (at the resubmit mayhaps).

Code Listings

Listing 1: `bignum.h`

```
#ifndef BIGNUM_H
#define BIGNUM_H

#include <iostream>

struct Consts{
    const size_t baseTen = 10;
    const size_t steps = 3;
};

size_t posMod(long long value, size_t modulus);
void printBig(std::ostream &out, long long value);
char printSmall(std::ostream &out, long long value);

#endif
```

Listing 2: `posmod.cc`

```
#include "bignum.h"

size_t posMod(long long value, size_t modulus)
{
    return ((value % modulus) + modulus) % modulus;
}
```

Listing 3: `printbig.cc`

```
#include "bignum.h"

```

```
using namespace std;

void printBig(ostream &out, long long value)
{
    if (value == 0)
        return;

    if (value < 0)
    {
        value = -value;
        out << '-';
    }

    // This loop removes 3 digits of the value and prints them.
    Consts consts;
    string remainder;
    for (size_t digit = 0; digit < consts.steps; digit++)
    {
        remainder = printSmall(out, value) + remainder;
        value = value / consts.baseTen;
    }

    printBig(out, value);
    if (value != 0)
        out << '\'';
    out << remainder;
}
```

Listing 4: `printsmall.cc`

```
#include "bignum.h"

using namespace std;

char printSmall(ostream &out, long long value)
{
    if (value == 0)
        return value;

    Consts consts;

    size_t remainder = posMod(value, consts.baseTen);
    value = (value - remainder) / consts.baseTen;
    return '0' + remainder;
}
```

Assignment 13, Fibonacci

We were tasked with making an efficent as well as a horribly inefficient way of calculating Fibonacci numbers. The functions `fib` and `rawfib` are the two methods we implemented to give you these fascinating numbers.

Time Tests

To show how absolutely primitive `rawfib` is we did a timing test where both functions were tasked with calculating the Fibonacci numbers at places 0 through 41. Here are the results of these tests with at the top the performance of our beloved `fib` function and at the bottom the noticably slower `rawfib`.

- `time ./main 41` gave a real of 0m0.043s
- `time ./main 41 x` gave a real of 1m37.808s

Code Listings

Listing 5: `main.h`

```
#ifndef MAIN_H
#define MAIN_H

unsigned long long fib(unsigned long long value, unsigned long long fibval[]);
unsigned long long rawfib(unsigned long long value);

#endif
```

Listing 6: `main.cc`

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

using namespace std;

int main(int argc, char **argv)
{
    unsigned long long idx = stoull(argv[1]);
    unsigned long long fibNumber = 0;
    unsigned long long fibval[idx] = {0};

    for (size_t step = 0; step < idx; step++)
    {
        if (argc > 2)
            fibNumber = rawfib(idx);
        else
            fibNumber = fib(idx,fibval);
    }
    cout << "The " << idx << "th fibonacci number is: " << fibNumber <<"\n";
}
```

Listing 7: `fib.cc`

```
#include "main.h"

unsigned long long fib(unsigned long long value, unsigned long long fibval[])
{

    if (value <= 2)
    {
        fibval[0] = 1;
        fibval[1] = 1;
        return 1;
    }

    if (fibval[value - 1] != 0)
        return fibval[value - 1];

    fibval[value - 1] = fib(value - 1, fibval) + fibval[value - 3];
    return fibval[value - 1];
}
```

Listing 8: `rawfib.cc`

```
#include "main.h"

unsigned long long rawfib(unsigned long long value)
{
    if (value <= 2)
        return 1;
    else
        return rawfib(value - 1) + rawfib(value - 2);
}
```

Assignment 14, Evaluator

We were tasked with designing a program that evaluates the design quality of a program. The design quality can roughly be translated to the number of comments divided by the number of lines and is said to be an indicator for the quality of source files.

Score Calculations

The actual design quality is calculated by taking the ratio of two scores. These scores are the line scores and comment scores which are based on the number of lines and comments. Every line counts only once but after 30 lines they count double and at an indentation above 3 they count doubly as well. Every end of line comment counts but c-style comments only fully count when they are at the start of a file, otherwise they only contribute one point per block.

Code Listings

Listing 9: `main.h`

```
#ifndef MAIN_H
#define MAIN_H

#include <iostream>

// Design Quality is multiplied by 2
// therefore it starts at 2 and is
// later multiplied.
struct Vars{
    size_t stringPos = 0;
    size_t indent = 0;
    size_t maxIndent = 0;
    size_t lines = 0;
    size_t lineScore = 0;
    size_t eolComments = 0;
    size_t cStyleComments = 0;
    size_t commentScore = 0;
    size_t commentLength = 0;
    size_t designQuality = 2;
};

```

```
struct Consts{
    const size_t indentLimit = 3;
    const size_t lineLimit = 30;
    const size_t percent = 100;
};

std::string getInput();
void computeScores(Vars &vars, std::string const file);
void computeQuality(Vars &vars);
void printScores(Vars &vars);
void endComment(Vars &vars, std::string const file);
void endLine(Vars &vars);
void startComment(Vars &vars, std::string const file);

#endif
```

Listing 10: `main.cc`

```
#include "main.h"

int main (int argc, char **argv)
{
    Vars vars;
    computeScores(vars, getInput());
    computeQuality(vars);
    printScores(vars);
}
```

Listing 11: `computequality.cc`

```
#include "main.h"

void computeQuality(Vars &vars)
{
    Consts consts;

    vars.designQuality *= consts.percent;
    vars.designQuality *= vars.commentScore / vars.lineScore;
    vars.designQuality = vars.designQuality > consts.percent
        ? consts.percent : vars.designQuality;
}
```

Listing 12: `computescores.cc`

```
#include "main.h"

using namespace std;

// Only characters indicative of indentations, lines and comments are
// taken into account. Other characters are simply ignored.
void computeScores(Vars &vars, string const file)
{
    Consts consts;
    for (vars.stringPos = 0; vars.stringPos < file.size(); vars.stringPos++)
    {
        switch(file[vars.stringPos])
        {
            case('\n'):
                endLine(vars);
            break;

            case('/'):
                startComment(vars, file);
            break;

            case('*'):
                endComment(vars, file);
            break;

            case('}'):
                --vars.indent;
            break;
            case('{'):
                vars.maxIndent = ++vars.indent > vars.maxIndent
                    ? vars.indent
                    : vars.maxIndent;
            break;
        }
    }
}
```

Listing 13: `endcomment.cc`

```
#include "main.h"

using namespace std;

// The combination "*/" indicates the end of a c style comment.
// Loose '*' are ignored.
void endComment(Vars &vars, string const file)
{
    if (file[++vars.stringPos] == '/')
    {
        ++vars.cStyleComments;
        vars.commentScore += vars.commentLength == vars.lines
            ? vars.commentLength - 1
            : 1;
    }
    else
        --vars.stringPos;
}
```

Listing 14: `endline.cc`

```
#include "main.h"

void endLine(Vars &vars)
{
    Consts consts;

    size_t mul1 = (++vars.lines > consts.lineLimit ? 2 : 1);
    size_t mul2 = (vars.indent > consts.indentLimit ? 2 : 1);
    vars.lineScore += mul1 * mul2;
    ++vars.commentLength;
}
```

Listing 15: `getinput.cc`

```
#include "main.h"

using namespace std;

string getInput()
{
    string input;
    string line;
    while(getline(cin, line))
    {
        input = input + line + '\n';
    }
    return input;
}
```

Listing 16: `printscores.cc`

```
#include "main.h"

using namespace std;

void printScores(Vars &vars)
{
    cout << "This file has " << vars.lines << " lines.\n";
    cout << "This file has " << vars.eolComments << " end of line comments.\n"
    ;
    cout << "This file has " << vars.cStyleComments << " c-style comments.\n";
    cout << "This file is at most " << vars.maxIndent << " indentations deep.\
    n";
    cout << "The design quality of this file is " << vars.designQuality << ".\
    n";
}
```

Listing 17: `startcomment.cc`

```
#include "main.h"

using namespace std;

// The combination "//" indicates an end of line comment.
// The combination "/*" indicates a c style comment.
// Loose '/' are ignored.
void startComment(Vars &vars, string const file)
{
    if (file[++vars.stringPos] == '/')
    {
        ++vars.eolComments;
        ++vars.commentScore;
    }
    else if (file[vars.stringPos] == '*')
        vars.commentLength = 0;
    else
        --vars.stringPos;
}
```

Assignment 15, Bit Shifts

We were tasked with making a program that can use three different methods to find the most significant big. These methods use bit shifts, taking a logarithm and truncating and binary search. It should also return the least significant bit.

Time Tests

We did tests on big and large numbers for every method. The methods in order are `shiftSearch`, `truncateSearch` and `binarySearch`.

- Shift search: `time ./main 10 1 30000000` gave a real of 0m0.238s
- Truncate search: `time ./main 10 2 30000000` gave a real of 0m2.919s
- Binary search: `time ./main 10 3 30000000` gave a real of 0m0.859s
- Shift search: `time ./main 18446744073709551615 1 30000000` gave a real of 0m4.978s
- Truncate search: `time ./main 18446744073709551615 2 30000000` gave a real of 0m2.904s
- Binary search: `time ./main 18446744073709551615 3 30000000` gave a real of 0m0.874s

Results

It seems that the first method is better for the low values and the binary is almost constant in spite of the value and is the best with large numbers. The second one is also almost constant in time in spite of the value but in low values is the slow process and in high values the binary outperforms it in matters of time.

Code Listings

Listing 18: `main.h`

```
#ifndef MAIN_H
#define MAIN_H

#include <iostream>
#include <cmath>

size_t shiftSearch(unsigned long long number);
size_t truncateSearch(unsigned long long number);
size_t binarySearch(unsigned long long number);
size_t lsbSearch(unsigned long long number);
#endif
```

Listing 19: `main.cc`

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

using namespace std;

int main(int argc, char **argv)
{
    unsigned long long value = stoull (argv[1], nullptr, 10);

    int met = stoi(argv[2], nullptr, 10);

    //1 iteration per default
    size_t iter = 1;
    if (argc == 4)
        iter = stoi(argv[3], nullptr, 10);

    size_t msbOff = 0;

    for (size_t idx = 1; idx <= iter; ++idx)
    {
        // 3 functions that search most significant bit
        switch (met)
        {
        case 1:
            msbOff = shiftSearch(value);
        break;
        case 2:
            msbOff = truncateSearch(value);
        break;
        case 3:
            msbOff = binarySearch(value);
        break;
        }

    }

    size_t lsbOff = lsbSearch(value);

    cout << "The MSB offset is: " << msbOff << "\n";
    cout << "The LSB offset is: " << lsbOff << "\n";
}
```

Listing 20: binary.cc

```
#include "main.h"

size_t binarySearch(unsigned long long number)
{
    int msb = 0;
    size_t low = 0;
    size_t high = 8 * sizeof (number);
    size_t mid = 0;
    bool exit = true;
    unsigned long long shift = 1;

    while(exit)
    {
    mid = (low + high) / 2;

    // I write this two lines because if I put them
    // in the else if the compiler interprets 1 << mid as an int
    // and then there are overflows.
    shift = 1;
    shift = shift << mid;

    if (mid == low)
    {
        msb = mid;
        exit = false;
    }
    else if (shift > number)
        high = mid;
    else
        low = mid;
    }
    return msb - 1;
}
```

Listing 21: `lsb.cc`

```
#include "main.h"

size_t lsbSearch(unsigned long long number)
{
    size_t rest = 0;;
    bool exit = true;
    size_t lsb = 1;

    while(exit)
    {
        rest = number % 2;
        number /= 2;

        if (rest == 0)
            ++lsb;
        else exit = false;
    }

    return lsb - 1;
}
```

Listing 22: `shift.cc`

```
#include "main.h"

size_t shiftSearch(unsigned long long number)
{
    bool exit = true;
    size_t msb = 0;

    while (exit)
    {
        number = number >> 1;
        ++msb;
        if (number == 1)
            exit = false;
    }

    return msb - 1;
}
```

Listing 23: `truncate.cc`

```
#include "main.h"

const double ln2 = 0.693147181;

size_t truncateSearch(unsigned long long number)
```

```
{
    size_t msb = 0;
    msb = log(number) / ln2;
    return msb - 1;
}
```

Assignment 16, Cipher

We were tasled with making a Vigenre cipher program. The idea of the program is to implement a Vigenre cipher that can translate newlines, tabs and all printable characters.

Design

To encrypt and decrypt the program goes through 3 simple steps for each character. First the program translates the character to a compressed ascii table using the `compressChar` function. This makes the character inhabit a range of numbers from zero to the total number of character minus one. The second step is shifting the compressed character by the right amount. For this the apropriate character in the cipher is used after it has also been compressed. After the shift the now translated character is shifted back into the normal ascii table again and printed.

Code Listings

Listing 24: `main.ih`

```
#ifndef CIPHER_H
#define CIPHER_H

#include <iostream>

using namespace std;

enum Action
{
    USAGE, ENCRYPT, DECRYPT
};

struct Vars
{
    string input;
    string cipher;
    int action;
};

Vars arguments(int argc, char **argv);
char compressChar(char character);
char decompressChar(char character);
void process(Vars args);

#endif
```

Listing 25: `main.cc`

```
#include "main.ih"

int main(int argc, char **argv)
```

```cpp
{
    process(arguments(argc, argv));
}
```

Listing 26: `arguments.cc`

```cpp
#include "main.ih"

Vars arguments(int argc, char **argv)
{
    string line;
    string input;
    while (getline(cin, line))
    {
        input += line;
    }

    int action = USAGE;
    if (argc == 2)
        action = ENCRYPT;
    else if (argc >= 3)
        action = DECRYPT;

    Vars args;

    if (action == USAGE)
    {
        args.input = "At least input a cipher.";
        args.cipher = "nocipher";
    }
    else
    {
        args.input = input;
        args.cipher = argv[1];
    }

    args.action = action;
    return args;
}
```

Listing 27: `compresschar.cc`

```cpp
// Characters \t and \n take the first two indices,
// the other printable characters take the rest, starting at ' '.
char compressChar(char character)
{
    switch(character)
    {
        case('\t'):
            return 0;
        case('\n'):
            return 1;
        default:
            return 2 + character - ' ';
    }
}
```

Listing 28: `decompresschar.cc`

```
// Numbers 0 and 1 are mapped to \t and \n,
// the other numbers are mapped to the printable characters
// starting with ' '.
char decompressChar(char character)
{
    switch(character)
    {
        case(0):
            return '\t';
        case(1):
            return '\n';
        default:
            return character + ' ' - 2;
    }
}
```

Listing 29: `process.cc`

```
#include "main.ih"

// The encryption/decryption occurs as follows.
// 1: The characters are compressed into the range 0 to
//       alphabetSize - 1.
// 2: The numbers are shifted by encryption/decryption.
// 3: the number range is transformed back into the
//       printable characters.
void process(Vars args)
{
    int alphabetSize = 2 + '~' - ' ';
    int cipherSize = args.cipher.length();

    size_t step = 0;
    for (char character : args.input)
    {
        char cipherChar = compressChar(args.cipher[step]);
        char newChar = compressChar(character);

        if (args.action == ENCRYPT)
            newChar = (newChar + cipherChar) % alphabetSize;
        else
            newChar = (alphabetSize + newChar - cipherChar) % alphabetSize;

        newChar = decompressChar(newChar);
        cout << newChar;
        step = (step + 1) % cipherSize;
    }
}
```