Regular: Jaap van der Leest
Attendee: Timon van der Berg
October 3, 2017

C++ Course
Assignment 3

Exercise 18

- 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.

```
1   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.

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

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

```
1   int multiply(int first, int second)
2   {
3       return (first * second);
4   }
```

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

```
1   std::string helloWorld(void)
2   {
3       std::string hello = "Hello World!\n";
4       return(hello);
5   }
```

- 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:

```
1   exercise_19 : exercise_19.o method1.o method2.o method3.o method4.o \
2               method5.o method6.o
3                   g++ -std=c++17 exercise_19.o method1.o method2.o method3.o \
4                   method4.o method5.o method6.o -o exercise_19
5
6   exercise_19.o : exercise_19.cc myheader.ih
7       g++ -std=c++17 -Wall -Werror -c exercise_19.cc
8   method1.o : method1.cc myheader.ih
9       g++ -std=c++17 -Wall -Werror -c method1.cc
10  method2.o : method2.cc myheader.ih
11      g++ -std=c++17 -Wall -Werror -c method2.cc
12  method3.o : method3.cc myheader.ih
13      g++ -std=c++17 -Wall -Werror -c method3.cc
14  method4.o : method4.cc myheader.ih
15      g++ -std=c++17 -Wall -Werror -c method4.cc
16  method5.o : method5.cc myheader.ih
17      g++ -std=c++17 -Wall -Werror -c method5.cc
18  method6.o : method6.cc myheader.ih
19      g++ -std=c++17 -Wall -Werror -c method6.cc
20  clean :
21          rm exercise_19 exercise_19.o method1.o method2.o method3.o method4.o \
22              method5.o method6.o
```

myheader.ih:

```cpp
// 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:

```cpp
// 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);
```

```
29        else if (method == 5)
30            method5(valueToAnalyze, nrOfTurns);
31        else if (method == 6)
32            method6(valueToAnalyze, nrOfTurns);
33        else
34            cout << "Please provide the second argument as a number 1 - 6.\n";
35
36    }
37
38    /*
39    The fastest method for finding the MSB depends on the input.
40    Method 1 is quick for small nrs. But big nrs can request up to 64 calculations.
41    Method 3 is relative fast when calculating big nrs. This method takes a maximum
42    of 6 calculations.
43    Method 2 seems to have a constant calc speed which is never the fastest.
44    */
```

method1.cc:

```
1    // method1.cc
2    #include "myheader.ih"
3
4    using namespace std;
5
6
7    void method1(unsigned long long int const valueToAnalyze, int nrOfTurns)
8    {
9        size_t counter = 0;
10
11        while (nrOfTurns != 0)
12        {
13            unsigned long long int valueToCompute = valueToAnalyze;
14            counter = 0;
15            while (valueToCompute != 0)
16            {
17                valueToCompute >>= 1;        // right shift bits
18                ++counter;
19            }
20
21            counter -= 1;                    // calc offset not the nr of bits
22            --nrOfTurns;
23        }
24
25        cout << "Method 1 right shift bits.\n";
26        cout << "MSbit of " << valueToAnalyze << " is at bit offset "
27            << counter << '\n';
28
29        return;
30    }
```

method2.cc:

```
1    // method2.cc
2    #include "myheader.ih"
3
4    using namespace std;
5
6
7    double const ln2 = log (2);
```

4

```
 8
 9    void method2(unsigned long long int const valueToAnalyze, int nrOfTurns)
10    {
11        size_t counter = 0;
12
13        while (nrOfTurns != 0)
14        {
15            unsigned long long int valueToCompute = valueToAnalyze;
16
17            counter = log (valueToCompute) / ln2;  // ln2 lives in ln2.cc
18
19            --nrOfTurns;
20        }
21
22        cout << "Method 2 logarithm.\n";
23        cout << "MSbit of " << valueToAnalyze << " is at bit offset "
24            << counter << '\n';
25
26        return;
27    }
```

method3.cc:

```
 1    // method3.cc
 2    #include "myheader.ih"
 3
 4    using namespace std;
 5
 6
 7    void method3(unsigned long long int const valueToAnalyze, int nrOfTurns)
 8    {
 9        size_t counter = 0;
10
11        while (nrOfTurns != 0)
12        {
13            unsigned long long int valueToCompute = valueToAnalyze;
14            size_t ttLow = 0;
15            size_t ttHigh = sizeof (valueToCompute) * 8;          // nr of bytes * 8 bits
16            size_t ttMid;
17
18            ttMid = (ttLow + ttHigh) / 2;
19
20            while (true)
21            {
22                valueToCompute = valueToAnalyze;
23
24                                                                  // compute if all bits are before ttMid
25                size_t shiftedValue = valueToCompute >>= ttMid;
26                if (shiftedValue == 0)
27                {
28                    ttHigh = ttMid;
29                    ttMid = (ttLow + ttHigh) / 2;
30                }
31
32                else if (ttLow == ttMid)
33                {
34                    counter = ttMid;
35                    break;
36                }
37                else
```

```
38              {
39                  ttLow = ttMid;
40                  ttMid = (ttLow + ttHigh) / 2;
41              }
42          }
43
44          --nrOfTurns;
45      }
46
47      cout << "Method 3 binary search.\n";
48      cout << "MSbit of " << valueToAnalyze << " is at bit offset "
49          << counter << '\n';
50
51      return;
52  }
```

method4.cc:

```
1   // method4.cc
2   #include "myheader.ih"
3
4   using namespace std;
5
6
7   void method4(unsigned long long int const valueToAnalyze, int nrOfTurns)
8   {
9       size_t counter = 0;
10
11      while (nrOfTurns != 0)
12      {
13          unsigned long long int valueToCompute = valueToAnalyze;
14          counter = 0;
15
16          while ((valueToCompute & 1) != 1)
17          {
18              valueToCompute >>= 1;    // right shift bits
19              ++counter;
20          }
21
22          --nrOfTurns;
23      }
24
25      cout << "Method 4 right shift bits.\n";
26      cout << "LSbit of " << valueToAnalyze << " is at bit offset "
27          << counter << '\n';
28
29      return;
30  }
```

method5.cc:

```
1   // method5.cc
2   #include "myheader.ih"
3
4   using namespace std;
5
6
7   void method5(unsigned long long int const valueToAnalyze, int nrOfTurns)
8   {
```

```
 9
10      while (nrOfTurns != 0)
11      {
12          // doing nothing as many times as you ask
13          --nrOfTurns;
14      }
15
16      cout << "Method 5 logarithm.\n";
17      cout << "I don't think it is possible to calculate the LSbit with "
18          << "logarithm \n";
19
20      return;
21  }
```

method6.cc:

```
 1  // method6.cc
 2  #include "myheader.ih"
 3
 4  using namespace std;
 5
 6
 7  void method6(unsigned long long int const valueToAnalyze, int nrOfTurns)
 8  {
 9      size_t counter = 0;
10      while (nrOfTurns != 0)
11      {
12          unsigned long long int valueToCompute = valueToAnalyze;
13          size_t ttLow = sizeof (valueToCompute) * 8;          // nr of bytes * 8 bits
14          size_t ttHigh = 0;
15                                                               // ttLow and ttHigh values are swapped for LSB
16
17          size_t ttMid;
18          ttMid = (ttLow + ttHigh) / 2;
19
20          while (true)
21          {
22              valueToCompute = valueToAnalyze;
23
24                                                               // compute if last bit is before ttMid
25              size_t shiftedValue = valueToCompute <<= ttMid;
26              if (shiftedValue == 0)
27              {
28                  ttLow = ttMid;
29                  ttMid = (ttLow + ttHigh) / 2;
30              }
31
32              else if (ttHigh == ttMid)
33              {
34                  counter = sizeof (valueToAnalyze) * 8 - (ttMid + 1);
35                  break;
36              }
37              else
38              {
39                  ttHigh = ttMid;
40                  ttMid = (ttLow + ttHigh) / 2;
41              }
42          }
43
44          --nrOfTurns;
```

```
45        }
46
47        cout << "Method 6 binary search.\n";
48        cout << "LSbit of " << valueToAnalyze << " is at bit offset "
49            << counter << '\n';
50
51        return;
52    }
```

Exercise 20

head.ih:

```
1    #include <unistd.h>           // isatty
2    #include <iostream>           // cin, cout
3    #include <getopt.h>           // getopt_long
4
5    // processing type
6    enum class Mode {
7        ERROR,
8        CAPITALIZE,
9        LOWER_CASE,
10       VERSION,
11       USAGE
12   };
13
14   // arguments type
15   struct vars_t {
16       bool help;               // -h --help
17       bool version;            // -v --version
18       bool capitalize;         // -c --uc --capitalize
19       bool lowercase;          // -l --lc --lower-case
20   };
21
22   // info for user
23   void usage();
24
25   // process input
26   void process(vars_t Vars);
27
28   // do stuff
29   vars_t arguments(int argc, char* argv[]);
30
31   // select mode from arguments
32   Mode selectOpt(vars_t Vars);
33
34   // cout version num
35   void version();
```

main.cc:

```
1    #include "head.ih"
2
3    int succesState = 0;                      // expections NC
4
5    int main(int argc, char* argv[])
```

8

```
6    {
7        if (isatty(0))
8        {
9            std::cout << "no file redirection" << '\n';
10           return 1;
11       }
12       process(arguments(argc,argv));
13       return succesState;
14   }
```

arguments.cc:

```
1    #include "head.ih"
2
3    // long options and short options
4    struct option longOpts[] =
5    {
6        {"capitalize",  0,     0, 'c'},
7        {"uc",          0,     0, 'c'},
8        {"lowercase",   0,     0, 'l'},
9        {"lc",          0,     0, 'l'},
10       {"version",     0,     0, 'v'},
11       {"help",        0,     0, 'h'},
12       { 0 }
13   };
14
15   vars_t arguments(int argc, char* argv[])
16   {
17       vars_t Vars = {false, false, false, false};
18       int opt;
19       while ((opt = getopt_long(argc, argv, "hvcl", longOpts, &opt)) != -1)
20           switch (opt)
21           {
22               case 'h':                    // help
23               {
24                   Vars.help = true;
25                   break;
26               }
27               case 'v':                    // version
28               {
29                   Vars.version = true;
30                   break;
31               }
32               case 'c':                    // capitalize
33               {
34                   Vars.capitalize = true;
35                   break;
36               }
37               case 'l':                    // lower-case
38               {
39                   Vars.lowercase = true;
40                   break;
41               }
42               default:
43               {
44                   Vars.help = true;
45                   break;
46               }
47           }
```

```
48        return Vars;
49    }
```

process.cc:

```
1    #include "head.ih"
2    #include <cctype>                                // toupper, tolower
3    extern int succesState;
4
5    void process(vars_t Vars)
6    {
7        Mode option = selectOpt(Vars);
8        switch (option)
9        {
10           case (Mode::ERROR):
11           {
12               succesState = 1;
13               std::cout << "ERROR" << '\n';
14               break;
15           }
16           case (Mode::USAGE):
17           {
18               usage();
19               break;
20           }
21           case (Mode::VERSION):
22           {
23               version();
24               break;
25           }
26           case (Mode::CAPITALIZE):
27           {
28               char ch;
29               while (std::cin.get(ch)) std::cout << static_cast<char>(toupper(ch));
30               break;
31           }
32           case (Mode::LOWER_CASE):
33           {
34               char ch;
35               while (std::cin.get(ch)) std::cout << static_cast<char>(tolower(ch));
36               break;
37           }
38        }
39        return;
40    }
```

selectopt.cc:

```
1    #include "head.ih"
2
3    Mode selectOpt(vars_t Vars)
4    {
5        if (Vars.help)
6            return Mode::USAGE;
7        if (Vars.version)
8            return Mode::VERSION;
9        if (Vars.capitalize and Vars.lowercase)         // can't do both
10       {
11           return Mode::ERROR;
```

```cpp
12          }
13          if (Vars.capitalize)
14              return Mode::CAPITALIZE;
15          if (Vars.lowercase)
16              return Mode::LOWER_CASE;
17          std::cout << "Invalid argument provided.";
18          return Mode::ERROR;
19
20      }
```

usage.cc:

```cpp
1   // instructions for users
2   #include "head.ih"
3
4   char const use[]=
5   R"(
6   20 V 1
7
8   Usage: ./main [options] < file
9   Where:
10      --captitalize   (--uc, -u);     captitalize the letters in 'file'
11      --help          (-h);           display this information
12      --lowercase     (--lc, -l);     convert letters to lowercase in 'file'
13      --version       (-v);           display version information
14
15  20 processes 'file' and writes the results
16  to the standard output stream.
17  )";
18  void usage()
19  {
20      std::cout << use << '\n';
21  }
```

version.cc:

```cpp
1   #include "head.ih"
2
3   void version(){
4       std::cout << "Version 1.45.12c.EY RC 5" << '\n';
5   }
```

Exercise 21

myheader.ih:

```cpp
1   // myheader.ih
2   #include <iostream>
3   #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:

```cpp
// main.cc
#include "myheader.ih"

using namespace std;

int main (int argc, char* argv[])
{
    string nrToCalc = argv[1];
    square(nrToCalc, "", "");   // left factor and rest are 0 at start
}
```

partial.cc:

```cpp
// partial.cc
#include "myheader.ih"

using namespace std;

int partial(string firstBlock, string leftFactor, string& rest)
{
    string newNr = "";
    int digit = 9;

    while(true)
    {
        newNr = leftFactor + to_string(digit);
        if (stoi(newNr) * digit <= stoi(firstBlock))
            break;
        --digit;
    }

    rest = to_string(stoi(firstBlock) - stoi(newNr) * digit);
    cout << digit;
    return(digit);
}
```

square.cc

```cpp
// square.cc
#include "myheader.ih"

using namespace std;

void square(string input, string leftFactor, string rest)
{
    string firstBlock = "";
    size_t length = input.length();

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

    if (length % 2 == 0)
    {
```

```
19        firstBlock = input.substr(0,2);   // take first 2 digits to calculate
20        input.erase(0, 2);
21    }
22    else
23    {
24        firstBlock = input.substr(0,1);   // take only the first digit
25        input.erase(0, 1);
26    }
27
28    firstBlock = rest + firstBlock;
29    string& remainder = rest;             // this reference can access rest from
30                                          // within the function partial()
31
32    int digit = 0;
33    digit = partial(firstBlock, leftFactor, remainder);
34    leftFactor = to_string(stoi(leftFactor + to_string(digit)) + digit);
35
36    square(input, leftFactor, rest);      // the function calls itself to
37                                          // calculate the rest
38    return;
39 }
```

Exercise 22

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

```
1  #!/bin/bash
2
3  ./main -d < input.txt >> output.txt
4  ./main -e < output.txt >> return.txt
5
6  echo "INPUT=============="
7  cat input.txt
8  echo "OUTPUT============"
9  cat output.txt
10 echo "INPUT?============"
11 cat return.txt
12
13 rm output.txt return.txt
```

header.ih:

```
1  #include <string>
2  #include <iostream>
3
4  // info for users
5  void usage(std::string const &programName);
6
7  // command line options
8  enum class EOption {ENCODE,
9                      DECODE,
10                     NONE};
11
12 // which command line option was provided
13 EOption getOpt(std::string opt);
14
```

```
15    // convert 2-digit hex to decimal
16    size_t hexToDec(std::string str);
17
18    // convert decimal to 2-digit hex
19    std::string decToHex(size_t num);
20
21    // test if ch is alphanumerical
22    bool isAlpha(char ch);
23
24    // test if ch is ˜ . - _
25    bool isOther(char ch);
26
27    // url-encode the stream
28    void encode(std::istream &is, std::ostream &os);
29
30    // decode url-encoded stream
31    void decode(std::istream &is, std::ostream &os);
```

main.cc

```
1    #include "header.ih"
2
3    int main(int argc, char* argv[])
4    {
5        std::string arg = (argc == 2 ? argv[1] : "");     // read if possible
6
7        switch (getOpt(arg))
8        {
9            case EOption::ENCODE:
10                encode(std::cin, std::cout);
11                break;
12            case EOption::DECODE:
13                decode(std::cin, std::cout);
14                break;
15            default:
16                usage(argv[0]);
17                int FAIL = 1;
18                return FAIL;
19        }
20    }
21
22    // Exercise 22: an URL stream decoder / encoder
23    //
24    //         Usage: main [-e/-d] < input.txt
25    //               (or provide stdin in other way)
26    //         Where:
27    //             -e        url-encode input
28    //             -d        decode url-encoded input
29    //         input.txt
30    //             contains either
31    //                 an url-encoded string (when using -d)
32    //                 an url-decoded string (when using -e)
```

decode.cc

```
1    #include "header.ih"
2    // decode url-encoded stream
3
4    void decode(std::istream &is, std::ostream &os)
```

14

```
 5    {
 6        char ch;
 7        while (is.get(ch))
 8        {
 9            if (ch == '%' && is.get(ch))
10            {
11                std::string str;
12                str.push_back(ch);                    // 2nd digit
13                if (is.get(ch))
14                    str.append(1, ch);
15                os << static_cast<char>(hexToDec(str));
16            }
17            else
18            os << ch;                                 // skip
19        }
20        return;
21    }
```

dectohex.cc

```
 1    #include "header.ih"
 2    // convert decimal to 2-digit hex
 3
 4    std::string decToHex(size_t num)
 5    {
 6        size_t radix = 16;
 7        std::string buff;
 8        while (num != 0)                              // process digits
 9        {                                            // in reverse
10            size_t remainder = num % radix;
11            if (remainder > 9)
12                buff.insert(0, 1, 'A' + remainder - 10);      // letter
13            else
14                buff.insert(0, 1, '0' + remainder);           // number
15            num /= radix;
16        }
17        if (buff.length() != 2) buff.insert(0,1,'0');        // trailing 0
18        return buff;
19    }
```

encode.cc:

```
 1    #include "header.ih"
 2    // url-encode stream
 3
 4    void encode(std::istream &is, std::ostream &os)
 5    {
 6        char ch;
 7        while (is.get(ch))
 8            if (isAlpha(ch) or isOther(ch))               // skip
 9                os << ch;
10            else
11                os << '%' << decToHex((size_t) ch);       // encode
12        return;
13    }
```

getopt.cc:

```
1   #include "header.ih"
2   // determine which option was provided
3
4   EOption getOpt(std::string opt)
5   {
6       if (opt == "-e") return EOption::ENCODE;
7       if (opt == "-d") return EOption::DECODE;
8
9       return EOption::NONE;
10  }
```

hextodec.cc

```
1   #include "header.ih"
2   // convert 2-digit hex to decimal
3
4   size_t hexToDec(std::string str)
5   {
6       std::string hexDigits = "0123456789ABCDEF";
7       return hexDigits.find(str[0]) * 16 + hexDigits.find(str[1]);   // hex base 16
8
9   }
```

isalpha.cc

```
1   #include "header.ih"
2   // test if ch is alphanumerical
3
4   bool isAlpha(char ch)
5   {
6       if (ch >= '0' && ch <= '9')
7           return true;
8       if (ch >= 'A' && ch <= 'Z')
9           return true;
10      if (ch >= 'a' && ch <= 'z')
11          return true;
12      return false;
13  }
```

isother.cc

```
1   #include "header.ih"
2   // test if ch is part of exlude set
3
4   bool isOther(char ch)
5   {
6       std::string others = "-_.~";                          // to skip
7       return others.find(ch) != std::string::npos;
8   }
```

usage.cc

```
1   #include "header.ih"
2   // instructions for users
3
4   char const use[]=
5   R"(
6   Exercise 22: an URL stream decoder / encoder
```

```
7
8    Usage: main [-e/-d] < input.txt
9        (or provide stdin in other way)
10   Where:
11       -e          url-encode input
12       -d          decode url-encoded input
13   input.txt
14       contains either
15           an url-encoded string (when using -d)
16           an url-decoded string (when using -e)
17   )";
18
19   void usage(std::string const &programName)
20   {
21       std::cout << use << '\n';
22   }
```

Exercise 23

header.ih:

```
1    #include <iostream>
2    #include <string>   // string, to_string
3
4    // insert value, keeping digit seperators
5    void printBig(std::ostream &os, long long value);
6
7    // direct method for inserting seps
8    void printBigDirect(std::ostream &ou, long long value);
9
10
```

main.cc:

```
1    #include "header.ih"
2
3    int main(int argc, char* argv[])
4    {
5        if (argc != 2)
6        {
7            std::cout << "Error: this program expects a single integer." << '\n';
8            return 1;
9        }
10
11       long long num = std::stoll(argv[1]);
12
13       std::cout << "direct method:" << '\n';
14       printBigDirect(std::cout, num);
15
16       std::cout << "recursive method:" << '\n';
17       printBig(std::cout, num);
18       std::cout << '\n';
19   }
```

printbig.cc:

```cpp
#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

```cpp
#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:

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

char const use[]=
R"(
23 V 1

Usage: ./bin number
Where:
    number is an integer

<20 processes number and writes it to the output stream after
adding seperators every 3 digits, using two different methods:
direct and indirect.
```

```cpp
    )";

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