Université Grenoble Alpes 2018-2019
M1 MSIAM

Algorithms and software tools - C++
Lab 3

Exercise 1. *Limitations of integer encoding*

1. Write a simple (C) program that takes as argument a positive integer x, computes and prints the value of factorial(x), for example:

```
$ my_factorial 6
The value of factorial(6) is 720
```

Up to which value of its argument does this application compute the expected result? Can you explain why? (to answer, use the information of the tables given in Appendix).

2. The directory `/home/p/pierrlau/MSIAM/bigint` (on mandelbrot) contains

- a shared library `libBigInt.so` that enables to encode big integers, using a class *BigInteger*
- a subdirectory `include` that contains the corresponding headers. In particular, the file `BigInteger.hh` contains the declaration of this class *BigInteger*, you can remark that it provides various methods: constructors, accessors, overloading of operators,…
- a example application `sample.cc`

Copy the file `sample.cc` in your own directory (you **do not need** to copy the other files). It includes `BigIntegerLibrary.hh` and it requires `libBigInt.so` for its compilation, you can simply use the appropriate compiling options `-I`, `-L` and `-l`.

Compile this application and execute it. Check that you get the results indicated within comments at the end `sample.cc`. Be sure that you understand the interest of using the class *BigInteger*, and how it can be used.

3. Modify your program of question 1. in order to use the class *BigInteger*. Do you observe the same results?

Exercise 2. *Generic collection*

1. Define a generic class *MyCollection* that represents a collection of elements of type *T*. These elements are stored in a C-style array which is dynamically allocated by the constructor, and deallocated by the destructor.

The attributes of this class are: the address of the array, the maximum number of elements that can be stored in the array (constant, initialized by the constructor), and the current number of elements stored in the array.

It has at least a method *insert_elem* to insert a new element at the end of the collection, and a method *get_elem* that enables to get the element stored at a specific index *i*. Define also an overloading of the << operator for this class.

2. Define two independent functions
void init(MyCollection<int> &c, int k);
and *void apply_fact(const MyCollection<int> &c, MyCollection<int> &res);*

The first one initializes the collection *c* with *k* integers (chosen pseudo-randomly), and function *apply_fact* fills the collection *res* with the factorials of the elements of collection *c*.

Create a *main* function to check these functions.

3. Define variants of the functions of question 2. to consider elements of type *BigInteger* (of exercise 1).

4. Adapt the functions of questions 2. and 3. to use the generic class *vector* (of the STL) instead of *MyCollection*.

Exercise 3. *Extension of exercises 1 and 2 of Lab 1…*

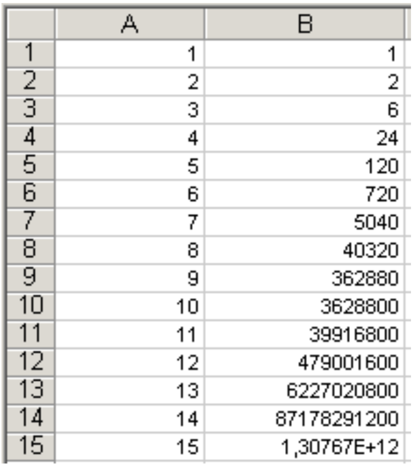
1. Modify your class *Date* so that it overloads the operators < and - as member functions. Adapt the definition of the method *duration* : it now uses these overloaded operators in place of functions *before* and *difference*.

2. Use the class *Trip* and the *main* function of the exercise 2 of Lab1 (they should be unchanged) to check this new implementation.

3. Now overload the operators of question 1. as independent functions.

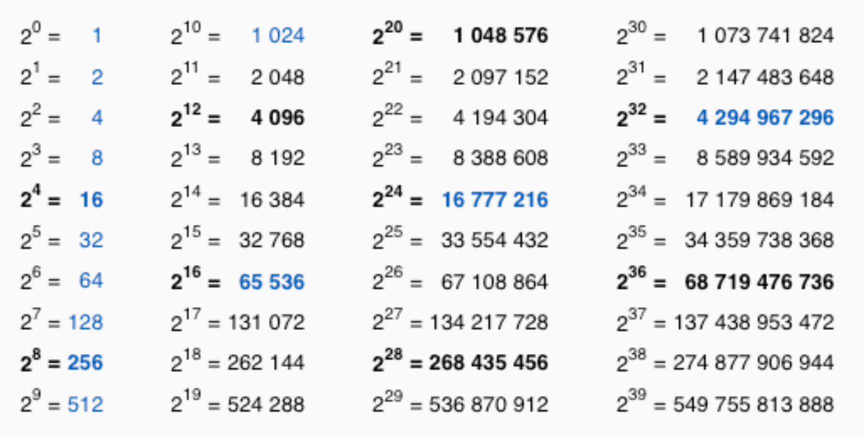
Appendix

Factorials :



	A	B
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5040
8	8	40320
9	9	362880
10	10	3628800
11	11	39916800
12	12	479001600
13	13	6227020800
14	14	87178291200
15	15	1,30767E+12

Powers of 2 :



2^{0} = 1
2^{1} = 2
2^{2} = 4
2^{3} = 8
2^{4} = 16
2^{5} = 32
2^{6} = 64
2^{7} = 128
2^{8} = 256
2^{9} = 512

2^{10} = 1 024
2^{11} = 2 048
2^{12} = 4 096
2^{13} = 8 192
2^{14} = 16 384
2^{15} = 32 768
2^{16} = 65 536
2^{17} = 131 072
2^{18} = 262 144
2^{19} = 524 288

2^{20} = 1 048 576
2^{21} = 2 097 152
2^{22} = 4 194 304
2^{23} = 8 388 608
2^{24} = 16 777 216
2^{25} = 33 554 432
2^{26} = 67 108 864
2^{27} = 134 217 728
2^{28} = 268 435 456
2^{29} = 536 870 912

2^{30} = 1 073 741 824
2^{31} = 2 147 483 648
2^{32} = 4 294 967 296
2^{33} = 8 589 934 592
2^{34} = 17 179 869 184
2^{35} = 34 359 738 368
2^{36} = 68 719 476 736
2^{37} = 137 438 953 472
2^{38} = 274 877 906 944
2^{39} = 549 755 813 888

Remark. Most companies and institutions require their software developers to respect a set of guidelines and conventions when writing code. This includes indentation style, variable naming, commenting, and more generally recommendations on what language constructs are preferred in a given situation. This is especially important in C++ where there are often many options to do the same thing.
The following document describes the conventions used by Google in their C++ projects: *https://google.github.io/styleguide/cppguide.html*