HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

BBM203 PROGRAMMING LAB.
ASSIGNMENT 2

Subject	**:** Data Structures and Algorithms (Stack, C++ Templates)
Submission Date	**:** 15.10.2014
Deadline	**:** 31.10.2014, 23:59 pm (Included Late Submit)
Programming Language	**:** C++ (GCC 4.4.7 20120313 (Red Hat 4.4.7-4))
Advisors	**:** Asst. Prof. Dr. Mustafa EGE, Asst. Prof. Dr. Lale ÖZKAHYA, R. A. Alaettin UÇAN

1. INTRODUCTION / AIM:

In this experiment you're expected to gain knowledge on advanced C++ topics including operator overloading, references, templates, and The Standard Template Library (STL). Prior knowledge on basic C++ syntax and class declarations is assumed.

2. BACKGROUND INFORMATION

2.1. C++: A MULTI-PARADIGM LANGUAGE

C++ is a general-purpose computer programming language. Since the 1990s, C++ has been one of the most popular commercial programming languages.

It is a multi-paradigm language supporting procedural programming (like C, Pascal, and other structured imperative languages), data abstraction (i.e. private members), object-oriented programming, and generic programming (i.e. templates).

Its template mechanism is so powerful that, in the recent years C++ has been “rediscovered”, even to surprise
its developers.

2.2. C++ STANDARD LIBRARY

The original C++ standard library contained the C standard library and a new I/O streams library, but lacked any standard base classes or templates. Later SGI's STL, which contains a set of class and function templates has been incorporated in the C++ standard. C++ standard library is still being extended.

There have been changes to inclusion and usage of library items. All the items are grouped in the “std” name space. And, the header names do no longer contain any suffix (no .h, .hpp or .hxx) and C standard library header names are preceded by “c” (e.g.: cmath).

2.3. TEMPLATES

C++ Function templates are those functions which can handle different data types without separate code for each of them. For a similar operation on several kinds of data types, a programmer does not need to write different versions by overloading a function. It will be sufficient if he writes a C++ template based function. This will take care of all the data types.

A trivial example is the “max” function. Normally one needs to define a specific function for each and every
data type necessary, because C++ is a strongly typed language.

However with C++ templates, we can define a “generic” function, and expect the compiler to generate and compile a specific function when necessary.

Whereas function templates can be used for generic functions, class templates can be used for generic classes.

```
template<typename T>
T Max(const T &a, const T &b)
{
	if(a > b)
		return a;
	else
		return b;
}

Max<int>(3, 5);
Max<double>(42.1,  43.6);
```

Example 1: This code will “instantiate” two different Max functions: Max<int> and Max<double>.

2.4. OPERATOR OVERLOADING

In computer programming, operator overloading (less commonly known as operator ad-hoc polymorphism) is a specific case of polymorphism in which some or all of operators like +, =, or == have different implementations depending on the types of their arguments. Sometimes the overloadings are defined by the language; sometimes the programmer can implement support for new types.

Operator overloading is useful because it allows the developer to program using notation closer to the target domain and allows user types to look like types built into the language[1].

```
Time Time::operator+(const Time& rhs) const
{
	Time result(this.hours, this.minutes, this.seconds);

	result.seconds += rhs.seconds;
	if (result.seconds >= 60)
	{
		result.seconds -= 60;
		result.minutes++;
	}
	result.minutes += rhs.minutes;
	if (result.minutes >= 60)
	{
		result.minutes -= 60;
		result.hours++;
	}
	result.hours += rhs.hours;
	return result;
}
```

Example 2: The addition operator is overloaded to allow addition on a user-defined type "Time".

2.5. ARBITRARY PRECISION ARITHMETIC

In computer science, arbitrary-precision arithmetic is a technique whereby calculations are performed on numbers whose digits of precision are limited only by the available memory of the host system. This contrasts with the faster fixed-point arithmetic found in most ALU hardware, which

[1] From http://en.wikipedia.org/wiki/Operator_overloading

typically offers between 6 and 16 decimal digits. It is also called bignum arithmetic, and sometimes even "infinite-precision arithmetic" (which is a misnomer, since the number of digits is both finite and bounded in practice)[2].

2.6. ONLINE RESOURCES

- Wikipedia C++ Entry: http://en.wikipedia.org/wiki/C_plus_plus
- SGI STL Docs: http://www.sgi.com/tech/stl/
- libstdc++ Docs: http://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/
- About.Com STL Tutorials: http://cplus.about.com/od/stl/
- C++ FAQ: http://www.parashift.com/c++-faq-lite/

3. EXPERIMENT

In this experiment, your aim will be to create a mathematical expression evaluator using arbitrary precision arithmetic. You have to implement addition, subtraction, multiplication, division and power operations for arbitrary precision numbers. To accomplish this, you have to use operator overloading, and your operators will be **+** for addition, **-** for subtraction, ***** for multiplication, **/** for division and **^** for power.

Your design should be able to represent and operate on positive and negative arbitrary precision numbers. The arbitrary precision numbers you are going to implement will be whole numbers; there will be no fraction part. Also all operations should result in whole numbers; so “5 / 3” will be equal to “1”, “4 ^ -2” will be equal to “0”.

When there is a division by zero, the result should be NaN (not a number) and any operation with at least one NaN operator should result as NaN.

Your program should evaluate the arithmetic expressions given in a custom stacked notation and write the results to the output file. For I/O operations, you should use the I/O streams library of C++.

The arithmetic expressions can be evaluated using the stack data structure. As one of the aims of this experiment is to make use of templates, you are expected to write your own stack implementation using templates. Also when evaluating expressions, you should use separate stacks for operators and APNs (If your APN class is named Apn, Your APN stack should be of the type Stack<Apn>).

An arithmetic expression in the custom stacked notation consists of N operators after N+1 numbers. The algorithm to evaluate expressions in the custom stacked notation is as follows:

1. Take an operator from the end of the operator sequence, and two numbers from the end of the number sequence, and perform the operation according to the operator (addition, subtraction …).
2. Add the result of the operation to the end of the number sequence.
3. Return to step 1 until there are no more operators left.

Here is an example of an arithmetic expression and how it looks after each iteration:

```
3    4    7    5    +    *    -
3    4    2    +    *
3    8    +
11
```

[2] From http://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic

3.1. EXECUTION

The name of the compiled executable program should be “apncalc”. Your program should read input/output file names from the command line, so it will be executed as follows:

apncalc [input file name] [output file name]

You can see sample input and output in ftp site. The program must run on DEV (dev.cs.hacettepe.edu.tr) UNIX machines. So make sure that it compiles and runs in one of the UNIX lab. If we are unable to compile or run, the project risks getting zero point. It is recommended that you test the program using the same mechanism on the sample files (provided) and your own inputs. You must compare your own output and sample output. If your output is different from the sample, the project risks getting zero point, too.

3.2. INPUT/OUTPUT FORMAT

In the input file, every line will represent a separate arithmetic expression. There will be whitespace (spaces and/or tabs) between operators and numbers. You do not need to check the input for syntax or semantic errors, every line will be a meaningful expression. An example input file can be given as follows:

```
49    13    +

5     3     ^
12    -5    +
-5    -34   -4    *     -
9     6     -8    /     +
-11   55    ^
5     0     6     2     3     +     /     -     *
```

Your program should write the results of the expressions to the output file, so that each line in the output file will contain the result of the expression in the corresponding line of the input file. Given the example input file above, the output file should be:

```
62
125
7
150
-4
-189059142471278104187151458457431977844930124660323803405
NaN
```

3.3. DESIGN EXPECTATIONS

You must perform a proper object oriented design for your solution. Writing spaghetti, or even structured code, without using Object Orientation or C++ features could render your entire assignment “unacceptable” and make it subject to huge (if not complete) grade loss.

3.4. VALID PLATFORMS

Your code will be compiled against gcc version 3.4.5. You should not assume the availability of any other non- standard libraries/features. If you use a different compiler, it is your responsibility to ensure that your code has no compilation issues with the specified version of gcc.

3.5. DEVELOPMENT ENVIRONMENT

You are advised to use the Eclipse CDT (C/C++ Development Tools) platform to develop your project. You can find instructions on how to configure CDT for your platform at http://max.berger.name/howto/cdt/.

4. EVALUATION

4.1. REQUIRED FILES

You should create and submit a ZIP archive in the following structure for evaluation. An invalid structured archive will cause you partial or full score loss.

You are required to submit a Makefile[3], which will be used to compile your program to generate the apncalc executable.

Directory	Files	Description
Source	*.cpp, *.h, Makefile	Program source/header files and Makefile
Report	*.pdf	Your report (Only pdf format is accepted)

4.2. REPORTS

You must write a report which is related to your program. (Report Format: ftp://ftp.cs.hacettepe.edu.tr/pub/dersler/genel/FormatForLabReports.doc).

LAST REMARKS:

- The output of your program will be graded automatically. Therefore, any difference of the output (even a smallest difference) from the sample output will cause an error and you will get 0 from execution. Keep in mind that a program that does not work 100% right is a program that works wrong.
- Regardless of the length, use **UNDERSTANDABLE** names to your variables, classes and functions.
- Write **READABLE SOURCE CODE** block
- **You will use online submission system to submit your experiments. https://submit.cs.hacettepe.edu.tr/ Deadline is: 23:59 pm. No other submission method (such as diskette, CD or email) will be accepted.**
- Do not submit any file via e-mail related with this assignment.
- **SAVE** all your work until the assignment is graded.
- The assignment must be original, **INDIVIDUAL** work. Duplicate or very similar assignments are both going to be punished. General discussion of the problem is allowed, but **DO NOT SHARE** answers, algorithms or source codes.
- You can ask your questions through course's web page and you are supposed to be aware of everything discussed in the newsgroup: **https://dersler.cs.hacettepe.edu.tr**.

REFERENCES

1. Fundamentals of Data Structures, Ellis Horowitz
2. C++: The Complete Reference, Herbert Schildt

[3] Make file example http://mrbook.org/tutorials/make/