

Advanced Programming

- Optional Exercises -

Prof. Fromm, Prof. Lipp

University of Applied Sciences

Faculty for Electrical Engineering and Information Technology

(Version 2015)

Content

1	Introduction	3
1.1	Motivation	3
1.2	Some tips.....	3
1.3	Organization	4
1.4	Development Tools.....	4
1.4.1	Tool Installations.....	4
1.4.2	Lernzentrum	4
1.5	Setting up a first Project.....	5
1.5.1	Naming Conventions	5
1.6	Additional Help.....	6
2	Simple Exercises	7
2.1	CCoordinate	8
2.2	Set of Measurement Values.....	12
3	Medium Exercises.....	15
3.1	CComplex: Operator Overloading.....	16
3.2	Calculator for Fractions.....	19
3.3	CBank: Class design.....	24
3.4	CPhoneList: Libraries.....	27
3.5	Tests for LIFO Buffer and RPN Calculator	29
4	Challenging Exercises	32
4.1	XML Parser.....	33

4.2	CSort: Template Class.....	37
4.3	Combining Techniques	39

1 Introduction

1.1 Motivation

Learning to program is like learning to play a music instrument or trying to learn a new language. You will not become a great violin player by simply listening to Mozart or by studying partitions. You have to play the instrument. Of course the first hours of practicing will sound horrible and the frustration level surely can be high, but after some time, you will notice progress and maybe even start to like your instrument.

The same is valid for learning C++ or any other programming language - without practical exercise it will not work. The compiler errors will sound weird and you will spend hours looking for the “;” you have forgotten. But once your program finally runs you feel the proud of a real achievement.

Programming certainly is a highly complex challenge and especially if you have not programmed before a time consuming learning process, but by today, it must be considered as an important and mandatory skill of every engineer. The areas of programming are wide:

- Programming of embedded controllers for consumer electronics, automotive, telecommunication, medical or military
- Development of factory simulation systems using Matlab
- Coding of numerical algorithms using digital signal processors
- And even programming of business processes on the management level

Without the ability to design and implement software the career opportunities for most engineers must be considered significantly lower compared to an engineer who has those skills.

The good news however:

- One you have learned and understood the concepts of one programming language, the next one will be much easier
- This set of exercises will help you to get a good understanding of C++

1.2 Some tips

The complexity levels of the exercises increase. For the first couple of exercises, only single classes have to be developed. Make sure, that you have understood the exercises before moving on to the next one. Check proposals for additional training. Solve these additional exercises before moving on to the more difficult ones.

If you get stuck invite one of your colleagues for a review of your solution. It is ok to cooperate; this will help you to find problems faster and will bring in new ideas on how to solve the individual tasks. But remember: Working in a team does not mean that one is writing the code and the rest simply receives a copy. Everybody has to pass the course individually.

1.3 Organization

Every student has to solve all mandatory exercises and present the results in order to get the course certificate. The mandatory exercises are available in Moodle. **The mandatory exercises will be marked and have to be passed in order to complete the module.**

This document contains additional exercises, which may be solved on a voluntary basis, if you feel that you need extra exercise in a special area. Special lab sessions will be provided to support you if needed. Again, the attendance of these extra lab sessions is voluntary but recommended, if you have problems with the subject. **The voluntary exercises will not be marked.**

1.4 Development Tools

All programs you have to develop are console applications. The default tools you should use are **Eclipse** and **Borland Together Architect 1.1**. You may use different tools and even draw the UML diagrams by hand (which is mandatory for the first exercises anyway) but you will have to accept certain limitations doing this. This is described in more detail in the chapter “Alternative Freeware Tools” later on.

1.4.1 Tool Installations

All tools including student licenses can be found on the central webpage:

- Link: <http://tools.eit.h-da.de>
- Login: student
- Password: fbeit

1.4.2 Lernzentrum

All tools are already installed on the Lernzentrum PC's available for every student at the university.
Private PC Installation

Important: All provided licenses are licenses which may be used for studying purposes only! It is within the user's responsibility to ensure, that they are not used for any other purposes or passed on to anybody!

1.5 Setting up a first Project

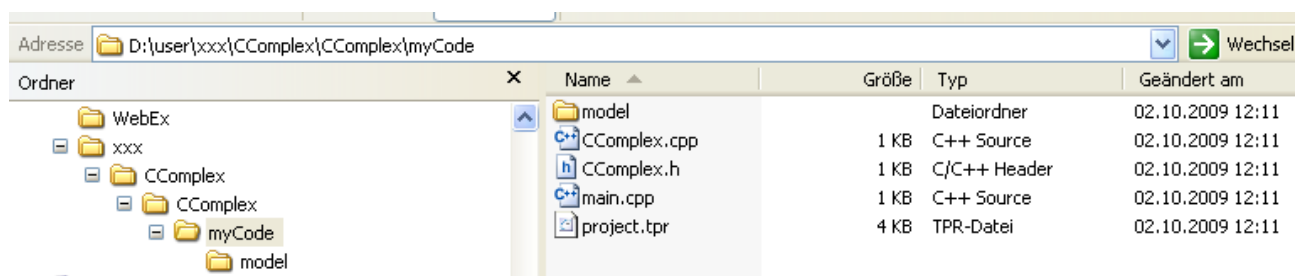
For setting up a new project, you may use one of the following approaches:

- Using the Project Template
- Use a copy of an empty project
- Manual setup

The first choice should be the use the Project Template, the other approaches should only be used if this does not work (e.g. you are using a different set of tools). It is highly recommended to use the project structure of the Template Project when setting up an own project

Please note the conventions for the project structure:

- All source files are stored in the subfolder mycode
- All model files are stored in the subfolder mycode\model



This has the following advantages:

- Your code files are separated from the Eclipse project files and can easily be copied, e.g. to continue working on a different PC.
- When opening Together, all code files are automatically parsed and become part of the model.

1.5.1 Naming Conventions

Please use the following naming conventions inside your project

- A class has the prefix C and starts with a capital letter, e.g. CMyClass
- One class per header (.h) and source (.cpp) file
- Name of the header and source files are equal to the name of the class (e.g. CMyClass.h)

- Member attributes have the prefix `m_` to distinguish them from parameters and local variables. They start with a small letter. Capital letters are used to separate words e.g. `m_myColorAttribute`
- Identify pointer variables by adding an additional `p` before the name, e.g. `m_pMyPointer`

1.6 Additional Help

The slides and examples presented in the lecture will help you with solving the exercises. In addition check the recommended literature and web tutorials.

2 Simple Exercises

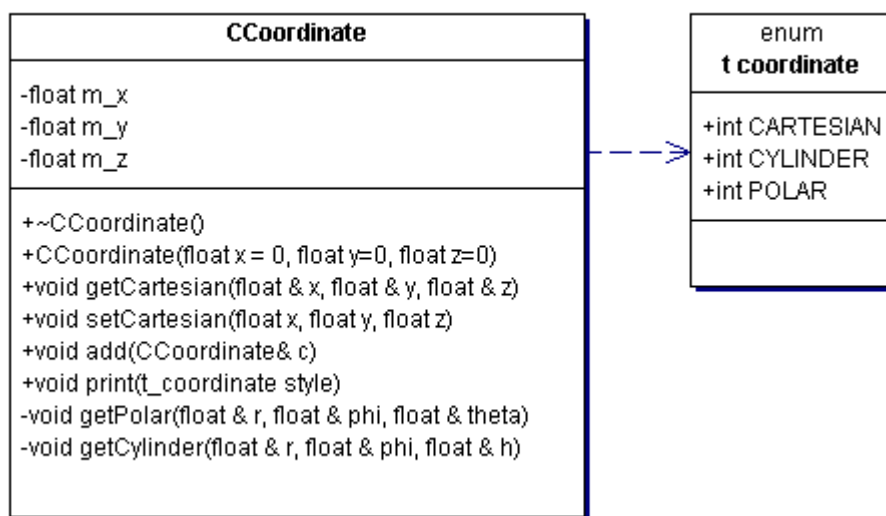
This section contains simple exercises which are intended to get you familiar with basic C++ syntax. You get a detailed description of what you have to do and the project scope is limited to a single class only.

2.1 CCoordinate

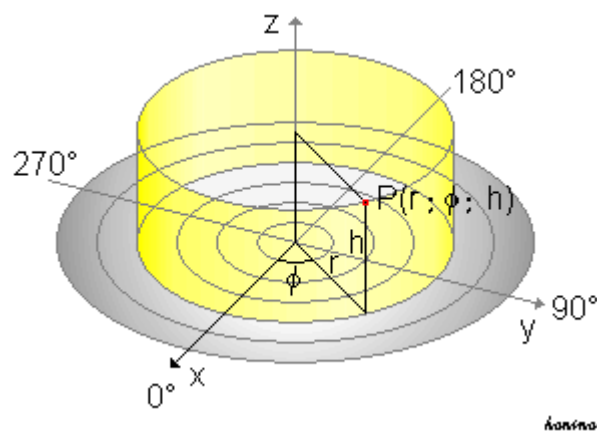
Develop a class which stores three coordinate values and represents them in either Cartesian, cylinder or polar format.

Hint: Compile the program after every task and correct the errors before continuing with the next task.

UML Diagram



Cylinder Coordinates

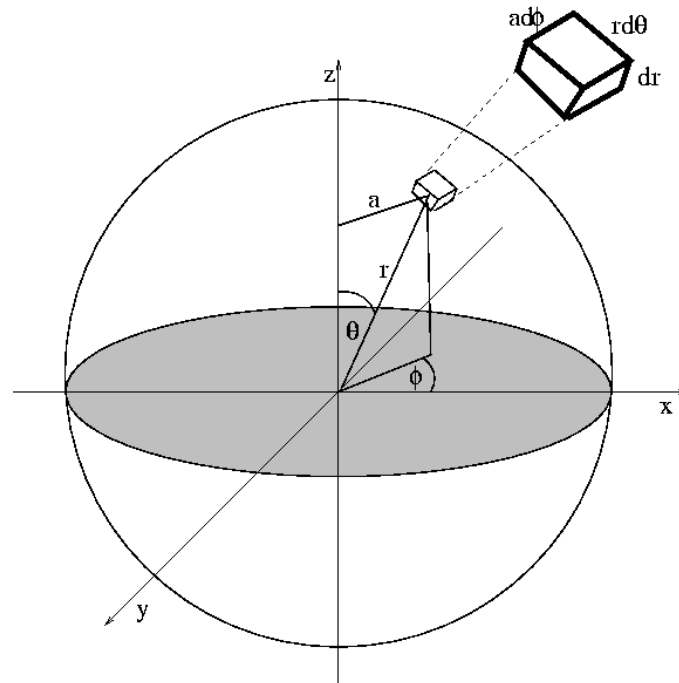


$$r = \sqrt{x^2 + y^2} \quad (2-1)$$

$$\varphi = \begin{cases} \arctan \frac{y}{x} & x > 0 \\ \arctan \frac{y}{x} + \pi & x < 0, y \geq 0 \\ \arctan \frac{y}{x} - \pi & x < 0, y < 0 \\ +\frac{\pi}{2} & x = 0, y > 0 \\ +\frac{\pi}{2} & x = 0, y < 0 \end{cases} \quad (2-2)$$

$$h = z \quad (2-3)$$

Polar Coordinates



$$r = \sqrt{x^2 + y^2 + z^2} \quad (2-4)$$

$$\varphi = \arctan 2(y, x) = \begin{cases} \arccos \frac{x}{\sqrt{x^2 + y^2}} & y \geq 0 \\ 2\pi - \arccos \frac{x}{\sqrt{x^2 + y^2}} & y < 0 \end{cases} \quad (2-5)$$

$$\theta = \frac{\pi}{2} - \arctan \frac{z}{\sqrt{x^2 + y^2}} \quad (2-6)$$

Implementation

Note: Write the answers to the questions as comments into the code

- a) Declare the class based on the UML diagram in the file CCoordinate.h.
- b) Implement the constructor in the file CCoordinate.cpp. Print the address of the object using the **this** Pointer as well as all attribute values.
- c) Implement the destructor. Print the address of the destroyed object.
- d) Implement the setCartesian() and getCartesian() operation. The setCartesian() operation sets the attributes to the passed parameter values, the getCartesian() operation returns the attributes using the parameter variables (call by reference). What happens if you use references for the setCartesian()? What happens if you do not use references for getCartesian?
- e) Implement the add() function, which adds the passed parameter values to the attributes. Compare this function with the setCartesian() operation. What do you notice?
- f) Implement the getCylinder() operation using the formulas above. Include the library <math.h> for the trigonometric functions.
- g) Implement the getPolar() operation using the formulas above. Check the functionality of the atan2 function compared to the atan function.
- h) Implement the print operation. Use a switch case statement to print the correct coordinate format based on the passed parameter.
- i) Define 3 objects in the main.cpp file:


```
CCoordinate c1(4, 4, 2);
CCoordinate c2;
CCoordinate c3(-4);
```

- j) Test the print operation.

```
c1.print(CARTESIAN);
c1.print(CYLINDER);
c2.print(CYLINDER);

c1.print(POLAR);
c2.print(POLAR);
c3.print(POLAR);
```

- k) Check the results on correctness. Extend the code in such a way, that a warning is generated for all illegal parameter combinations and set the resulting calculated value to 0. Hint: check the provided formulas for critical values and missing definitions.

- l) Test the add operation. Explain the generated messages from the constructor and destructor.

```
c1.add(c3);
c2.add(CCoordinate(1,2,3));
c1.print(CARTESIAN);
c2.print(CARTESIAN);
```

Generated printout:

```

d:\user\150_HDA\60_Lehre\40_Vorlesungen\20_MB01_C++\40_Lab\WS_APT\CCoordinateD...
Constructor
=====
Constructor generates object at 0012FF4C
Value = (4, 4, 2)
Constructor generates object at 0012FF3C
Value = (0, 0, 0)
Constructor generates object at 0012FF2C
Value = (-4, 0, 0)
Printing
=====
Cartesian representation (x,y,z) = (4, 4, 2)
Cylinder representation (r,phi,z) = (5.65685, 45, 2)
WARNING: Undefined result. Setting phi to 0
Cylinder representation (r,phi,z) = (0, 0, 0)
Polar representation (r,phi,theta) = (6, 45, 70.5288)
WARNING: Undefined result. Setting all angles to 0
Polar representation (r,phi,theta) = (0, 0, 0)
Polar representation (r,phi,theta) = (4, 180, 90)
Adding
=====
Constructor generates object at 0012FED8
Value = (1, 2, 3)
Destructor destroys object at 0012FED8
Cartesian representation (x,y,z) = (0, 4, 2)
Cartesian representation (x,y,z) = (1, 2, 3)
Drücken Sie eine beliebige Taste . . . _

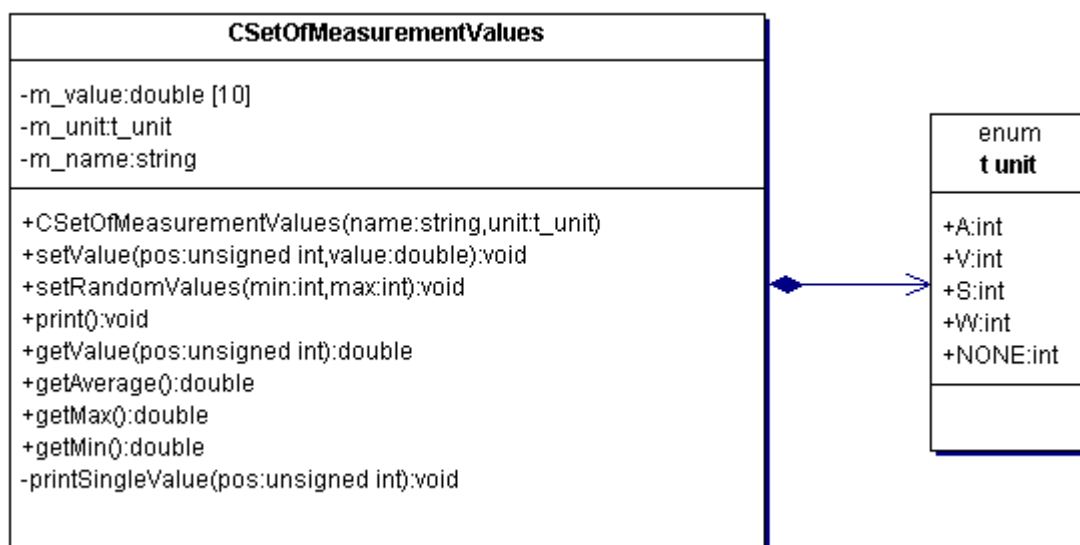
```

2.2 Set of Measurement Values

Develop a class which stores a set of measurement values. Do not forget to comment your code.

Hint: Compile the program after every task and correct the errors before continuing with the next task.

UML Diagram



Implementation

Note: Write the answers to the questions as comments into the code

- Declare the class based on the UML diagram in the file `CSetOfMeasurementValues.h`. To support extensibility, provide a constant (using `#define`) `MAXVALUE` with the value 10 and a constant `NOVALUE` with the value -9999.
- Implement the constructor in the file `CSetOfMeasurementValues.cpp`. All values of the array are initialized with `NOVALUE`. Set `m_name` and `m_unit` to the given parameters. Provide sensible default values for all parameters of the constructor. Print the address of the object using the `this` Pointer.
- Implement the method `setValue`, setting `m_value` at the given position to the corresponding parameter value. The values of the `m_unit` and `m_name` attributes are not changed. In

case the parameter `pos` is out of the range of the array, an error message shall be printed and no value may be changed.

- d) Implement the method `printSingleValue()`, printing the value of `m_value` at the given position as well as the unit, using the following strings: Ampere, Volt, seconds, Watt. Hint: Use a switch statement for the units.
- e) Implement the method `getValue()`. In case the position parameter is out of range, an error message shall be printed and the value `NOVALUE` is returned.
- f) Test the class by creating an object `myVoltageValues` in `main.cpp`, having the name "MyVoltageValues" and the unit V.
- g) Implement the method `getMin()`. This function only considers valid measurement values, i.e. values not having the value `NOVALUE`. The minimum value is returned. In case no valid value is contained in the array, the value `NOVALUE` shall be returned.
- h) Implement the method `getMax()` following the same logic like `getMin`.
- i) Implement the method `getAverage` following the same logic like the previous methods, i.e. the average value of all valid values is calculated.
- j) Implement the method `print()`, which prints all valid values and the min, max and average value of the array. Hint: Do not copy any code, use the already implemented methods instead.
- k) Test the code by setting element 1 to the value 10 and element 3 of `myVoltageValues` to 20. Print the object `myVoltageValues`. Hint: Check the output at the end of the page.
- l) Test the code by setting the value at position 30 to 20 and try to read the value from `pos 77`.
- m) Implement the method `setRandomValues()`. This method sets all values of the array to a value between min and max, having 1 decimal digit accuracy. Hint: Use the function `rand()` of the standard libraries and seed it with the current time value.
- n) Create a second object `myCurrentValues` (Unit A, name "MyCurrentValues") and set all values to random value between 0 and 100.
- o) Print the newly created object.

Sample Output:

```
Testing a first object.....
Created object at : 0x28fea8
myVoltageValues:
  Value[1]: 10 Volt
  Value[3]: 20 Volt
  Max Value: 20
  Min Value: 10
  Avg Value: 15
Testing some exceptions.....
ERROR setValue: Position out of bounds
ERROR getValue: Position out of bounds
Returning a value: -9999
Created object at : 0x28fe50
myCurrentValues:
  Value[0]: 4.1 Ampere
  Value[1]: 46.7 Ampere
  Value[2]: 33.4 Ampere
  Value[3]: 50 Ampere
  Value[4]: 16.9 Ampere
  Value[5]: 72.4 Ampere
  Value[6]: 47.8 Ampere
  Value[7]: 35.8 Ampere
  Value[8]: 96.2 Ampere
  Value[9]: 46.4 Ampere
  Max Value: 96.2
  Min Value: 4.1
  Avg Value: 44.97
```

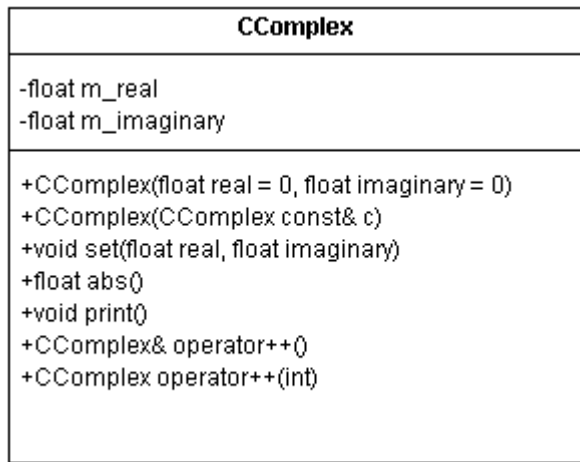
3 Medium Exercises

These exercises are slightly more challenging. You will have to deal with more than one class and use more advanced language feature like operator overloading, dynamic memory and similar.

3.1 CComplex: Operator Overloading

Develop a class which stores complex numbers of the format $c = a+bi$ and supports the unary operations $+$, $-$, $*$, $/$ as well as the unary operations $c++$ and $++c$.

UML Diagram



Implementation

- Declare the class as defined in the UML diagram.
- Implement the constructor, considering the given boundaries below. In case one of the values violates these boundaries, both values shall be set to 0

-100 < m_real < 100

-100 < m_imaginary < 100

Print the address of the object as well as the attribute values on the screen.

- Implement the set function, considering the boundary values defined above.
- Implement the abs operation, which calculates the absolute value of the real number
- Implement the print operation, which prints the complex number in the format $m_real + i m_imaginary$
- Test the constructor by defining the following objects and printing them:

```
CComplex c1;  
CComplex c2(1,3);
```

```
CComplex c3(4);
CComplex c4(c3);
```

- g) Implement overloaded operators for the +, -, * and / operation, supporting operations between 2 complex numbers as well as operations between a float and a complex number.

Hint: You will need 3 overloaded operator per operation, e.g. complex + complex, complex + float, float + complex. Once you have implemented complex + complex, you should use the Constructor to transform the float variable into a complex number.

- h) Implement the overloaded << Operator, which prints the complex number in the following format:

a + ib	b > 0
a - ib	b < 0
a	b = 0

Hint: The overloaded << operator has the following interface

```
friend ostream& operator <<(ostream& out, CComplex& c);
```

and the following implementation:

```
ostream& operator <<(ostream& out, CComplex& c)
{
    out << "your text" << c.yourVariable;
    return out;
}
```

- i) Test the overloaded operators by adding some calculations and printouts in the following format. Make sure that all operators are tested:

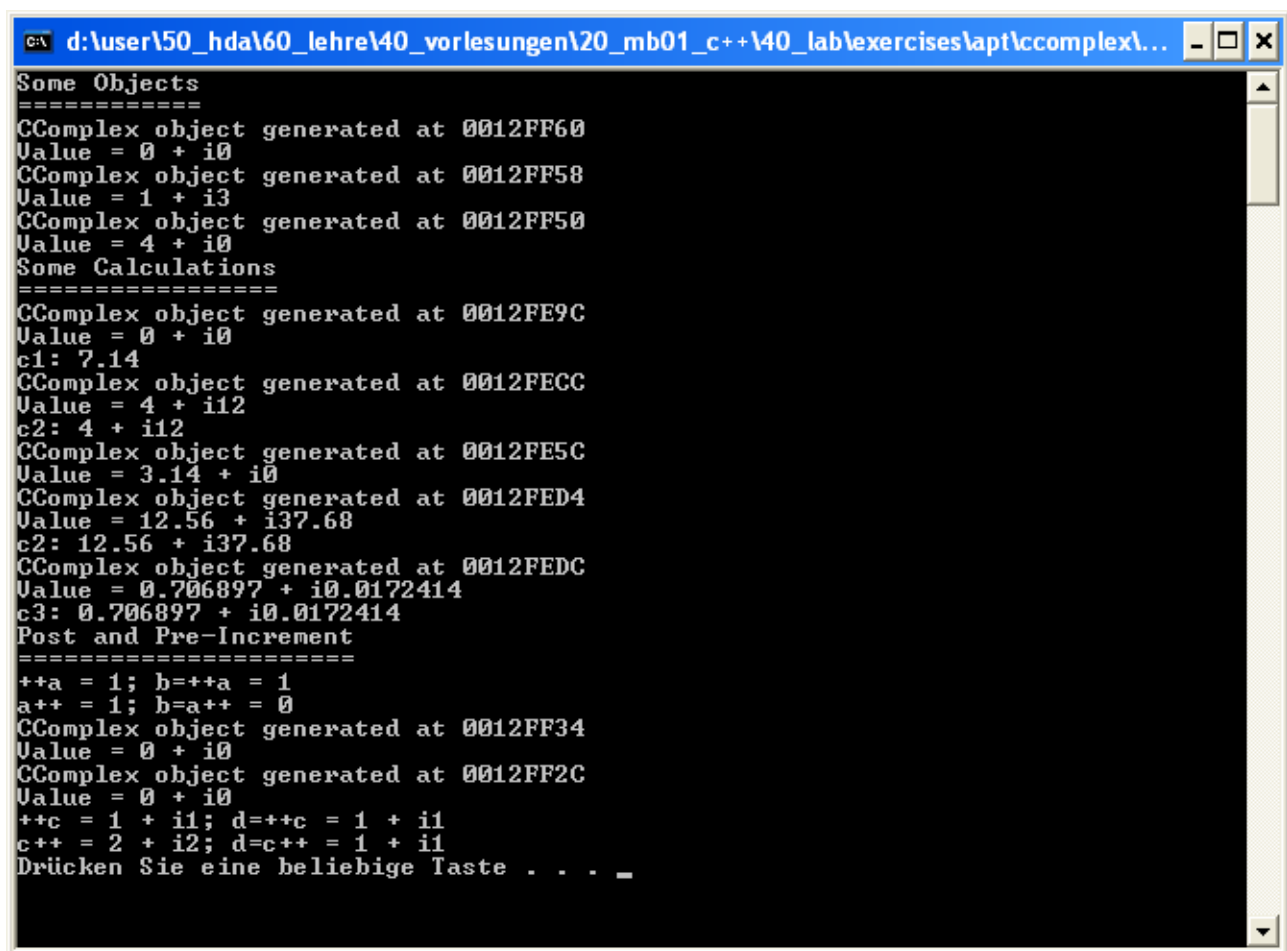
```
float x = 3.14;
c1 = c3+x;
cout << "c1: " << c1 << endl;
// Add more testcases as needed
```

- j) Implement the post and pre-increment operators, which add the value 1 to both the real and the imaginary part of the complex number. Add the following testcase to your program and compare the implemented operators with the behavior of the standard post- and pre-increment operators (e.g. for an integer variable).

```
CComplex c;
```

```
CComplex d;  
  
d = ++c;  
cout << "++c = " << c << "; d=++c = " << d << endl;  
  
d.set(0,0);  
d = c++;  
cout << "c++ = " << c << "; d=c++ = " << d << endl;
```

Example printout:

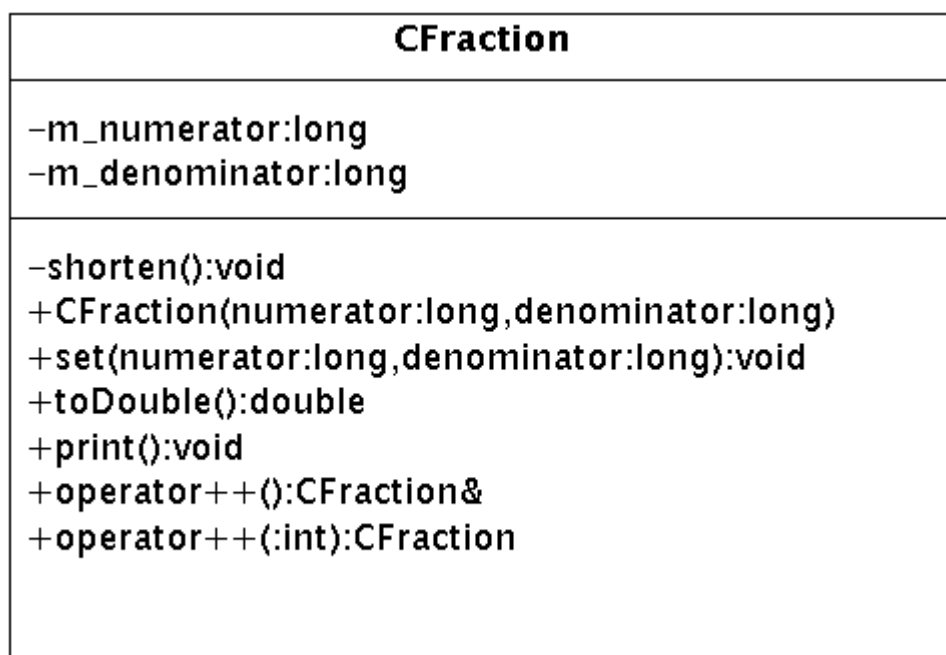


```
C:\ d:\user\150_hda\60_lehre\40_vorlesungen\20_mb01_c++\40_lab\exercises\lapt\ccomplex\...  
Some Objects  
=====  
CComplex object generated at 0012FF60  
Value = 0 + i0  
CComplex object generated at 0012FF58  
Value = 1 + i3  
CComplex object generated at 0012FF50  
Value = 4 + i0  
Some Calculations  
=====  
CComplex object generated at 0012FE9C  
Value = 0 + i0  
c1: 7.14  
CComplex object generated at 0012FECC  
Value = 4 + i12  
c2: 4 + i12  
CComplex object generated at 0012FE5C  
Value = 3.14 + i0  
CComplex object generated at 0012FED4  
Value = 12.56 + i37.68  
c2: 12.56 + i37.68  
CComplex object generated at 0012FEDC  
Value = 0.706897 + i0.0172414  
c3: 0.706897 + i0.0172414  
Post and Pre-Increment  
=====  
++a = 1; b=++a = 1  
a++ = 1; b=a++ = 0  
CComplex object generated at 0012FF34  
Value = 0 + i0  
CComplex object generated at 0012FF2C  
Value = 0 + i0  
++c = 1 + i1; d=++c = 1 + i1  
c++ = 2 + i2; d=c++ = 1 + i1  
Drücken Sie eine beliebige Taste . . . _
```

3.2 Calculator for Fractions

Develop a class which stores fractions using the format $f = \frac{n}{d}$ and supports the binary operations $+$, $-$, $*$, $/$ as well as the unary operations $c++$ and $++c$.

UML Diagram



Implementation

1. Declare the class as defined in the UML diagram. Note that Together does not show default parameter values in the class diagram. The constructor is fully specified as `+CFraction(numerator:long=0, denominator:long=1)`
2. Implement the `shorten()` method. It's purpose is to "normalize" the fraction, i. e. make sure that nominator and denominator have no common divider greater than 1 (a quick research using the Internet will show you how this can be done easily). The fraction is to be kept in a normalized form by all methods and functions described below.
3. Implement the constructor. What special condition must be observed? How can you handle it? (Supply the answer as comment in the code and implement appropriately.)
4. Implement the `set` method. What special condition must be observed? How can you handle it? (Supply the answer as comment in the code and implement appropriately. Note that the

best approach to handle the condition in this method differs from the best approach for handling the problem in the constructor.)

5. Implement the `toDouble()` method, which calculates the value of the fraction as floating point number.
 6. Implement the `print()` method, which prints the fraction number in the format `CFraction[m_nominator=value; m_denominator=value]`
 7. Test the constructor by defining the following objects and printing them:

```
CFraction f1;  
CFraction f2(1, 3);  
CFraction f3(24, 3);
```
 8. Add code for testing `toDouble()` using `f2`.
 9. Implement overloaded operators for the `+`, `-`, `*` and `/` operation, supporting operations between 2 fractions as well as operations between an integer and a fraction.
Hint: You will need 3 overloaded operators per operation, e. g. fraction + fraction, fraction + integer, integer + fraction. Choose the most efficient way to do each implementation.
 10. Implement the overloaded `<<` Operator, which prints the fraction as `"(nominator/denominator)"` (without trailing newline).
 11. Test the overloaded operators by adding some calculations and printouts in the following format. Make sure that all operators are tested:

```
cout << "(3/4)+(2/3) = "  
<< CFraction(3, 4) + CFraction(2, 3) << endl;  
// Add more testcases as needed!
```
 12. Whenever possible, choose values that also show that normalization (shortening) works (the above example does not fulfill this criterion).
 13. Implement the post and pre-increment operators. Add the following test case to your program and compare the implemented operators with the behavior of the standard post- and pre-increment operators (e. g. for an integer variable).

```
CFraction f(1, 3);  
CFraction g;  
g = ++f;  
cout << "++f == " << f << "; (g=++f) == " << g << endl;  
f.set(1, 3);  
g = f++;  
cout << "f++ == " << f << "; (g=f++) == " << g << endl;
```
-

Extension - An RPN Calculator

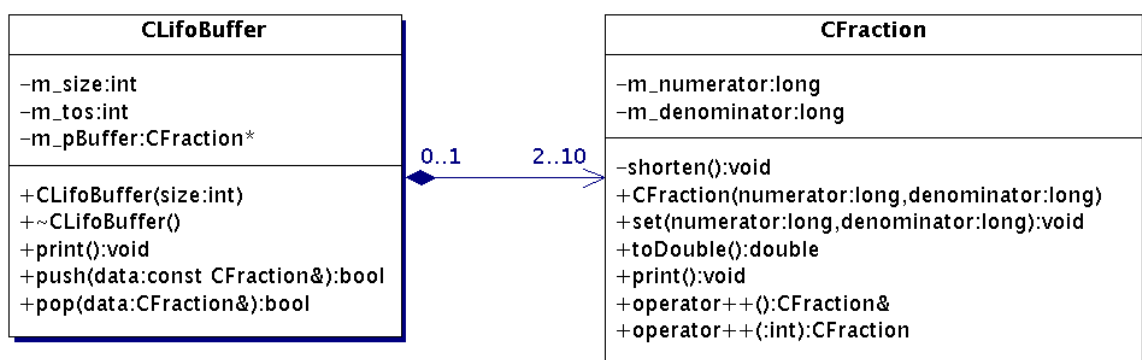
Now you will implement a calculator for evaluating expressions of fractions noted in reverse polish notation.

Reverse polish notation (RPN, also known as postfix notation) is a mathematical notation that puts the operator behind the operands. In contrast, the notation we are accustomed to is called infix notation, because the operator is put between the operands. As a simple example, consider infix noted “3 + 4” which becomes “3 4 +” using postfix notation (see also http://en.wikipedia.org/wiki/Reverse_Polish_notation).

An interesting point about postfix notation is that no parentheses are required for complex expression. Consider the more complex example “(3 + 4) * (5 + 6)”. Using RPN this becomes “3 4 + 5 6 + *”. This simplified processing led to the implementation of RPN in early hand held calculators (burdening the user with the task of rearranging the expression). The German Wikipedia page about RPN shows a picture of such a calculator (http://de.wikipedia.org/wiki/Umgekehrte_Polnische_Notation).

The first step in the implementation of the RPN calculator is providing the calculator’s memory. The memory of an RPN calculator is organized as a stack or LIFO buffer (Last In First Out) that stores the operands until the operation is chosen. The UML diagram below shows the LIFO buffer.

UML Diagram

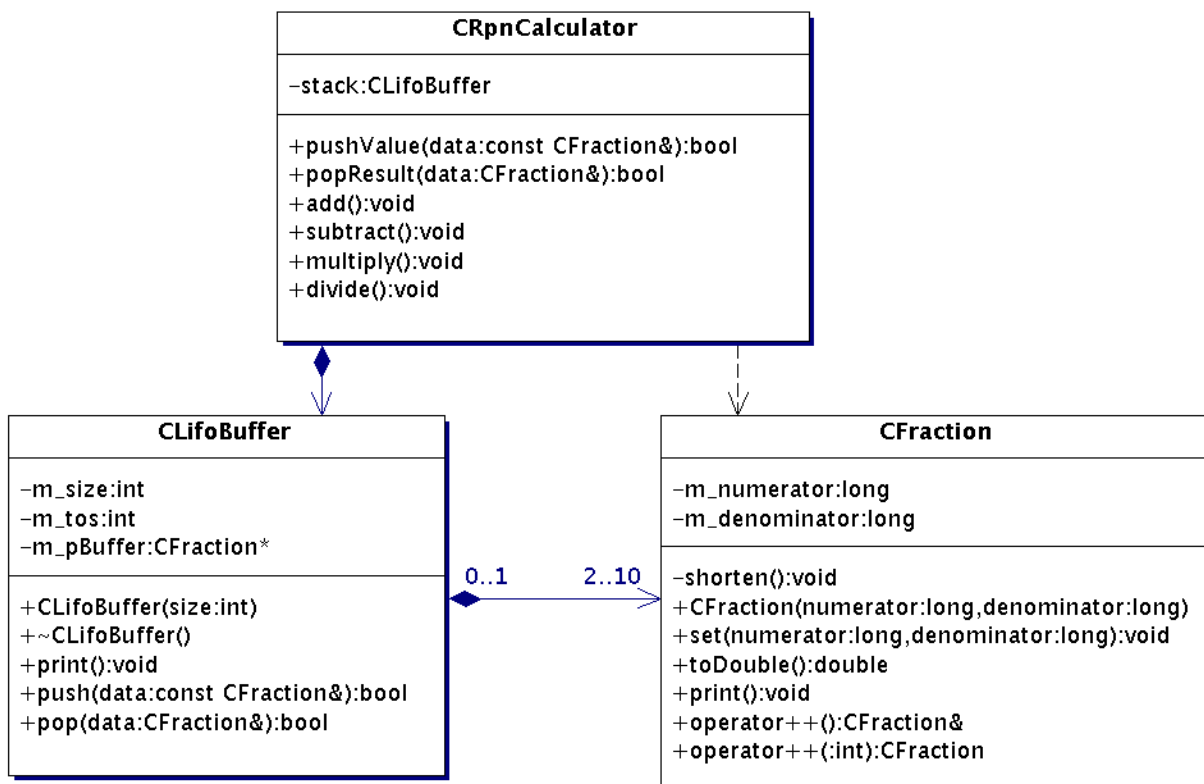


Implementation

14. Copy the existing class `CFraction` to the new project.
15. Define the class `CLifoBuffer` specified in the UML diagram.

16. Implement the constructor. Allocate the required memory. Set the pointer to the top of stack (`tos`) to -1. Using this invalid index indicates that the stack is empty. Print all attributes and the content of the buffer.
17. Implement the destructor. Make sure that all allocated memory is released.
18. Implement the `push()` operation. Check if there is still memory left. In case there is, increment the top of stack pointer and store the value.
19. Implement the `pop()` operation. Check if there is a value in the buffer. In case there is, return it and decrement the top of stack.
20. Implement the `print()` operation, printing the values of all attributes as well as the content of the buffer.
21. Test the first version of the class by generating a LIFO buffer object with 5 elements, adding three items, printing its content, retrieving the elements and printing the content again.

Now add a class that implements the actual RPN calculator as specified by the UML diagram below.

UML Diagram**Implementation**

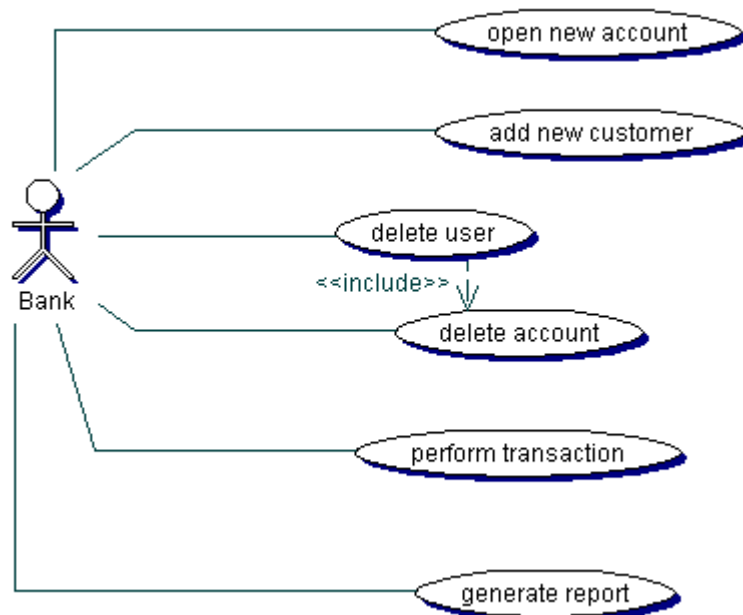
22. Add the new class `CRpnCalculator` to the project.
23. Implement `pushValue()` and `popValue()`. These methods delegate directly to the corresponding methods of the `stack` attribute (the LIFO buffer).
24. The methods `add()`, `subtract()`, `multiply()` and `divide()` perform the respective operations using the two top values on the stack and replace these values with the result.
25. Test the implementation. Create an instance of `CRpnCalculator`. Write a test program

$$\frac{\frac{5}{2} \cdot \frac{6}{5} + \frac{1}{2} - \frac{1}{3}}{5 + \frac{1}{6} + \frac{11}{4}}$$

that invokes its methods as necessary to evaluate and print the result. There may be only a single invocation of `popResult()` in your code.

3.3 CBank: Class design

Your task is to develop a bank simulation program which implements the following Use Cases.



Open New Account

The customer can open a new account with a specific amount of money. Every account is identified by a unique number (int), which is automatically generated by the program. If the customer does not exist, a warning is generated.

Delete Account

The account is deleted. All remaining money is destroyed.

Perform Transaction

The bank transfers a certain amount from the customer's account to another account.

Generate Report

The bank generates can get a list of all transactions from all the customer's account identifying the transaction date, transaction source and transaction target account as well as the transferred amount. Furthermore the list of customers and accounts is printed.

Add new Customer

The bank generates a file for a new customer.

Delete User

The bank may delete a customer record, which includes the deletion of all his accounts.

Technical requirements

- The maximum number of customers, transactions and accounts is defined when the bank object is generated
- Data may only be stored at one location. No redundant data is allowed.

Implementation

- a) Draw a class diagram identifying all classes, their attributes and operations and their relationships.
- b) Draw a sequence diagrams for the transaction operation
- c) Implement the classes
- d) Test the classes by performing the following actions:
 - Generate the customers "Paul Panther" and "Whoppie Goldberg".
 - Open new accounts for the customers. Add an account for the non-existing customer "Jonny Depp"
 - Perform some transactions
 - Delete an account
 - Delete a customer

Generate a report after every step and check the validity of the data.

Example Printout:

```
C:\WINDOWS\system32\cmd.exe
ERROR: Generating Account - Customer <Jonny Depp> not found!
Bank Customers....
  Paul Panther
  Whoppie Goldberg
Bank Accounts....
  123456 Paul Panther 100
  123457 Paul Panther 200
  123458 Paul Panther 200
  123459 Whoppie Goldberg 100
Bank Transactions....
...for customer: Paul Panther
  ...Account 123456
  ...Account 123457
  ...Account 123458
...for customer: Whoppie Goldberg
  ...Account 123459
=====

Bank Customers....
  Paul Panther
  Whoppie Goldberg
Bank Accounts....
  123456 Paul Panther 80
  123457 Paul Panther 220
  123458 Paul Panther 200
  123459 Whoppie Goldberg 100
Bank Transactions....
...for customer: Paul Panther
  ...Account 123456
15.11.08 20EURO to 123457
  ...Account 123457
  ...Account 123458
...for customer: Whoppie Goldberg
  ...Account 123459
=====

Deleting Customer: Paul Panther
Deleting Account : 123456
Deleting Account : 123457
Deleted Customer: Paul Panther who owned 2 accounts
Bank Customers....
  Found deleteted customer at position 0
  Whoppie Goldberg
Bank Accounts....
  Found deleted account at position <0>
  Found deleted account at position <1>
  Found deleted account at position <2>
  123459 Whoppie Goldberg 100
Bank Transactions....
...for customer: Whoppie Goldberg
  ...Account 123459
=====

Drücken Sie eine beliebige Taste . . . _
```

3.4 CPhoneList: Libraries

Your task is to implement a set of classes which reads a set of names and phone numbers from a file and transforms them into a “relational” format. In addition, the data representation is cleaned up and the data is checked on consistency, i.e. detection of double occupied numbers and empty records

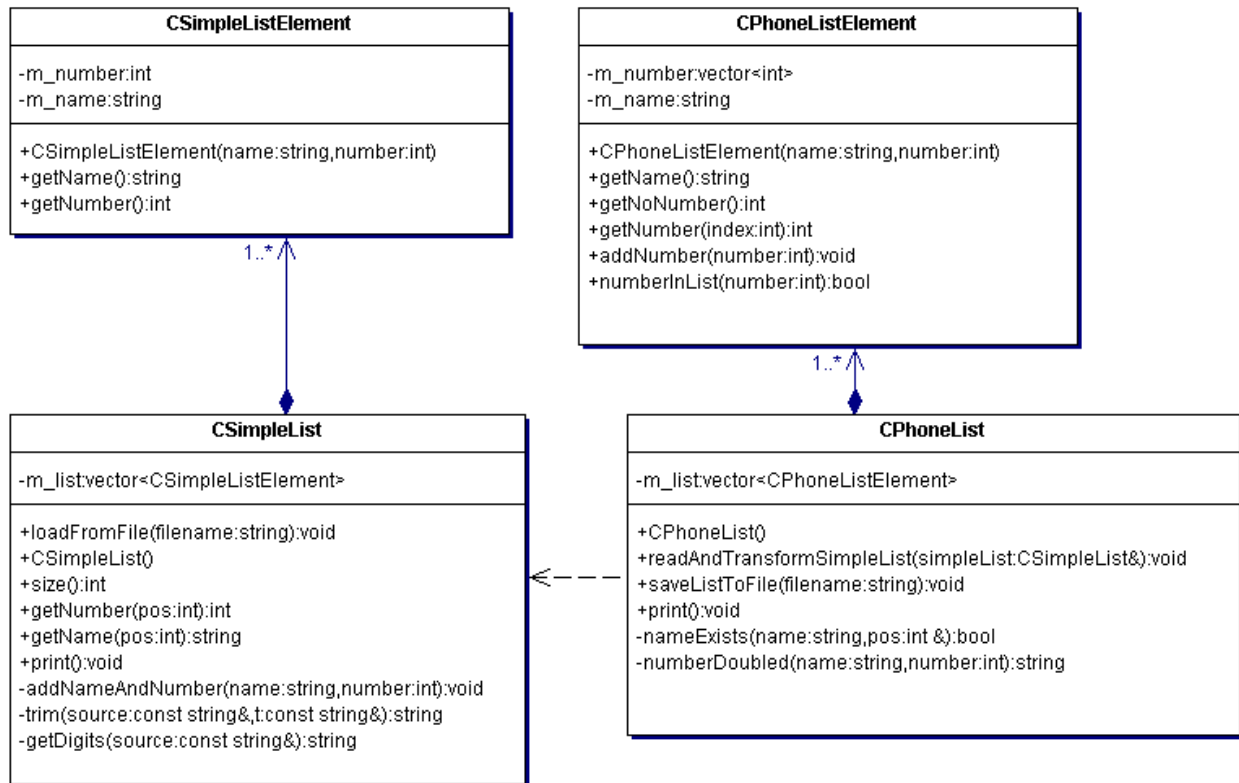
Input List:

```
Max Maier; 0173 12345
Wilhelm Knauser; 0973 45678
Max Maier; 06073-1234567
Berta Klunkerfrau; 06073-1234567
Willy Wichtig
```

Output List:

```
Max Maier; 17312345; 60731234567 (Conflict with Berta Klunkerfrau)
Wilhelm Knauser; 97345678
Berta Klunkerfrau; 60731234567 (Conflict with Max Maier)
Willy Wichtig; No number found
```

Note: The following UML diagram only shows the basic operations. You may add additional members as required.

UML Diagram**Implementation**

- Check which library functions can be used to solve the task.
- Draw an activity diagram for the method `CPhoneList::readAndTransformSimpleList()`
- Implement and test the classes by extending the provided Input List and checking the generated Output file.
- Add wrong lines to the Input List and check the generated Output List.

3.5 Tests for LIFO Buffer and RPN Calculator

In this exercise you will provide test cases for the LIFO buffer and RPN calculator developed in exercise 5, using the CppUnit library and the Coverage Tool presented in the lecture.

Note: fully (!) completed tasks a) – e) are considered 50%, fully completed tasks a) – g) are considered 80%.

Tasks

- a) Draw a use case diagram for the LIFO buffer and derive test cases from it. Document the test cases using the template presented in the lecture.
- b) There is an additional requirement for the LIFO buffer: there is a constraint for the size specified as “ $\{ 2 \leq m_size \leq 10 \}$ ”. If the constraint is violated, `m_size` is set to 5. Provide a hand written description of how you can apply the Boundary Values method to the LIFO buffer's constructor for finding relevant test cases.
- c) To find more test cases, consider the possible states of the LIFO buffer. The LIFO buffer can assume the three states “Empty”, “Partially Filled” and “Full”. The events are “evPush” and “evPop”. Use these events and appropriate guard conditions and draw a complete state diagram. Derive test case tables from the state diagram as described in the lecture.
- d) Add the CppUnit library to your RPN calculator project as shown during the lecture.
- e) Implement the test cases found in a), b) and c) as three `CppUnit::TestSuites` comprising the respective `CppUnit::TestCases`. Note that the LIFO class does not provide methods for directly querying state information in `CPPUNIT_ASSERT` statements. There is, for example, no method to check whether the buffer is empty. A check for “empty” can nevertheless be implemented by invoking the `pop` method and asserting that the returned value is `false`. The other states can be checked in similar ways using sequences of `pop` and `push` invocations¹.
- f) Check the test coverage using the coverage report provided by Eclipse. Add test cases until you have reached 100% C1 coverage. If you think that you cannot reach 100% coverage, provide a hand written explanation why you consider it impossible to reach a 100% coverage.

¹ Here's another example: to test that your buffer has really been created with a size of 5, do five pushes (they should all be successful) and then another final push which must then return false.

- g) Next, we want to test the RPN calculator. Consider the methods and the values passed to the methods. Find Equivalence classes for the input value. Provide a hand written rationale for the classes that you have formed.
- h) Implement an additional set of test cases as `CppUnit::TestCases` and a `CppUnit::TestSuite` that provide 100% C1 coverage for the `CRpnCalculator` class.
- i) Compare the time you spent initially on implementing the RPN calculator and the time you spent on developing and writing the test cases. What is the approximate ratio?

4 Challenging Exercises

The following set of exercises is more challenging. They use extended programming constructs like templates and more complex class relations like polymorphism and recursion. The algorithms also might be a little bit more complex.

4.1 XML Parser

In this exercise you will implement a simple XML Parser, which translates textual XML format into an internal representation.

The Extensible Markup Language (XML) is a markup language created to structure, store, and transport data by defining a set of rules for encoding documents in a format that is both human-readable and machine-readable².

For this exercise we define a simplified version of XML. In this version, an XML text (document) consists of ASCII characters with the following structure:

- An *element* consists of the sequence: `<tag>content</tag>`
- Every *content* is surrounded by a start and end tag
- A start-tag is a sequence of '<', a *tag* (name) and '>'
- An end-tag is a sequence of '<', '/', a *tag* (name) and '>'
- Start-tag and end-tag must have the same *tag* (name)
- A *tag* is a sequence of ASCII characters that starts with 'A'-'Z', 'a'-'z' or '_' and continues with characters 'A'-'Z', 'a'-'z', '_', '-', '0'-'9' or '.'
 - Examples of legal tags: `Sample`, `Item1to3`, `point-or-square`
 - Examples of illegal tags: `-not.good` (illegal start character), `not:good` (illegal character in tag)
- The *content* consists of an arbitrary mixture of *text* and nested *elements*
- *text* is a sequence of ASCII characters (including spaces, tabs, newlines) except '<'
- A document consists of an *element* and can span several lines.

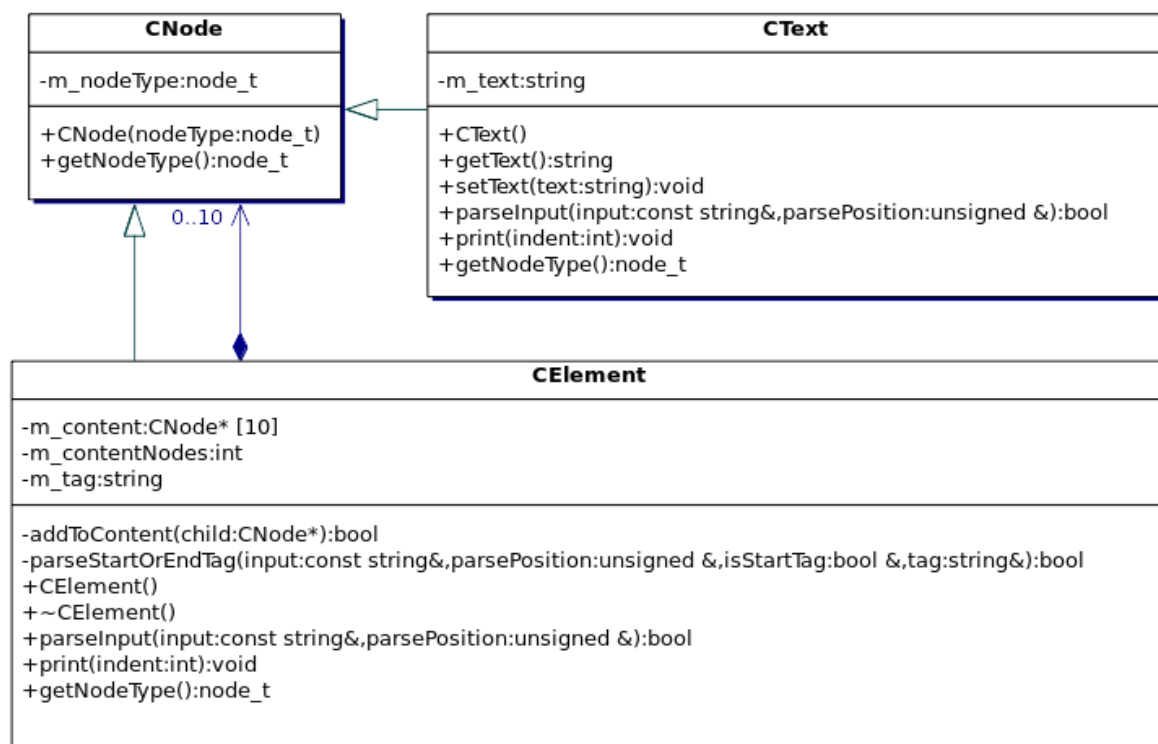
Consider the following example:

- `<top>`
- `<nested>some data</nested>`
- `more data`
- `</top>`

This document consists of an *element* that starts with a start-tag with *tag* (name) “top” and ends with an end-tag with *tag* (name) “top”. The content consist of whitespace *text* (a newline and two spaces), a nested *element*, more *text* (a newline, two spaces, “more data” and another newline). The nested *element* consists of a start-tag with *tag* (name) “nested”, its *content* and an end-tag with *tag* (name) “nested”. The *content* of the nested *element* consists of the *text* “some data”.

The goal is to parse the text into the **tree like data structure** described in the next section.

² Wikipedia

UML Diagram

Some hints regarding the classes and methods:

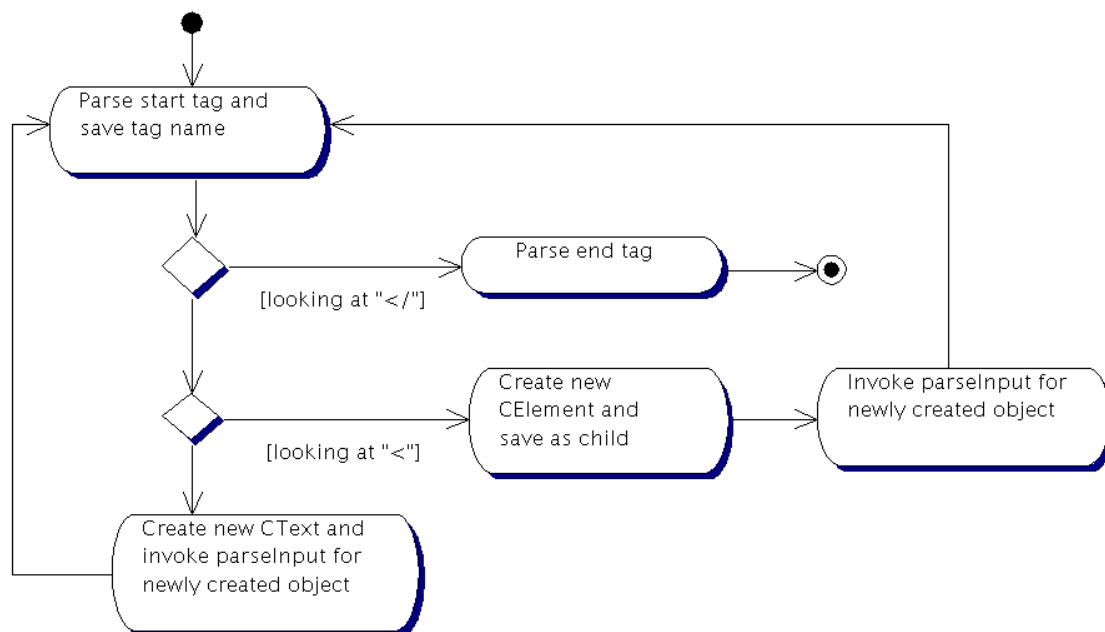
- `node_t` is defined as `enum node_t { ELEMENT, TEXT };`
- “CNode” is the base class for both “CText” and “CElement”, so a `CNode*` may point to a `CText` object or a `CNode` object. If you have a `CNode*`, you can use the method `getNodeType` to find out what it actually points to, and then cast it as appropriate to `CText*` or `CElement*`. (You'll learn about a better way to handle that situation – called dynamic polymorphism or late binding – later during the course.)
- The value of parameter “input” is always the complete input text. Parameter “parsePosition” indicates the next character to be processed within the text.
- All `parse...` methods consume characters from `input` by advancing the `parsePosition`. The methods return `false` if the input cannot be parsed, i. e. if it does not comply with the rules above (e. g. “<top>Hello</end>” or “<top>”).
- Method `parseStartOrEndTag` consumes a complete start or end tag including the opening “<” or “</” and the terminating “>”. E. g. invoking the method with “...<Hi>...”

and `parsePosition = 3` assigns `true` to parameter `isStartTag`, "Hi" to parameter `tag` and advances `parsePosition` to 7.

- Method `CText::parseInput` consumes all characters up to (but not including) an '<'. The consumed characters are stored in attribute `m_text`.
- "`m_content`" will store pointers to both `CElement` and `CText` objects.
- Methods `print(int indent)` output the class name and the values of the important attributes (see sample output below). Output is indented by the given number of spaces. If a `CElement` prints its content, it indents the contained `CTexts` and `CElements` by 2 additional spaces. Hint: Typecast the pointers to call the correct print method.

Approach to `CElement::parseInput`

The following activity diagram shows how to implement `CElement::parseInput` (error handling omitted).



As you can see in the diagram, you create new objects of type `CElement` and `CText` as “children” of `CElement` during parsing. The child relationship is implemented by storing pointers to the objects in `CElement::m_content`. As a child can be of type `CElement` or `CText`, we need the possibility to store either in `m_content`. Luckily, due to the inheritance relationship, `CElement` *is-a* `CNode` and `CText` also *is-a* `CNode`. So no matter whether the child is of type `CElement` or `CText`, we can store a pointer to the child as `CNode*`.

When you access an entry in `m_content` later, it is of type `CNode*` and you don't know whether it actually points to an object of derived type `CElement` or `CText`. This is where `getNodeType()` comes in. When you invoke `ptr->getNodeType()`, you'll get either `ELEMENT` or `TEXT` as return values. Depending on the result, you're then free to cast the `Node*` to `CElement*` or `CText*` and use the result to work with the specific methods of either `CElement` or `CText`.

What we do here is similar to invoking `malloc`, which returns a `void*`, and then casting this “pointer to nothing (or anything)” to a pointer to the type that you have allocated memory for. Here, we are a bit more specific, because we know that in any case it is a pointer to a `CNode` and depending on the actual type we cast it to a pointer to `CElement` or `CText`.

Sample usage

```
int main (void)
{
    unsigned pos = 0;
    CElement doc;
    if (!doc.parseInput("<top>\n  <nested>some "
        "data</nested>\n  more data\n</top>", pos)) {
        cout << "Parsing failed at position " << pos << endl;
    } else {
        doc.print(0);
    }

    return 0;
}
```

Sample output:

```
CElement[m_tag="top"]
  CText[m_text="
  "]
  CElement[m_tag="nested"]
    CText[m_text="some data"]
    CText[m_text="
more data
"]
```

Your tasks

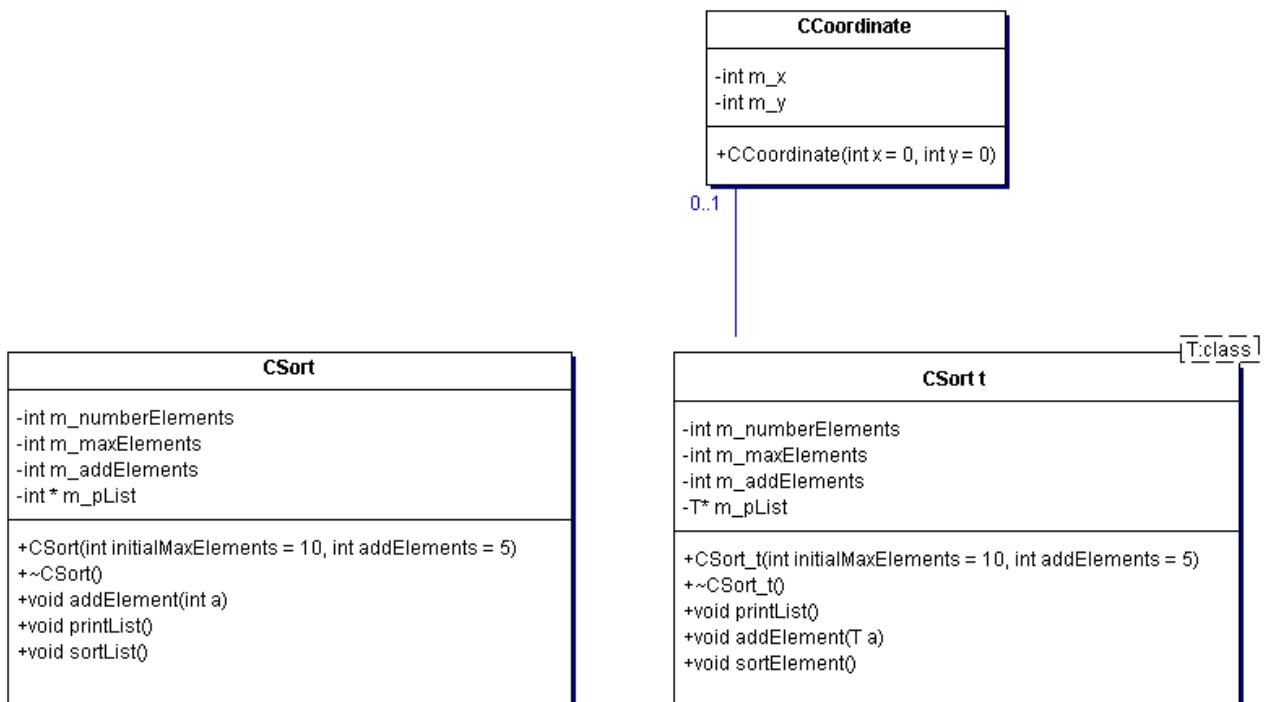
Implement and test all the classes. Provide at least

- a test case for the example used above,
- an extended example of your choice (>3 levels of nesting, at least one element with more than 2 nested elements) and
- additional test cases that show that parsing “illegal” input returns `false` for as many problematic cases as you can imagine.

4.2 CSort: Template Class

Your task is to implement a class, which is able to store any number of any object and to sort the list of these objects.

UML Diagram



Implementation

- a) In a first development step, implement the class `CSort`, which stores and sorts a list of integers.
 - The initial size of the list is equal to the value of `initialMaxElements` provided as parameter with the Constructor.
 - The current maximum size of the list is stored in the member `m_maxElements`.
 - If the list is full (`m_numberElements >= m_maxElements`), the size `m_maxElements` is increased by the value of `m_addElements` (provided as parameter of the constructor)

Hint: the required behavior is comparable to a STL container. You may implement any sorting algorithm you like, e.g. bubblesort, straight insertion, quicksort,...

- b) Test the class CSort. Add some elements (more than originally planned) to the class, print the list, sort the list, print the elements again
- c) Copy the content of the class CSort (.h and .cpp) to the class CSort_t.h. Change the code in such a way, that the class sorts any kind of object (the type passed as template parameter)
- d) Add a class CCoordinate, which stores 2d coordinate objects. A coordinate object is bigger than another coordinate object, if the absolute value is bigger. Provide all required overloaded operators to be able to use this class as parameter for the CSort_t class.
- e) Test the class CSort_t with similar testcases like the class CSort.
- f) Extend the test by adding the coordinate objects to a STL <list> container and sort it
- g) Generate a testcase which automatically generates a large number (100 and 100.000) of random (rand()) elements and compare the runtime (difftime()) of your sorting algorithm with the runtime of the list sort algorithm. Note: Make sure you enter EXACTLY the same elements into your own list and the STL container!

4.3 Combining Techniques

In this exercise you will apply all programming techniques that you have learned so far to the “Graphics” project introduced in the lecture.

Your starting point is the reference project that you can download with this task description.

Implement the following features/changes:

- The project has been implemented before you knew about the container types from STL. Replace all “hand made” implementations of sequences with appropriate STL container types.
- Put all classes in namespace “GraSys”.
- Implement missing methods (e. g. “CPlane::boundingBox”). Implement missing copy-constructors, destructors and operator= or operator== where required.
- The graphic system should be usable to represent shapes on a plane. The plane can be a computer screen (where coordinates are denoted using integers) or a mathematical plane (where you use float or double as coordinates). Use templates to make the plane and coordinate type configurable.
- Provide comment for all classes and their methods (in the header file). This includes analyzing existing methods and adding comments where they are missing or insufficient.
- Provide comments that explain the algorithms used (in the cpp file or in template implementations). This includes analyzing existing methods and adding comments where they are missing or insufficient.
- Provide test cases. Some ideas (add your own):
 - Provide a test case for the CCoordinate::move method.
 - Provide test cases for the move methods of the graphic elements.
 - For each type of element, add an element to a new CPlane and check the value of the calculated bounding box.
 - Add an element of each kind with different colors to a CPlane. Check the values of the calculated bounding box applying different filter values.
 - Do the above for both integer and double coordinates.

- Implement all test cases as “automatic test”. This means that you do not decide about success or failure by comparing the output from the screen with what you have in mind as proper result. Rather, follow this pattern:

```
// Overall test result
bool failed = false;

// Invoke method to be tested
CRectangle r("yellow", CCoordinate(1, 1), CCoordinate (2, 2));
r.move(2, 3);
if (r.getCoordinate(0) != CCoordinate(2, 4)
    || r.getCoordinate(1) != CCoordinate(3, 5)) {
    cout << "Testcase move rectangle failed" << endl;
    failed = true;
}

// More testcases follow, each sets failed to true in case of failure.

// ...

// Finally output overall result
if (failed) {
    cout << "Some test cases failed" << endl;
} else {
    cout << "All test cases completed successfully" << endl;
}
```