An Implementation of Banking System using Object-Oriented Programming

Innovative Project Report (LAB)

Course Name: Object Oriented Programming
Course Code: CO-203

Submitted By:
Arjun Manoj Kumar (**2K20/C098)**

Under the Supervision of:
Mr. Manoj Sethi

DEPARTMENT OF COMPUTER ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formely Delhi College of Engineering)

Shahbad Daulatpur, Bawana Road, Delhi- 110042

Certificate

This is to certify that the project titled "An implementation of Banking System using Object-Oriented Programming" submitted by Arjun Manoj, 2K20/CO/098 in partial fulfillment of the requirement for the innovative work component is a record of project work completed by students under my supervision. This work has not, to my knowledge, been submitted in part or in full for any other degree or diploma.

Submitted By:	Submitted To:
Arjun Manoj Kumar	Prof. Manoj Sethi
2K20/CO/098	

ACKNOWLEDGEMENT

I would like to thank my mentor, Mr. Manoj Sethi, Faculty of the Department of Computer Science and Engineering at Delhi Technological University, for his expertise, insights, and advice on this research. I'd also like to thank the Department of Computer Science and Engineering at Delhi Technological University for giving me the opportunity to explore and pursue research in the field of computer science.

CONTENTS:

INTRODUCTION

Overview

This report implements a banking system on C++ platform. It’s part of an Object-Oriented Programming project at Delhi Technological University's Computer Science and Engineering Department. This project aims to create a program that makes it simple to perform banking functions like opening an account, viewing account balances, depositing or withdrawing money, and so on. This project is very efficient because it makes use of the key characteristics of Object-Oriented Programming.

Background & Motivation

As I was learning about OOP's key characteristics and features, I began to consider how useful these features would be in creating a program that can assist in real-world scenarios.

We have a difficult time managing our finances as students. This inspired me to create something to help us keep track of our daily withdrawals and deposits. As the project progressed, this idea evolved into the creation of a mini database that can handle a variety of functions provided by banks.

My previous knowledge of the C language was inadequate for the scope of this project, and working with and improving our knowledge of C++ was critical to the project's success. We looked at and implemented features like Objects and Classes, I/O Streams, Operator Overloading, Exception Handling, and the Standard Template Library from a theoretical standpoint.

Objective

The project's core objectives, which have been designated as essential, are as follows:

- Understand object-oriented programming concepts and apply them in a real-world situation

  We needed to improve our knowledge of object-oriented programming in order to implement this program because it has a lot of features.

- Identify, comprehend, and describe a variety of banking-industry-based methods before incorporating them into our program.

  Opening an account, checking account balances, depositing or withdrawing money from an account, viewing all opened accounts, and closing an account are all examples of bank functions.

CONCEPTS USED

Objects & Classes:

A class is a blueprint or prototype that is defined by the user and from which objects are created. It denotes the set of properties or methods that all objects of a given type have in common. An instance of a class is an object. We've used four classes in this program.

- Account: This class contains the functions that perform all of the mathematical calculations and initializations.
- Bank: In this class, we store all of the account information and call the Account functions.

- InsufficientFunds: The sole purpose of this class is to use it to throw an exception.
- NoAccno: The sole purpose of this class is to use it to throw an exception.

Input & Output streams:

C++ IO are based on streams, which are sequence of bytes flowing in and out of the programs (just like water and oil flowing through a pipe). Data bytes flow into the program from an input source (such as a keyboard, file, network, or another program). Data bytes flow from the program to an output sink during output operations (such as console, file, network or another program).

Streams acts as a bridge between programs and the actual IO devices, allowing programmers to archive device-independent IO operations without having to deal with the actual devices.

Formatted and unformatted IO functions are both available in C++. Formatted I/O functions were used in our program. The stream insertion (<<) and stream extraction (>>) operators are overloaded to support formatted IO operations, resulting in a consistent public IO interface.

I used data files to keep track of the account information in this program. This is used because the data is always saved in the file even after the program has finished running.

Operator Overloading:

We can make operators work for user-defined classes in C++. This means that C++ has the ability to give operators a special meaning for a data type, which is referred to as operator overloading.

For example, we can overload an operator ‘+’ in a class like String so that we can concatenate two strings by just using +.
I've used the insertion and extraction operators as both output stream and outfile, infield stream operators in this program. This is done in order to make the code shorter and easier to read.

Exception Handling:

Exceptions provide a systematic, object-oriented approach to handling errors generated by C++ classes. Exceptions are errors that occur at runtime. They are caused by a wide variety of exceptional circumstance, such as running out of memory, not being able to open a file, trying to initialize an object to an impossible value, or using an out-of-bounds index to a vector.
More information than just the error type is contained in C++ exception objects. They also include a message. When one of our systems fails, it generates a series of messages. When we want to report an error to the user, we usually need to write a more descriptive message that explains how the error is related to what the user did. If the error was caused by incorrect input, for example, it is beneficial to inform the user which of the input values was incorrect. Occasionally, the exception message contains information that we want to show to the user.

In this program, I have used it to display messages such as “Insufficient Funds” when the bank balance is less than 500 and “Incorrect account number” when the user enters an account number which is not present in the data file.

The Standard Template Library:

The Standard Template Library (STL) is a collection of C++ template classes for common programming data structures and functions like lists, stacks, and arrays.

It's a container library with algorithms, iterators, and container classes. Objects and data are stored in containers, also known as container classes. There are seven "first-class" container classes in total, as well as three container adaptor classes.

I used an associative container-Maps in this program. Maps keep track of values and assign an index to each element. The account number is determined by the index, and the account constructor is determined by the element.

IMPLENTATION

CODE:

```
#include<iostream>
#include<fstream>
#include<cstdlib>
#include<vector>
#include<map>
using namespace std;
#define MIN_BALANCE 500
class Bank
{
private:
 map<long,Account> accs;
public:
 Bank();
 Account AccountOpen(string fname,string lname,float
balance);
 Account EnquireBalance(long accountNumber);
 Account Deposit(long accountNumber,float amount);
 Account Withdraw(long accountNumber,float amount);
 bool isPresent(long accountNumber);
 void AccountClose(long accountNumber);
 void ViewAllAccounts();
 ~Bank();
};
class Insufficient_Funds{
public:
 string str;
 Insufficient_Funds(string s)
 {
 str = s;
 }
};
class NoAccno{
public:
 string str;
 NoAccno(string s)
 {
 str = s;
 }
};
class Account
{
private:
 long accountNumber;
 string firstName;
 string lastName;
 float balance;
 static long NextAccountNumber;
public:
 Account(){}
 Account(string fname,string lname,float balance);
 long getAccNo(){return accountNumber;}

 string getFirstName(){return firstName;}
 string getLastName(){return lastName;}
 float getBalance(){return balance;}
 void Deposit(float amount);
 void Withdraw(float amount);
 static void setLastAccountNumber(long accountNumber);
 static long getLastAccountNumber();
 friend ofstream & operator<<(ofstream &ofs,Account
&acc);
 friend ifstream & operator>>(ifstream &ifs,Account
&acc);
 friend ostream & operator<<(ostream &os,Account &acc);
};
long Account::NextAccountNumber=0;
int str2int(string s)
{
 if(s=="1")
 return 1;
 else if(s=="2")
 return 2;
 else if(s=="3")
 return 3;
 else if(s=="4")

 return 4;
 else if(s=="5")
 return 5;
 else if(s=="6")
 return 6;
 else if(s=="7")
 return 7;
 else
 return 0;
}
int main()
```

```
{
 Bank b;
 Account acc;
 int x;
 string fname,lname, choice;
 long accountNumber;
 float balance;
 float amount;
 cout<<" Banking System"<<endl;

 cout<<" By Arjun Manoj Kumar [2K20/CO/98]";
 cout<<"\n\tSelect one option below ";
 cout<<"\n\t1. Open an Account";
 cout<<"\n\t2. Balance Enquiry";
 cout<<"\n\t3. Deposit money";
 cout<<"\n\t4. Withdraw money";
 cout<<"\n\t5. Close an Account";
 cout<<"\n\t6. Show All Accounts";
 cout<<"\n\t7. Quit";
 do
 {
 cout<<"\nPlease enter your choice: ";
 cin>>choice;
 cout<<endl;
 switch(str2int(choice))
 {
 case 1:
 cout<<"Enter First Name: ";
 cin>>fname;
 cout<<"Enter Last Name: ";
 cin>>lname;
 cout<<"Enter initial Balance: ";

 cin>>balance;
 acc=b.AccountOpen(fname,lname,balance);
 cout<<endl<<"Congratulations! A new account has been Created"<<endl;
 cout<<acc;
 break;
 case 2:
 cout<<"Enter Account Number to check balance:";
 cin>>accountNumber;
 x = b.isPresent(accountNumber);
 try {
 if(x==0)
 throw NoAccno("Invalid Entry, try again!");
 acc=b.EnquireBalance(accountNumber);
cout<<endl<<"Your Account Details"<<endl;
 cout<<acc;
 } catch (NoAccno n) {
 cout<<n.str;
 }
 break;
 case 3:
 cout<<"Enter Account Number:";
 cin>>accountNumber;

 x = b.isPresent(accountNumber);
 try {
 if(x==0)
 throw NoAccno("Invalid Entry, try again!");
 cout<<"Enter Amount to be deposited:";
 cin>>amount;
acc=b.Deposit(accountNumber, amount);
cout<<endl<<"Amount has been Deposited"<<endl;
 cout<<acc;
 } catch (NoAccno n) {
 cout<<n.str;
 }
 break;

 case 4:
 try {
 cout<<"Enter Account Number:";
 cin>>accountNumber;
x = b.isPresent(accountNumber);
try {
 if(x==0)
 throw NoAccno("Invalid Entry,try again!");
 cout<<"Enter Amount to be withdrawn:";
 cin>>amount;
acc=b.Withdraw(accountNumber,
amount);
 cout<<endl<<"Amount Withdrawn"<<endl;
 cout<<acc;
 }
catch (NoAccno n) {
 cout<<n.str;
 }
 }
 catch (Insufficient_Funds i){
 cout<<i.str<<endl;
 }
 break;
 case 5:
 cout<<"Enter Account Number to be closed:";
 cin>>accountNumber;
 x = b.isPresent(accountNumber);
 try {
 if(x==0)
 throw NoAccno("Invalid Entry, try again!");
 b.AccountClose(accountNumber);
cout<<endl<<"Account is Closed"<<endl;
// cout<<acc;
 } catch (NoAccno n) {
 cout<<n.str;
 }
 break;
 case 6:
```

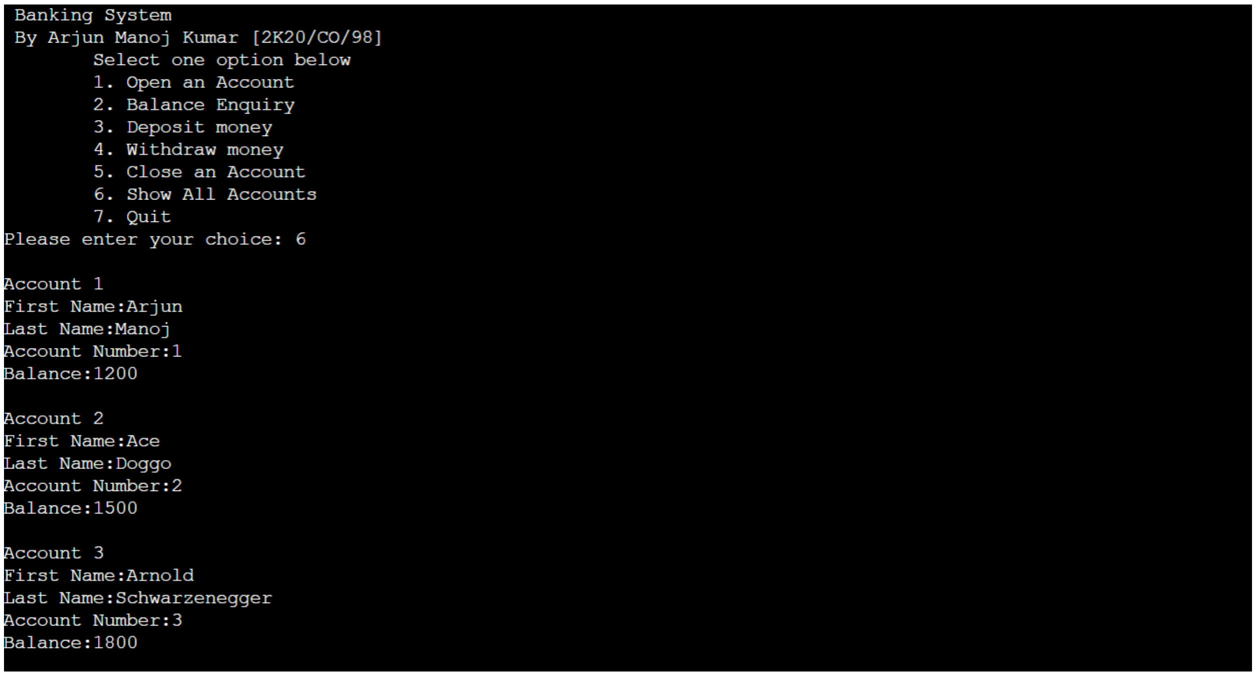
```
 b.ViewAllAccounts();
 break;
 case 7:
 break;
 default:
 cout<<"Enter correct choice";
 break;
 }
 }while(str2int(choice)!=7);
 return 0;
}
Account::Account(string fname,string lname,float balance)
{
 NextAccountNumber++;
 accountNumber=NextAccountNumber;
 firstName=fname;
 lastName=lname;
 this->balance=balance;
}
void Account::Deposit(float amount)
{
 balance+=amount;
}
void Account::Withdraw(float amount)
{
 if(balance-amount<MIN_BALANCE)
 throw Insufficient_Funds("Insufficient Funds");
 balance-=amount;
}
void Account::setLastAccountNumber(long accountNumber)
{
 NextAccountNumber=accountNumber;
}
long Account::getLastAccountNumber()
{
 return NextAccountNumber;
}

ofstream & operator<<(ofstream &ofs,Account &acc)
{
 ofs<<acc.accountNumber<<endl;
 ofs<<acc.firstName<<endl;
 ofs<<acc.lastName<<endl;
 ofs<<acc.balance<<endl;
 return ofs;
}
ifstream & operator>>(ifstream &ifs,Account &acc)
{
 ifs>>acc.accountNumber;
 ifs>>acc.firstName;
 ifs>>acc.lastName;
 ifs>>acc.balance;
 return ifs;
```

```
}
ostream & operator<<(ostream &os,Account &acc)
{
 os<<"First Name:"<<acc.getFirstName()<<endl;
 os<<"Last Name:"<<acc.getLastName()<<endl;
 os<<"Account Number:"<<acc.getAccNo()<<endl;
 os<<"Balance:"<<acc.getBalance()<<endl;
 return os;
}
Bank::Bank()
{
 Account account;
 ifstream infile;
 infile.open("Bank.data");
 if(!infile)
 {
 cout<<"Error in Opening! File Not Found!!"<<endl;
 return;
 }
 while(!infile.eof())
 {
 infile>>account;


accs.insert(pair<long,Account>(account.getAccNo(),account))
;
 }
 Account::setLastAccountNumber(account.getAccNo());
 infile.close();
}
Account Bank::AccountOpen(string fname,string lname,float
balance)
{
 ofstream outfile;
 Account account(fname,lname,balance);

accs.insert(pair<long,Account>(account.getAccNo(),account))
;
 outfile.open("Bank.data", ios::trunc);
 map<long,Account>::iterator itr;
 for(itr=accs.begin();itr!=accs.end();itr++)
 {
 outfile<<itr->second;
 }
 outfile.close();
 return account;
}
Account Bank::EnquireBalance(long accountNumber)
{
 map<long,Account>::iterator
itr=accs.find(accountNumber);
 return itr->second;
}
Account Bank::Deposit(long accountNumber,float amount)
```

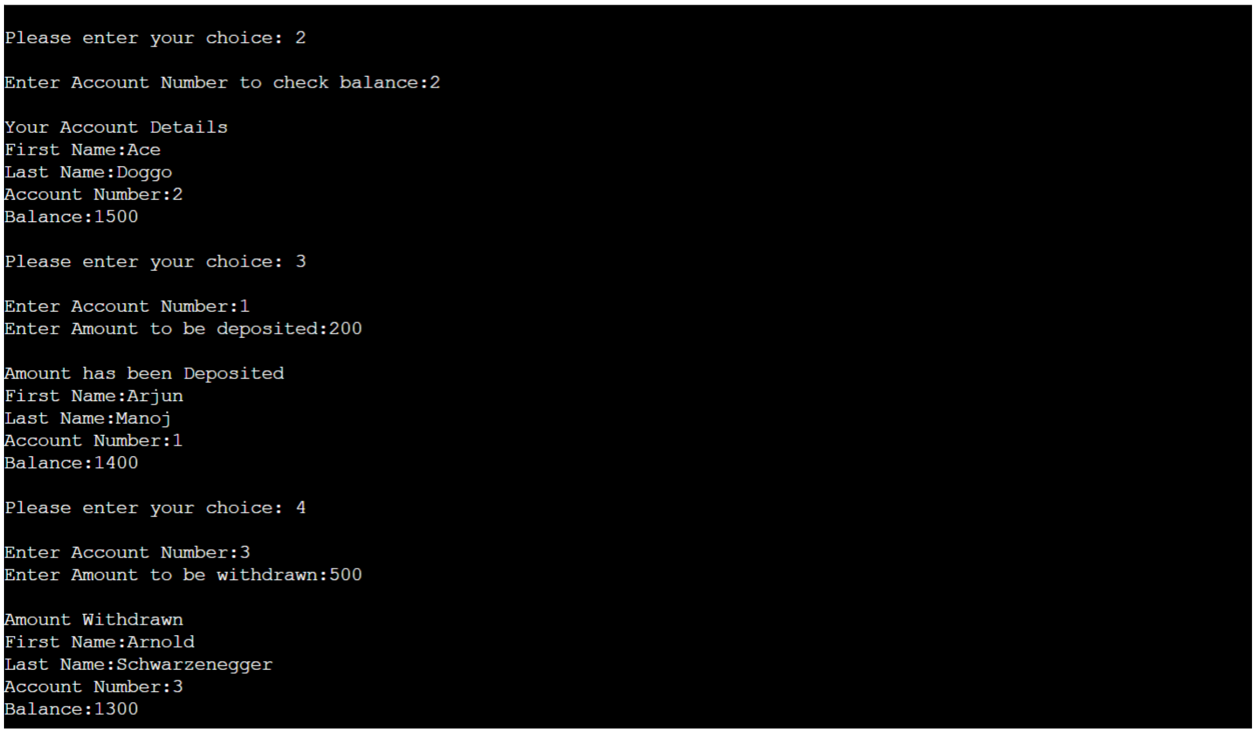
```
{
 map<long,Account>::iterator
itr=accs.find(accountNumber);
 itr->second.Deposit(amount);
 return itr->second;
}
Account Bank::Withdraw(long accountNumber,float amount)

{
 map<long,Account>::iterator
itr=accs.find(accountNumber);
 itr->second.Withdraw(amount);
 return itr->second;
}
void Bank::AccountClose(long accountNumber)
{
 map<long,Account>::iterator
itr=accs.find(accountNumber);
 cout<<"Account Deleted"<<endl<<itr->second;
 accs.erase(accountNumber);
}
void Bank::ViewAllAccounts()
{
 map<long,Account>::iterator itr;
// if(accounts.begin()==accounts.end())
// {
// cout<<"No Accounts open."<<endl;
// return;
// }
 for(itr=accs.begin();itr!=accs.end();itr++)
 {
 cout<<"Account "<<itr->first<<endl<<itr->second<<endl;
 }
}
bool Bank::isPresent(long accountNumber)
{
 map<long,Account>::iterator itr;
 int key = 0;
 for(itr=accs.begin();itr!=accs.end();itr++)
 {
 if(itr->first==accountNumber)
 key++;
 }
 return key==1? 1:0;
}
Bank::~Bank()
{
 ofstream outfile;
 outfile.open("Bank.data", ios::trunc);
 map<long,Account>::iterator itr;
 for(itr=accs.begin();itr!=accs.end();itr++)
 {
 outfile<<itr->second;}
 outfile.close();}
```

OUTPUT:

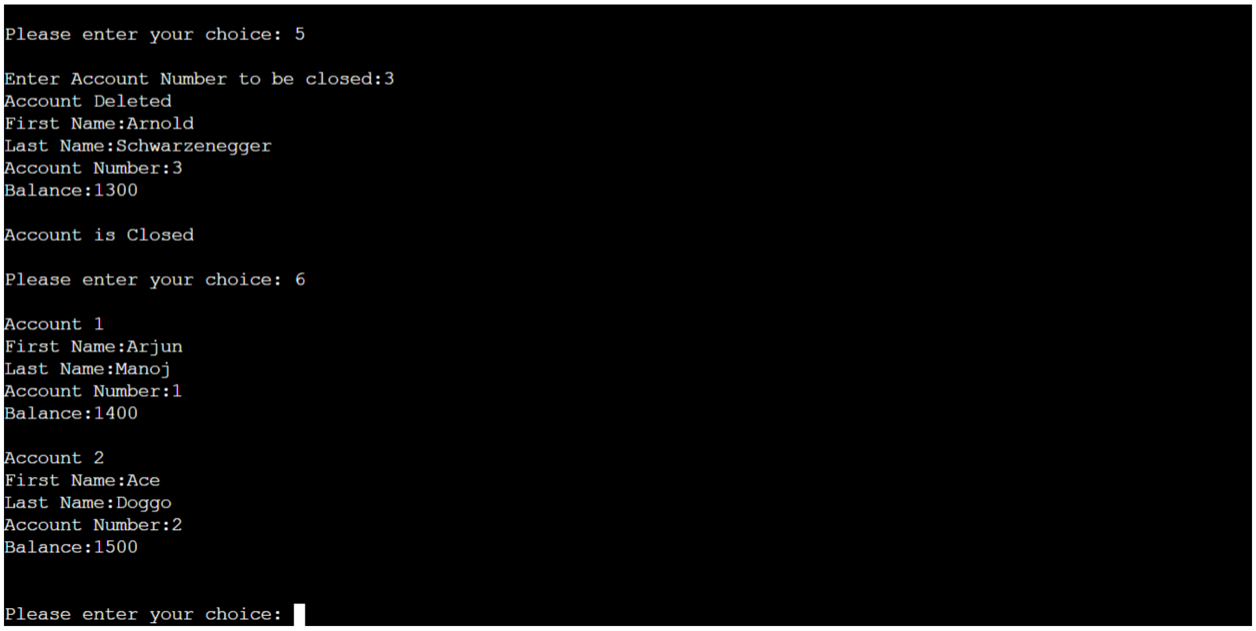
To begin, I added three accounts to the data file. The following outputs contain all of the operations.



```
 Banking System
 By Arjun Manoj Kumar [2K20/CO/98]
        Select one option below
        1. Open an Account
        2. Balance Enquiry
        3. Deposit money
        4. Withdraw money
        5. Close an Account
        6. Show All Accounts
        7. Quit
Please enter your choice: 6

Account 1
First Name:Arjun
Last Name:Manoj
Account Number:1
Balance:1200

Account 2
First Name:Ace
Last Name:Doggo
Account Number:2
Balance:1500

Account 3
First Name:Arnold
Last Name:Schwarzenegger
Account Number:3
Balance:1800
```



```
Please enter your choice: 2

Enter Account Number to check balance:2

Your Account Details
First Name:Ace
Last Name:Doggo
Account Number:2
Balance:1500

Please enter your choice: 3

Enter Account Number:1
Enter Amount to be deposited:200

Amount has been Deposited
First Name:Arjun
Last Name:Manoj
Account Number:1
Balance:1400

Please enter your choice: 4

Enter Account Number:3
Enter Amount to be withdrawn:500

Amount Withdrawn
First Name:Arnold
Last Name:Schwarzenegger
Account Number:3
Balance:1300
```



```
Please enter your choice: 5

Enter Account Number to be closed:3
Account Deleted
First Name:Arnold
Last Name:Schwarzenegger
Account Number:3
Balance:1300

Account is Closed

Please enter your choice: 6

Account 1
First Name:Arjun
Last Name:Manoj
Account Number:1
Balance:1400

Account 2
First Name:Ace
Last Name:Doggo
Account Number:2
Balance:1500


Please enter your choice:
```

Conclusion

This project is based on a Banking system application which deals with tasks such as opening an account, withdrawing and depositing money ,etc.
I have been able to implement various coding methods and techniques that I have learnt through my OOPS course in DTU in order to make this project a successful one.

Reference

- https://www.geeksforgeeks.org/c-classes-and-objects/
- https://www.tutorialspoint.com/cplusplus/cpp_basic_input_output.htm
- https://en.cppreference.com/w/cpp/language/operators
- https://www.geeksforgeeks.org/operator-overloading-c/
- https://www.mygreatlearning.com/blog/standard-template-library-in-c/
- https://docs.microsoft.com/en-us/cpp/cpp/errors-and-exception-handling-modern-cpp?view=msvc-160
- https://www.geeksforgeeks.org/exception-handling-c/#:~:text=Exception%20handling%20in%20C%2B%2B%20consists,us%20create%20a%20custom%20error.