PART ONE: CONSTRUCTOR, COPY CONSTRUCTOR, DESTRUCTOR, ASSIGNMENT OPERATOR

1.1. Constructor

Constructors will be called whenever an object of the class is created. A class can have more than one constructor.

```
//Student.h – start
#ifndef STUDENT_H
#define STUDENT_H

#include <iostream>
using namespace std;

class Student {

private:
char *stdName;
int stdID;
double CGPA;

public:
Student(){
	stdName = new char[100];
	strcpy(stdName, "Not provided");
	stdID = 0;
	CGPA = 0;
	cout << "Student() has been called!" << endl;
}

Student(char *sName, int sID, double sCGPA){
	stdName = new char[100];
	strcpy(stdName, sName);
	stdID = sID;
	CGPA = sCGPA;
	cout << "Student(char *sName, int sID, double sCGPA) has been called!" << endl;
}
```

1.2. Copy Constructor

Copy constructor is used to construct an object from an existing object of the same class.

```
Student(const Student& std){
	stdName = new char[100];
	strcpy(stdName, std.stdName);
	stdID = std.stdID;
	CGPA = std.CGPA;
	cout << "Student(const Student& std) has been called!" << endl;}
```

1.3. Destructor

Destructors will be called whenever an object of the class is destroyed.

```
~Student(){
	delete[] stdName;
	cout << "~Student() has been called!" << endl;
}
//Student.h - end
```

1.4. Assignment Operator

Assignment operator is used to assign an existing object to another existing object of the same class.

```
Student& operator=(const Student& std){
       if (this == &std)
       {
            cout << "Student& operator=(const Student& std) has been called!" << endl;
            return *this;
       }

       delete[] stdName;

       stdName = new char[100];
       strcpy(stdName, std.stdName);
       stdID = std.stdID;
       CGPA = std.CGPA;

       cout << "Student& operator=(const Student& std) has been called!" << endl;
       return *this;
}

};
#endif
```

PART TWO: INHERITANCE

You can find an example class below which is derived from the Student class.

```
#ifndef UNDERGRADUATESTUDENT_H
#define UNDERGRADUATESTUDENT_H

#include <iostream>
using namespace std;

#include "Student.h"

class UndergraduateStudent : public Student {

private:
       int year;

public:
       UndergraduateStudent(){
              year = 0;
              cout << "UndergraduateStudent() has been called!" << endl;
       }

       UndergraduateStudent(char *sName, int sID, double sCGPA, int sYear) :
Student(sName, sID, sCGPA){
              year = sYear;
              cout << "UndergraduateStudent(char *sName, int sID, double sCGPA, int
sYear) has been called!" << endl;
       }
};
#endif
```

Exercises – Part 1

1. Determine the output of the following C++ program.

```
#include <iostream>
using namespace std;
#include "Student.h"
int main(){
	Student *mystudent1 = new Student();
	Student *mystudent2 = new Student("Sukru Eraslan", 100, 4.00);
	Student *mystudent3 = new Student(*mystudent1);
	*mystudent3 = *mystudent2;
	delete mystudent1;
	delete mystudent2;
	delete mystudent3;
	return 0;
}
```

2. Design, implement and test a class that represents an address that consists of a house number, street name and city. By default, the street name and city should be "not provided" and the house number should be equal to 0. This class should also provide a constructor which enables all the values to be set. You need to override the copy constructor, assignment operator and destructor. <u>Note: You need to define the street name and city as a character pointer.</u>

Exercises – Part 2

1. Determine the output of the following C++ program.

```
#include <iostream>
using namespace std;
#include "UndergraduateStudent.h"
int main(){
	UndergraduateStudent *mystudent1 = new UndergraduateStudent();
	UndergraduateStudent *mystudent2 = new UndergraduateStudent("Sukru Eraslan", 100,
4.00, 4);
	UndergraduateStudent *mystudent3 = new UndergraduateStudent(*mystudent1);
	*mystudent3 = *mystudent2;
	delete mystudent1;
	delete mystudent2;
	delete mystudent3;
	return 0;
}
```

Additional Exercises

2. Implement an Account class, which contains two private data elements an integer accountNumber and a floating-point value accountBalance, and the following default constructor and member functions:

   a. A default constructor that set initial values for accountNumber and accountBalance to 0.

   b. Getter and setter functions for each attribute (accountNumber and accountBalance).

   c. The inputTransaction function, which takes a character value for transactionType ('D' for deposit and 'W' for withdrawal), and a floating-point value for transactionAmount, which updates accountBalance. This function should give an error message if there is no sufficient amount for withdrawal.

   d. The calculateFutureBalance function, which takes annual interest rate and number of years, and then calculates the future balance for the given period. For example, if the current amount is 1000, we pass 0.05 as the interest and 1 as the number of years. The amount will be 1050 at the end of the year.

   e. The mortgageYear function, which takes total amount of a mortgage and then calculates the number of years required to pay the mortgage only by using the interest of the current balance. Note that for the mortgage, apart from the lump sum we have to pay, interest for the borrowed money.

   Let say we have a mortgage for 100.000 with an annual interest rate 8.90 %, and we make annual payments. Let us assume we have 300.000 in our account with 5% interest rate. At the end of first year mortgage will become 108.900 and from my money I will have 15000 interest rate. Then I can reduce the mortgage to 93900.

3. You need to write a complete C++ program for a bank which uses the class implemented in Part (2). This program should provide the following menu (shown in sample run) for a particular bank account. **Hint**: You first need to create an Account object.

4. Update the class implementation in Part (2) to add a private char* accountName variable. Write a copy constructor, assignment operator overloading and a destructor for this version. Additionally, whenever the destructor is called, your program should print “Account is closed!”. Make sure to handle heap variables.

Sample Run:

```
1. Change the account balance
2. Get the current balance
3. Deposit
4. Withdrawal
5. Plan your future balance
6. Mortgage plan
7. Exit
Enter your selection: 1
Enter a new account balance: 300000
New account balance is 300000 TL

1. Change the account balance
2. Get the current balance
3. Deposit
4. Withdrawal
5. Plan your future balance
6. Mortgage plan
7. Exit
Enter your selection: 2
Account balance is 300000 TL

1. Change the account balance
2. Get the current balance
3. Deposit
4. Withdrawal
5. Plan your future balance
6. Mortgage plan
7. Exit
Enter your selection: 3
Enter an amount for deposit: 20000
New account balance is 320000 TL

1. Change the account balance
2. Get the current balance
3. Deposit
4. Withdrawal
5. Plan your future balance
6. Mortgage plan
7. Exit
Enter your selection: 4
Enter an amount for withdrawal:
400000
No sufficient amount in the bank
account
New account balance is 320000 TL

1. Change the account balance
2. Get the current balance
3. Deposit
4. Withdrawal
5. Plan your future balance
6. Mortgage plan
7. Exit
Enter your selection: 4
Enter an amount for withdrawal: 20000
New account balance is 300000 TL

1. Change the account balance
2. Get the current balance
3. Deposit
4. Withdrawal
5. Plan your future balance
6. Mortgage plan
7. Exit
Enter your selection: 5
Enter an annual interest rate: 0.05
Enter a number of years: 3
You balance will be 347287.5 TL after
3 years with the interest rate 0.05

1. Change the account balance
2. Get the current balance
3. Deposit
4. Withdrawal
5. Plan your future balance
6. Mortgage plan
7. Exit
Enter your selection: 6
Enter a total amount of a mortgage:
100000
Enter an annual mortgage interest
rate: 0.089
Enter an annual interest rate: 0.05
You require 11 years to pay

1. Change the account balance
2. Get the current balance
3. Deposit
4. Withdrawal
5. Plan your future balance
6. Mortgage plan
7. Exit
Enter your selection: 7
Goodbye!
```