# Programming II Exam project

This is the final examination project for the subject Programming II in C++. The examination consists of five programming tasks which the student will have to solve using C++. Complete all the questions and insert all your files into a single word document. You can simply copy your code and paste it into a word document, so that it looks like this:

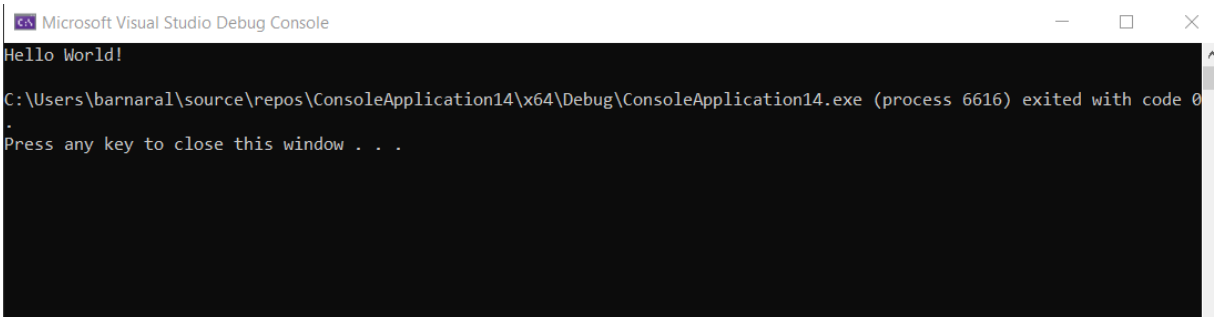
1. This is my main.cpp file:

```
#include "Rectangle.h"
using namespace std;

int main()
{
	Rectangle rect1;
	Rectangle rect2(10, 20);
	Rectangle rect3(100, 50);
	Rectangle sumRect;

	sumRect = rect1 + rect2;
	cout << sumRect.getLength() << " " << sumRect.getWidth()<<endl;
	cout << sumRect.area() << " " << sumRect.perimeter() << endl;
	cout << "Rect1 == Rect3? " << boolalpha<<(rect1 == rect3) << endl;
	cout << "Rect1 == Rect2? " << boolalpha << (rect1 == rect2) << endl;
	return 0;
}
```

This is just an example but I expect that it should look like this, with the different colours. The console output should look like this (an example):




Save your document as a single PDF and name it YourName_StudentNumber_Programming2.pdf. Replace YourName with your own name and StudentNumber with your own student number. To be extra safe, also include your name and student number inside the document.

If you have any questions, please let me know. You will have until Tuesday the 3rd of May at 23:00 pm (China time) to email me your single pdf document to my email address: **Alfred.barnard@srh.de**

Good Luck!

## Question 1 -Basic UML

Consider the following UML (unified modelling language) class descriptive diagram:

| ProgClass |
| --- |
| - name : string<br>- studNum : int<br>- marks : double |
| + ProgClass(name : string, studNum : int, marks: double)<br>+ getName() : string<br>+setName(name:string): void<br>+getStudNum(): int<br>+setStudNum(studNum : int) :void<br>+getMarks(): double<br>+setMarks(marks : double) :void<br>+studPassed() : bool |

Create a **ProgClass** header file with a separate implementation file. Also create a main file where you create at least one object of the class **ProgClass**. The main file should be used to demonstrate the methods in your implementation file. The last implementation method called **studPassed** should return a true or false value if the **marks** is greater than 60%. You should print out true or false value inside of the main program when this method is called. The value for **marks** can be anything between 0 and 100 %. I will need the following:

1. The header file **ProgClass.h**
2. The implementation file **ProgClass.cpp.**
3. The main file where the objects are created and filled with values.
4. Screenshot of the console showing your output.

## Question 2 – Derived UML

Consider the following UML (unified modelling language) class descriptive diagram:

| Vehicle |
| --- |
| - make : string<br>- price : double |
| + Vehicle(name : string, price: double)<br>+ getMake() : string<br>+setMake(make:string): void<br>+getPrice(): double<br>+setPrice(price : double) : void |

And:

| Bus |
| --- |
| - passengers : int |
| + Bus(make :string, double: price ; passengers : int)<br>+ setPassengers(int passengers) : void<br>+ getPassengers() : int<br>+ horn() : void |

The Bus class is derived from the Vehicle class via a public inheritance.

Create a **Vehicle** and separate **Bus** header files with a separate implementation file. The class **Vehicle** should contain only pure virtual functions. Also create a main file where you create at least one object of the class **Bus**. The main file should be used to demonstrate the methods in your implementation file. The last implementation method called **horn,** print out the sound that a bus actually makes. I will need the following:

1. The header file **Vehicle.h.**
2. The header file **Bus.h.**
3. The implementation file **Bus.cpp.**
4. The main file where the objects are created and filled with values.
5. Screenshot of the console showing your output.

## Question 3 – Class Template

Create a class using templates that stores four items. These items can be either from the data type : Integer, String or Boolean. The template class called **Store4** should have at least one user defined constructor to set all the data items. There should also be four methods to get the individual data items, for example a method to get the first item, method to get the second item, method to get the third item and a method to get the fourth item. Do not use overloaded functions, you should make use of the template functionality. Finally there should also be a final method where all the data members are printed out in one line with at least one space between them.

The main file should create three objects using the **Store4** class. The first object should accept integers, the second object should accept strings and the third object should accept boolean values. You can decide the values entered into the objects. Finally inside main you should call the print method on all three objects to display their data. I will need the following:

1. The header file **Store4.h**
2. The main file where the objects are created and filled with values.
3. Screenshot of the console showing your output.

## Question 4 – Runtime Error

Create a separate compilation class implementation called **NumCheck.h** along with its implementation **NumCheck.cpp**. This class should have a user defined constructor that accepts one integer. It should also have two methods, **getNum** and **setNum. setNum** should have a conditional statement that checks if the incoming integer is 0. If the integer is 0, throw a **runtime_error** which gives the user a message that a zero number should not be entered. Otherwise save the integer to the private data member **intNum.** In the main file, create a try and catch block. In the try block create a couple of objects from the class **NumCheck,** one of the objects should try and pass along a 0 to the created object. The catch block, should catch the exception and print out the error. I will need the following:

1. The header file **NumCheck.h**
2. The implementation file **NumCheck.cpp.**
3. The main file where the objects are created and filled with values.
4. Screenshot of the console showing your output.

## Question 5 – RPG character save and load

Create a separate compilation implemented class called PlayerCharacteristics according to the following UML class descriptive diagram:

| PlayerCharacter |
| --- |
| - health : int<br>- attack : int<br>- defence : int<br>- name : string |
| + PlayerCharacter()<br>+ PlayerCharacter(name : string, health : int, attack : int, defence : int)<br>+ ~PlayerCharacter()<br>+ getName() : string<br>+setName(name:string): void<br>+getAttack(): int<br>+setAttack(attack : int) :void<br>+getDefence(): int<br>+setDefence(defence : int) :void<br>+getHealth(): int<br>+setHealth(health : int) :void<br>+printCharacter() : void |

You should create a **PlayerCharacteristics.h** header file and **PlayerCharacteristics.cpp** implementation file**.** In the main file you should create an object from the class **PlayerCharacteristics.** After the class has been created, the contents of the objects should be saved to a text file called **saveFile.txt.** The content of the file should look like this, as an example:

Name:

Larry

Defence:

80

Attack:

20

Health:

200

After the file has been created, you should then read the contents from the text file **saveFile.txt** and then use it to create a new object from the class **PlayerCharacteristics** moving all of the information to this newly created object called **newCharacter.** The method **printCharacter** should then be called to print out all the data contained in the object **newCharacter.** I will need the following:

1. The header file **PlayerCharacteristics.h.**
2. The implementation file **PlayerCharacteristics.cpp.**
3. The main file where the objects are created and filled with values.
4. Screenshot of the console showing your output.

# END OF EXAM