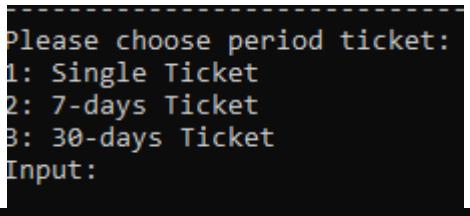
Introduction

In this course of C++ / C I was assigned to a project of different topics. My group and I decided on choosing the topic of Train Booking Tickets. All of us knew how a Train Booking system works. The motivation behind our choice was that we found the topic very interesting and easy to visualize and execute with C++. In order for the program to work, you need to compile all of the .cpp , .h and main.cpp to get the executable file

Problem Statement

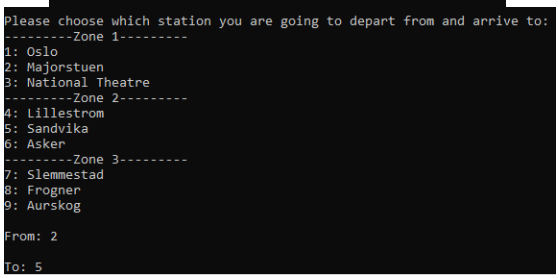
The project's problem statement is the following: Create a train booking ticket program that takes an input user and by that input determine what train ticket the user will get. According to the user's choice, the price will vary. When the user executes the program of the train booking ticket system, he/she is tasked to choose options from 3 important menus that will determine the ticket.

- *The first menu is the following:*



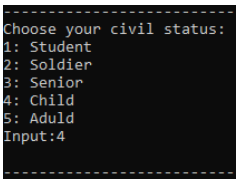
This way the program will determine what kind of type of a ticket the user would choose and then adjust the price of the ticket.

- *The second menu is the following:*



After the first menu, the user is then tasked to choose from where he/she wants to depart, to where he/she wants to arrive. Every single station listed on the menu belongs to a zone. The price of the ticket will vary when the user wants to travel from one zone to another.
For example, if the user decides to travel from Oslo that belongs to Zone 1, to Slemmestad which belongs to Zone 3, then there will be additional expenses to the base price of that ticket. Each zone has its own base price. Therefore, traveling from station to station within the same zones, will the price remain the same, in other words, equivalent to that specific zone's base price.

- *The third menu is the following*:



After the user has selected where he/she wants to travel to, then the programs askes the civil status of the user. This is an important query to the program because each civil status listed on the menu has special price adjustments. For example, if the user chooses nr 4 (child) then the program will set the price to 0 no matter where the destination is. If the user is a student, then there will be a percentage that will be deducted from the overall pricing of the ticket.

Methods

The program is divided into 3 important classes, ticket-class, station-class and the main-class.

- Ticket-Class Methods:

We used a header file *ticket.h* to access the methods and variables to other classes within the project.
Inside the ticket-class we have a constructor, setters and getters-methods, an method called *receipt ()*, an method called *ticket_Price()* and finally an method called *printReceipt()*. Every method in this class serves a unique role to the program.

1. Constructor:

```
Ticket::Ticket(string station_from, string station_to, int zone_From, int zone_To,int civilStatus,int type)
{
    this->station_from = station_from;
    this->station_to = station_to;
    this->zone_From = zone_From;
    this->zone_To = zone_To;
    this->civilStatus = civilStatus;
    this->type = type;

}
```

Ticket() constructor takes parameters of type string and int. *Ticket()* will create an object that will contain the station the user will be taking the train from, the station the user will arrive, the zone of the station the user will depart, the zone that the user will arrive to, the civil status as an integer (will later be used in a switch loop to

determine the civil status of the user), and the type of ticket as an integer (will later be used in a switch loop to determine the type of the ticket) . With *this->* I assign the variables to the ticket class with the values of the constructor's parameters.

2. Setters and getters:

```
// setters
        void Ticket::setStationFrom(string from)
        {
            this->station_from = from;
        }

        void Ticket::setStationTo(string to)
        {
            this->station_to = to;
        }

        void  Ticket::setZoneFrom(int from)
        {
            this->zone_From = from;
        }

        void Ticket::setZoneTo(int to)
        {
            this->zone_To = to;
        }

        void Ticket::setCivilStatus(int civilStatus)
        {
            this->civilStatus = civilStatus;
        }

        void Ticket::setPeriodType(int periodType)
        {
            this->type = periodType;
        }
```

```
//getters
string Ticket::getStationFrom()
{
        return this->station_from;
    }

string Ticket::getStationTo()
{
    return this->station_to;
}

int Ticket::getZoneFrom()
{
    return this->zone_From;
}

int Ticket::getZoneTo()
{
    return this->zone_To;
}

int Ticket::getCivilStatus()
{
    return this->civilStatus;
}

int Ticket::getPeriodType()
{
    return this->type;
}
```

With *setters and getters*, I will later change the values of any variable in the main-class. Set methods are used here to take a value and store it within one of the variables from the class *Ticket*. *Get* methods will return the value of the variable from the class *Ticket.*

3. *Receipt():*

The *receipt()* method is here used to display the ticket information to the user. The method contains information such as: The stations that the user will depart and arrive to, the type of ticket, the civil status. Station from ,station to, zone from , zone to will be later added to the output in the main-class by the help of station-class. In order to determine what type of ticket the user chose from the program, we created a *switch-loop*. This *switch-loop* takes in the parameters of the variable *int type.* We have created several cases that correspond to different types of tickets such as *1 : Single ticket , 2: 7-days ticket and 3: 30-days ticket.* If the variable *type* is equal to any of those *cases*, then the program will print out that specific type. The same will occur for the *int civilStatus switch case*.

```
switch(type)
{
    case 1: cout<<"| Type: "<<"-----Single Ticket-----"<<" |"<<endl;
            break;
    case 2: cout<<"| Type: "<<"-----7-days Ticket-----"<<" |"<<endl;
            break;
    case 3: cout<<"| Type: "<<"-----30-days Ticket-----"<<" |"<<endl;
            break;

    default: break;
}
```

4. *ticket_Price():*

The *ticket_Price()* method's purpose is to calculate the final price of the ticket according to the user's input. Civil status, type of ticket, zone will play a big role to the price's adjustment. The method begins by initializing *int price* with a base price of *10 kr*. The base price will increase at the end of the method and will be the final price for the ticket. With switch-loops we are able to figure out the final price.

```
//calculates tickets Price
    float Ticket::ticket_Price()
    {
  //base value for price
        price = 10;

        // price adjustment according to zones
        if(std::abs(zone_From-zone_To) == 0)
        {
            price += 50;
        }
        else if(std::abs(zone_From-zone_To) == 1)
        {
            price += 100;
        }
        else if(std::abs(zone_From-zone_To) == 2)
        {
            price += 150;
        }

  //price adjustment according to civilStatus
        switch(civilStatus)
        {
            case 1 :
            price = price * 0.6;
            break;

            case 2 : price = price * 0.7;
                break;

            case 3 : price = price * 0.8;
                break;
            case 4 : price = 0;
                break;

            case 5 : break;

            default : break;
        }

    //price adjustment according to ticket type
        switch(type)
        {
            case 1: price += 50;
                break;
            case 2: price = price * 0;
                break;
            case 3: price = price * 10;
                break;

            default: break;
        }
        return price;
    }
```

- First, we use *if- else if* to add *price* with the price for each *zone*. Here we made it so that every zone has its own base price. Within the parameters of each statement we subtract the variables *zone_From – zone_To*. With *abs()* we are able to determine the absolute value of the result. Then we say that if the absolute value for *zone_From – zone_To* is equal to 0 then the base price will increase by 50. This means that the user has chosen to travel from stations that belong to the same zone. However, if the result is equal to 1 then it means that the user has travelled one zone more.

-For example: *Oslo -> Lillestrøm* . *Oslo* belongs to *Zone 1* while *Lillestrøm* belongs to *Zone 2*. The user has to travel one zone more. If the result is equal to 2, then the user will travel to a station that is 2 zones away from the zone he/she will depart from. Every civil status and type of ticket has a unique price. For *civil status* we decided to have the *price* be deducted by percentages based on what the civil status corresponds to.

As for the *type* of the ticket, we created again a *switch*-loop that each of its cases will increase the price according to the parameter's value.

5. *printReceipt():*

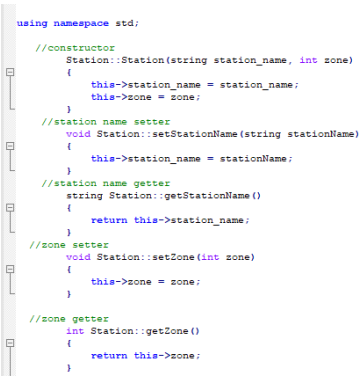
This method will save the finalized ticket into a *.txt* file. This will serve the purpose of being the ticket for the user to use when entering the train. In order to create the *.txt* file and save the information in it we did the following:

a. *#include<fstream>* we included fstream.
b. With *ofstream myfile* we are able now to actually file manage.
c. With *myfile.open()* we create the *.txt* file. Inside the *open()* we write the name that the file is going to have, in this case we named it *ticket.txt*
d. With the help of a *switch*-case we are able to determine the type of ticket. If type is equal to 1 then with *myfile<<"Type: -----Single Ticket----- \n";* we write to *ticket.txt* file that type.
e. We do the same for the civil status.
f. We write in *ticket.txt* the stations that the user chose and finally the price by calling the method *ticket_Price()* which will return the *price*'s value.
g. At the end we make *myfile.close();* so that we prevent any memory leaks of the program.

Station-Class Methods:

We used a header file *station.h* to access the methods and variables to other classes within the project.

Inside the station-class we have a constructor, setters and getters-methods. The purpose of this class is to capture the stations names and zones from main and transfer them to ticket-class to create the actual ticket.



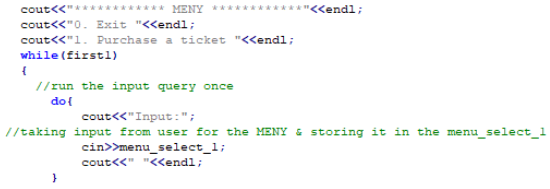
- Main- function Methods:

In main we have one big method. Its purpose is to use ticket, station objects and display the menu of the program that has inputs and at the end to display the ticket.

1. *Menu()*- method:

In this method we literally run the program. We have *cout* and *cin* to display and capture the user's input. In the beginning of the method we have created the variables. *menu_select_N* ,(where N = numbers) , are used in every single menu option to store the decision that the user has made.
For example, in one of the menus from the program we use *menu_select_1* to store the user's input and later use in a *switch*-loop. With *cin>>* we were able to store that input.

In the first menu the user is asked to either exit or buy a ticket. The input is stored in *menu_select_1* and then used in a switch-loop. With a if-statement if the input is equal to 0 then with *exit(1);* the program shuts down. With 1 the program continues to the next menu. In order to validate the user's input, we created the following:



In this piece of code, we have 2 while loops, in which one is inside the other. The outermost while-loop all it does is to run the innermost while-loop until the parameter's value turns to false. The inner while loop's purpose is to run an if-statement that checks the users input if it is equal and not more or less with the options that are available from the menu. We did this by saying *menu_select_5 <= 3 &&menu_select_5 >0* then the user's input is correct. When *menu_select_5 > 3 || menu_select_5 <=0* then the input does not match with the options available and in return it will ask the user to type in again until the input has been corrected. When the input is correct then the *bool first* variable is set to false which will stop the loop and continue down with the next menu. The same validation technique is used in all the menus therefore I am not going to repeat the implementation further on.

```
while(firstl)
{
    do{
        cout<<"Input:";

        cin>>menu_select_5;
        cout<<" "<<endl;
    }
    while(first);
    {
         if(menu_select_5 <= 3 &&menu_select_5 >0)
        {
            first = false;
            break;
        }else if(menu_select_5 > 3 || menu_select_5 <=0)
        {
            cout<<"Please type 1 to 3: "<<endl;
            firstl = true;
        }

    }    //END User Input Validity Check
}
```

Every option the user chooses from the menu is stored via the *menu_select_N* variable and some *switch*-cases. In this example *menu_select_2* stores the value for the station that the user will depart from. If the variable's value is equal to one of those cases, then we have hard-coded the names and zones for each station. So, if *menu_select_2 = 1* then with the *string variable station_name_from* we would save the name "*Oslo*". The same will occur for the *zone*. Those variables are essential for the ticket class in order to determine the ticket. At the near end of the *menu()* method we instantiate 3 main objects. 1 and 2 objects are from the class Station and the other from the class Ticket. In the *station_from* object we store the variables *station_name_from* and *zone_from*. In *station_to* we do the exact same thing but with the variables *station_name_to* and *zone_to* . Finally, in the *travel* object from the *ticket* class we store all the values needed to create the price and the ticket itself. In the parameters of *travel*, by calling the get-methods from station class we are able to store the values in the object.

```
// assign values to later store in station object.
switch(menu_select_2)
{
    case 1 : station_name_from = "Oslo";
             zone_from = 1;
             break;

    case 2 : station_name_from = "Majorstuen";
             zone_from = 1;
             break;

    case 3 : station_name_from = "National Theatre";
             zone_from = 1;
             break;

    case 4 : station_name_from = "Lillestrom";
             zone_from = 2;
             break;

    case 5 : station_name_from = "Sandvika";
             zone_from = 2;
             break;
    case 6 : station_name_from = "Asker";
             zone_from = 2;
             break;

    case 7 : station_name_from = "Slemmestad";
             zone_from = 3;
             break;

    case 8 : station_name_from = "Frogner";
             zone_from = 3;
             break;

    case 9 : station_name_from = "Aurskog";
             zone_from = 3;
             break;
```

With *travel->printReceipt();* we run the method that will create the .txt file that will contain the price and the journey's information. With *while(menu_select_1 != 0);* we are able to run the program indefinitely unless *menu_select_1* is equal to 1 where in this case 1 is for choosing the option to buy another ticket in the menu.

Results

The result of the program at the end is the following:

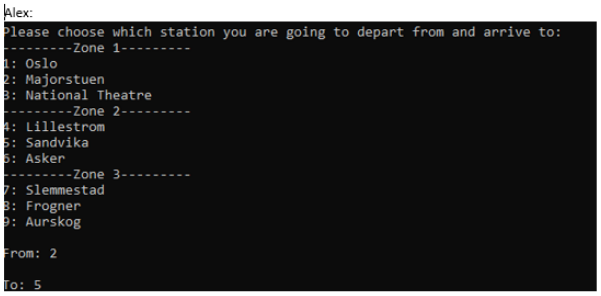
```
-------------------------------------------
| ----------Receipt---------- |
| Type: -----7-days Ticket----- |
| Civil Status: Student
 From: National Theatre
 To: Sandvika
 Price: 440 kr
-------------------------------------------
Press any key to continue . . .
--------------------------------------------------------------------------
                -----TRAIN BOOKING TICKET SERVICE-----
--------------------------------------------------------------------------

************ MENY ************
0. Exit
1. Purchase a ticket
Input:0
```

```
**********************************
--------- Ticket ----------
Type: -----7-days Ticket-----
Civil Status: Student
From: Sandvika
To: National Theatre
Price: 440 kr
**********************************
```

The program at the end will display the receipt of the purchase of the ticket. It displays the type of the ticket the user chose, the user's civil status, the stations that the user wants to travel and finally the overall price of the ticket. When the user presses any key, he will be redirected to the first menu. The user can either press 0 to exit or 1 to make another purchase. In the same file where the source code is, an .txt file will be generated that will contain the ticket itself. It is similar to the receipt, only this time the user can print it out and carry it to the train.

Individual Contribution

Alexandros Messaritakis Chousein Aga:



- The coding of having the user choosing the stations of departure and arrival. That would consist of input, the logic behind choosing a station.
- Created a way to make a .txt file, the receipt the user receives at the end of the program.
- In the ticket class, I created the pricing system for every station and zone.
- Everyone in the group worked with everything. In every meeting that we had we were to show what we have done and then agree on to what to keep and what to throw. Every class was coded by everyone, but at the end we kept the best version that was mixed of all our inputs.

Jan-Fredrik Eri Kopperud:

```
// price adjustment according to zones
if(std::abs(zone_From-zone_To) == 0){
    price += 50;
}
else if(std::abs(zone_From-zone_To) == 1){
    price += 100;
}
else if(std::abs(zone_From-zone_To) == 2){
    price += 150;
}
```

- We collaborated on most parts of the project, but I specifically did the part of the menu where you select a ticket type. It is fairly simple: it asks you if you want a single ticket, 7-day ticket or 30-day ticket. You choose them by either typing 1, 2 or 3 where each corresponds to one type. If you type anything that isn't those numbers, the program will tell you it's invalid and ask for a new input.
- I also did the part of ticket_Price() function where it calculates how many zones you've travelled in. It finds the distance travelled by using the absolute value of the difference between departure zone and arrival zone. By using the absolute value, it always gets a positive value no matter which way you're travelling.

Nablis Ogubamichael Gebrehiwet :

- The Civil status part is where a user chooses between five different status (student, solder, senior, child, and adult) so that the program can make a ticket from the information that was gather from the user.
- This part of the program is vital to the pricing system because different types of civil status gets different types of discounts. if a user is a student then it gets a 20% discount. price = price * 0.8; if a user is a solder then it gets a 30% discount. price = price * 0.7 if a user is a senior then it gets a 50% discount. price = price * 0.5 if a user is a child then it gets a 100% discount. price = 0 if a user is an adult then it won't get a discount. In order to get the right discount, one has to have a full information about the stations, zone and types of ticket.
- Even though most part of the program was done in group collaboration, specifically, this part of the menu and ticket was done by me.

Evaluation

Nablis : He was consistent and good on his work list. He would come to all the meetings and discuss topics about the project.

Jan : He was consistent and good on his work list. He would come to all the meetings and discuss topics about the project.

Alexandros: He was consistent and good on his work list. He would come to all the meetings and discuss topics about the project.

Summary

All in all, the program's purpose is to create a train ticket for the user to use when boarding. The program will take the input of the user, calculate the price for that ticket and finally print the ticket for the user to use. Everyone in the group worked very effectively and were diligent with their work list. This project has given us a very useful exposure to the C++ programming language.

References

Course's slides.

Tutorial videos on YouTube on C++.