# Code Documentation

C++ | Streaming Payment Calculator

Kendall Vargas

Programmer | Tester

December 19th, 2024

## INTRODUCTION

This document describes the overall functionalities of the code and the requirements of the project.

The C++ Streaming Payment Calculator assists in **calculating the monthly streaming consumption fee** based on the hours watched on a streaming platform.

Although the project context focuses on streaming-related content, the calculator can be applied to any scenario where different penalties are applied based on specific consumption metrics.

## PROJECT OBJECTIVES

The system provides a comprehensive solution for:

- **Calculating** the price of the streaming service consumed based on the hours watched.
- Applying different **'penalties'** and adjusting the price based on the hours watched.
- **Adjust** the calculation price depending on whether the user is tax exempt or not.
- Displaying a receipt with all the **necessary information** and the **final price** to be paid.

## TECHNICAL REQUIREMENTS

The main menu consists of 3 options:

1. Enter data and information to calculate the payable amount.

2. Display a receipt with all the required data entered in menu option 1.

3. Close the system.

The user is asked for these details:

- Client ID
- Full Name
- Invoice number
- Number of hours watched on the platform
- If it is exempt from taxes

**Conditions to be met in the code:**

1. The menu repeats until a **correct option is entered** (invalid: incorrect letters or numbers).
2. The client ID must not allow letters or numbers that are longer than 9 digits.
3. The invoice number must not allow letters or numbers that are longer than 6 digits.
4. Total hours viewed must not allow letters or numbers less than 1 or greater than 720.
5. The tax menu repeats **until a correct letter is entered**.
6. The return to the main menu repeats **until a correct letter is entered.**
7. Access to option 2 **is not possible** without having gone through option 1.

**Receipt:**

The receipt must include the following details:

Personal information:

1. ID number
2. Name of the service’s owner
3. Invoice number
4. Number of watched hours

Invoice details:

1. Fixed fee Price
2. Content Protection Price
3. Additional Consumption
4. Taxes
5. Total pay amount in USD
6. Code for completing payment

**Logic of calculation for the final price**

There are already 2 fixed payments each month, which are:

- Fixed Fee: 10 USD
- Content Protection: 5 USD

The streaming hour has a fixed price of 2 USD per hour. For the additional hourly charging, it works as follows:

Category 1: The first 10 hours have **no additional charges**.

Category 2: After 10 to 50 hours consumed, an additional 10% is applied to the final price + the price of the first 10 hours consumed.

Category 3: From 51 to 100 hours, the price of the first 10 hours consumed + 10% of the previous 40 hours, and 20% is applied to the final price based on the hours consumed in this category.

Category 4: Above 100 hours, the price of the first 10 hours consumed(first category) + 10% of the previous 40 hours (second category) + 20% of the third category + 30% applied to the hours above 100 hours.

Example of the calculation:
Let's say I've seen 720 hours in the whole month:

> $15 fixed expense
> 10 x 2= $20
> 40 x 2 * 1,10 = $88
> 50 x 2 * 1,20 = $120
> 620 x 2 * 1,30 = $1612
>
> Without taxes: $1855
> With taxes: $2096,15

The remaining hours after the calculation of the first category are included in the next category, and so on with the other categories.

The tax price is 13% of the final total payment price. If the client is tax exempt, this additional price is not added.

## TECHNICAL SPECIFICATIONS:

### Functions and processes used throughout the program

To store the information, I created 2 structures:

The first structure is called InvoiceData which stores the different data from the price calculation, along with the fixed payments as well:

```
struct InvoiceData

{

   float totalToPayWithIVA;

   float ivaAmount;

   int fixedFee;

   int contentProtection;

   int totalStreamingConsumption;

} invoice;
```

The second structure is called informationCustomer which stores the data inserted by the user, like name, invoice, ID and the watched hours.

```
struct informationCustomer

{

   string fullName;

   string invoice;

   string clientID;

   string totalWatchedHoursCalc;

   int totalWatchedHours;

} info;
```

Now, let’s break down the functions based on their use in the system:

**Menu**(): Presents the menu to the user and asks them which section they want to enter.

1. Enter streaming usage data for the month:

**DataEntry**(): Here are stored all the questions asked to the user about the ID, invoice number, full name and hours watched, this is stored in the informationCustomer structure.

**exceptIva**(): This asks the user if they are exempt from taxes or not, which stores the variable that will be used in the calculation to add the tax to the total payment amount or not.

2. Generate the monthly bill for streaming service:

**calculation**(): The calculation is saved in this section, breaking down how it is calculated:

Considering the requirements, there are different penalties for the hours consumed, and the **hourly rate of $2 per hour.**

For the first 10 hours consumed, I used the min() function, which takes as its first parameter the one inserted by the user, and as its second parameter the maximum number I can take from it, so in this case if the user inserts 100, it will only discount 10, since it is the maximum number I put in the second parameter, and the rest it will be passed to the next category.

I saved this in a variable, to then multiply it by 2, which is the price of the hourly rate.

For categories 2, 3 and 4, the **only difference** is that it will include the ‘penalties’ mentioned above, and that the maximum numbers are different:

Category 2: The range of hours taken is from 11 to 40 > these hours are multiplied by 2 > the ‘penalty’ that is applied is to multiply the previous result by 1.10, which means that there is an additional charge of **10% on the total charge**.

Category 3: The range of hours taken is 51 to 100 > these hours are multiplied by 2 > the 'penalty' applied is to multiply the previous result by 1.20, meaning there is an additional charge of **20% on the total charge.**

Category 4: The range of hours taken is 100 to 670 > these hours are multiplied by 2 > the 'penalty' applied is to multiply the previous result by 1.30, meaning there is an additional charge of **30% on the total charge.**

**wholeCalculation**(): This function is implemented inside the calculation() function, I decided to do this to separate the total amount and then based on the user's response, whether it is tax exempt or not, recalculate the charge:

If the user said that he is tax exempt: the price remains the same.

If the user said that they are not tax exempt: the total amount is multiplied by 1.13, that is, **13% of the total amount.**

**generateInvoice**(): This is the section where the **complete receipt is presented**, including all customer information and information related to the invoice, fulfilling the requirement of what is required to be shown in this section.

**RandomNumber**(): This is called in the generateInvoice() function, the main reason is to display a code that the user can use to complete a payment (which is fictitious, since no payment will be made).

Used in both sections:

**returnMenu**(): This question asks the user if they would like to return to the main menu. Valid input is 'Yy' or 'Nn', if the answer is invalid or 'Nn' the question will be repeated.

Besides the above functions, I have used some additional functions which are very self-explanatory:

clearScreen(): Whenever I need to clear the screen to display clean information and remove all previous rows, this feature comes in handy.

Regex:

bool **isValidName**(const string &): Validates that the name only includes relevant characters.

bool **isValidMenuInput**(const string &): Validates that only accepts numbers from 1 to 3.

bool **isValidYesNoInput**(const string &): Validates that the string only accepts Yy or Nn character.

bool **isValidID**(const string &): Validates that the ID only has 9 digits from 0-9.

bool **isValidInvoice**(const string &): Validates that a numeric Invoice matches a length of 6 digits.

bool **areValidHours**(const string &): Regex for the hours value, between 1 and 720 are allowed.

## Option 3 (Exit)

No function or processed used, it is just message “Exiting the system, see you soon!”.