




# Test Plan

C++ Code Streaming Project

Kendall Vargas

QA Software Tester



November 18th, 2024.

Contents

Project Overview: ..... 2

Requirements: ..... 2

Business Logic: ..... 3

Objectives:..... 3

In Scope: ..... 4

Out of Scope: ..... 4

Test Strategy: ..... 5

Exit Criteria:..... 5

Testing Tools:..... 5

## Project Overview:

The C++ code consists of a short system that calculates the total amount to be paid **based on hours watched** on a streaming platform.

The main menu consists of 3 options:

1. First, enter the data and information so that the system calculates the amount.
2. The second, show on a receipt all the required data based on what was entered on menu 1.
3. The third, leave the system completely.

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

## Requirements:

- The main menu includes Entry section – Receipt Generation – Exit.
- The menu repeats until a correct option is entered (invalid: incorrect letters or numbers).
- The payment calculation is expected based on the different categories (Business Logic section).
- Client ID must not allow letters or numbers other than 9.
- Invoice number must not allow letters or numbers other than 6.
- Total hours viewed must not allow letters or numbers for less than 0 or greater than 720.
- The tax menu repeats until a correct letter is entered.
- The return to the main menu repeats until a correct letter is entered.
- The Receipt Generation should display: Client ID, Full Name, Invoice Number, Hours Watched, final price calculation and code generation to complete payment.
- The system should handle unexpected inputs and errors without closing the system.

**Important:** You cannot access option 2 without having gone through option 1.

## Business Logic:

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

fixedFee: 10 USD

contentProtection: 5 USD

The streaming hour has a fixed price of 2 USD per hour. For the additional hourly charge, 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:

Supposed I have watched in the whole month 720 hours:

\$15 fixed expense

$10 \times 2 = \$20$

$40 \times 2 \times 1,10 = \$88$

$50 \times 2 \times 1,20 = \$120$

$620 \times 2 \times 1,30 = \$1612$

Without taxes: \$1855

With taxes: \$2096,15

The hours are subtracted when they are included in the respective categories, and the other rest of the hours move to the next category and so on.

For the tax price, it is 13% of the final total price of the payment. This is if the client is not exempt from taxes, if they are exempt, this additional price is not added.

## Objectives:

For the correct delivery of this C++ project, I need to test all the code and validate it works correctly based on the project requirements.

Since this project does not have a GUI as such, it can be said that it is more related to a static and white box analysis and testing, however, in the data entry, I used some black box testing.

### In Scope:

#### ❖ Option 1: “White box”:

- I need to perform sentence coverage and decision coverage, since I know the structure of the code internally. This can be performed by going through the different decisions of the code to validate each output is correct and represents the respective path.
- Check that there’s no errors on the IDE or variables not used.
- Dead code, or paths that are not reachable as a normal user.

#### ❖ Option 2: Black Box:

With the output, a small screen is received to insert data and then present the receipt result. This can be considered as our “front-end” to validate the code is working as expected before implementing this to a real page:

- Verifying all menus are working as expected.
- Apply testing techniques in the input of data (int, string, float).
- Validate the response of the receipt includes expected data.
- Calculate manually if the amount given is correct.

#### ❖ Experienced based techniques:

- Apply positive and negative scenarios across menus and all different inputs.
- Validate the correct structure of menus, receipt, and grammar issues.

Since the project is small, I will tackle mainly Black Box testing and add some documentation about White Box testing to the TC sheet.

### Out of Scope:

Database: This project does not include a database, due to this, there’s no way to store an ID to a specific object when entering the name and the ID, or matching the information with an invoice ID.

Multiple receipts: Currently only 1 receipt can be generated. If we go over the flow 1 and 2 and then go back to 1, some prices are going to be added to the current price, showing an incorrect result.

## Test Strategy:

### - Code creation and understanding the requirements:

- Create the code based on the requirements to also understand how to perform the respective testing for the whole flow.

### - Preparing Test Cases:

The creation of test cases for the different features and calculations will be covered by some testing techniques to cover the requirements:

- ❖ Boundary Value Analysis
- ❖ Exploratory Testing: Perform exploratory testing besides execution of the Test Cases.
- ❖ Sentence coverage and Decision coverage
- ❖ E2E Testing: Test the normal E2E flow covering user actions and responses.

### - Executing Test Cases:

- Executing the created test cases and logging the test result in the respective documentation.
- Report the defects by using the assigned testing tool for the respective fix.

### - Bug Fix:

- Once all Test Cases have been executed and all issues have been reported, fix > and retest.

## Exit Criteria:

- The system should handle unexpected inputs and errors without closing the system.
- No high priority(blocker) issue is remaining in the system.
- The price calculation has the correct parameters and the respective %.

## Testing Tools:

Process	Tools
Test Case Creation	Word
Bug Reporting	Notion
Test Case Execution	Code::Block / Manual
Codification	Code::Block