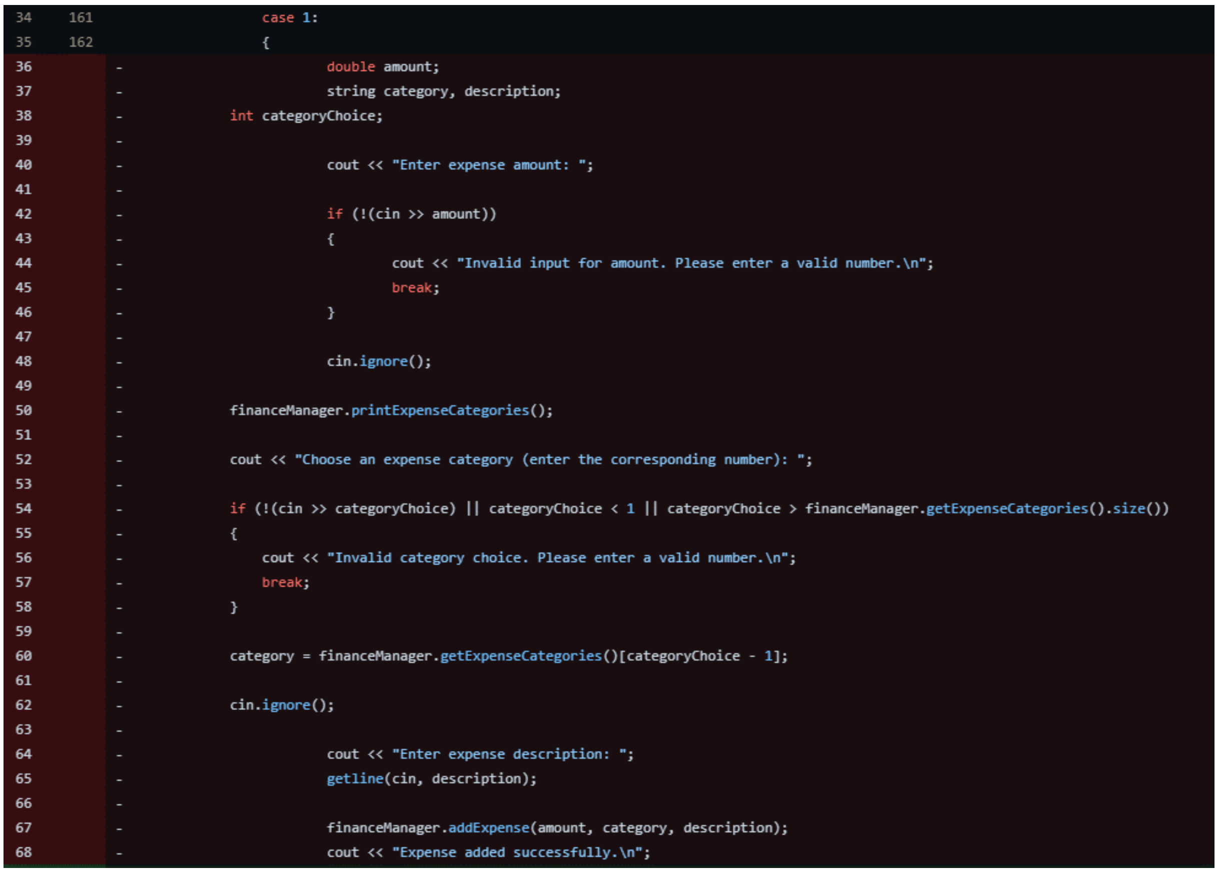
Clean Code Development

1. **Reducing amount of code in void main() for better view and creating functions instead**

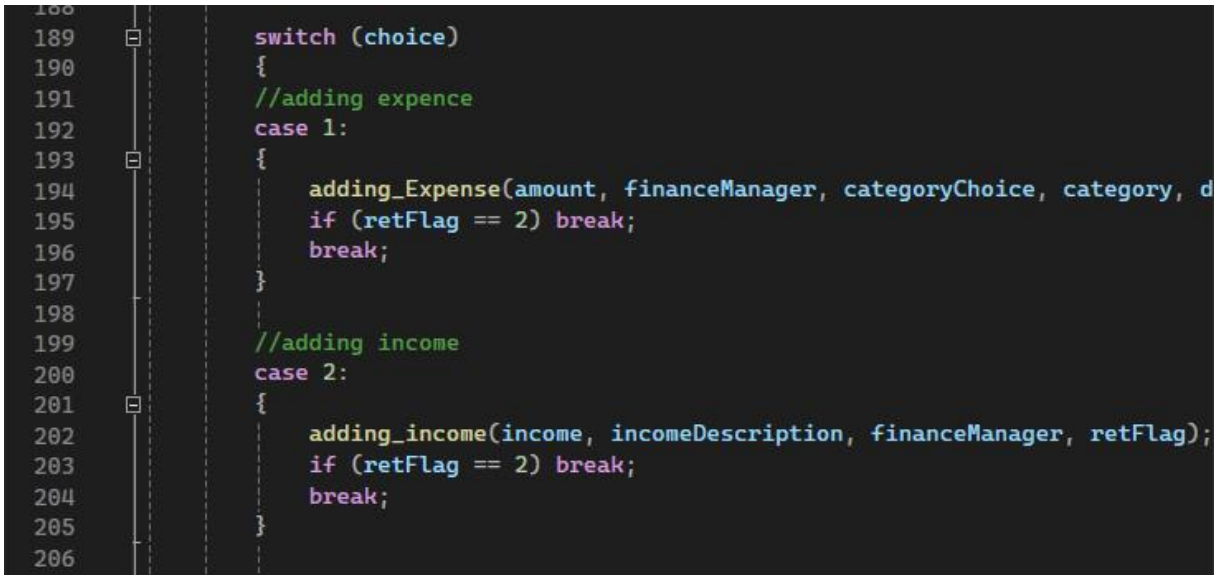
Reducing amount of code in switch helped me with organizing space and makes it more readable.

THEN:

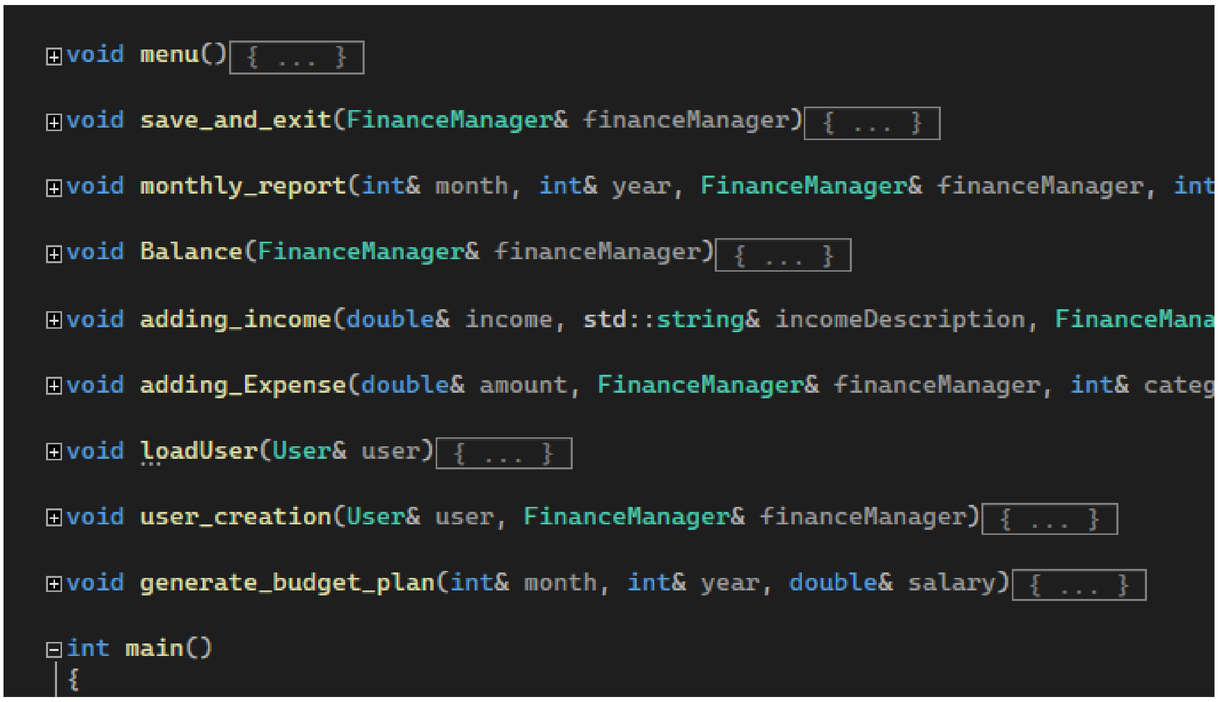


```
                    case 1:
                    {
-                           double amount;
-                           string category, description;
-               int categoryChoice;
-
-                           cout << "Enter expense amount: ";
-
-                           if (!(cin >> amount))
-                           {
-                                   cout << "Invalid input for amount. Please enter a valid number.\n";
-                                   break;
-                           }
-
-                           cin.ignore();
-
-               financeManager.printExpenseCategories();
-
-               cout << "Choose an expense category (enter the corresponding number): ";
-
-               if (!(cin >> categoryChoice) || categoryChoice < 1 || categoryChoice > financeManager.getExpenseCategories().size())
-               {
-                   cout << "Invalid category choice. Please enter a valid number.\n";
-                   break;
-               }
-
-               category = financeManager.getExpenseCategories()[categoryChoice - 1];
-
-               cin.ignore();
-
-                           cout << "Enter expense description: ";
-                           getline(cin, description);
-
-                           financeManager.addExpense(amount, category, description);
-                           cout << "Expense added successfully.\n";
```

NOW:



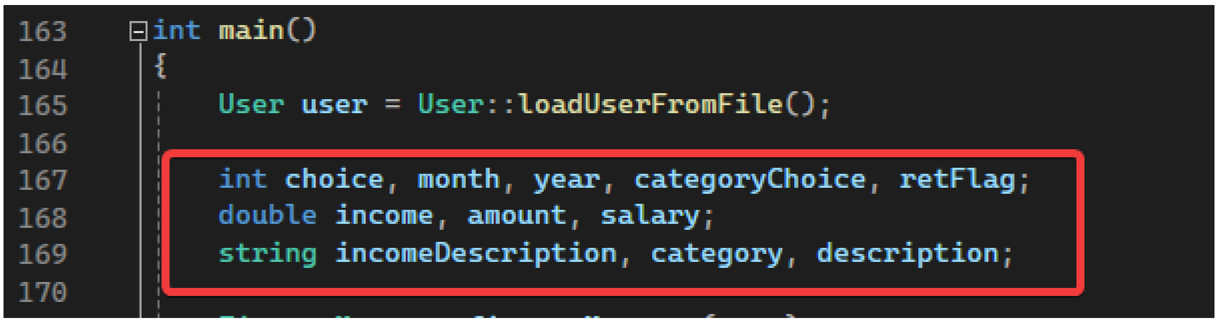
```
            switch (choice)
            {
            //adding expence
            case 1:
            {
                adding_Expense(amount, financeManager, categoryChoice, category, d
                if (retFlag == 2) break;
                break;
            }

            //adding income
            case 2:
            {
                adding_income(income, incomeDescription, financeManager, retFlag);
                if (retFlag == 2) break;
                break;
            }
```



```
void menu() { ... }

void save_and_exit(FinanceManager& financeManager) { ... }

void monthly_report(int& month, int& year, FinanceManager& financeManager, int

void Balance(FinanceManager& financeManager) { ... }

void adding_income(double& income, std::string& incomeDescription, FinanceMana

void adding_Expense(double& amount, FinanceManager& financeManager, int& categ

void loadUser(User& user) { ... }

void user_creation(User& user, FinanceManager& financeManager) { ... }

void generate_budget_plan(int& month, int& year, double& salary) { ... }

int main()
{
```

2. Explanatory variables

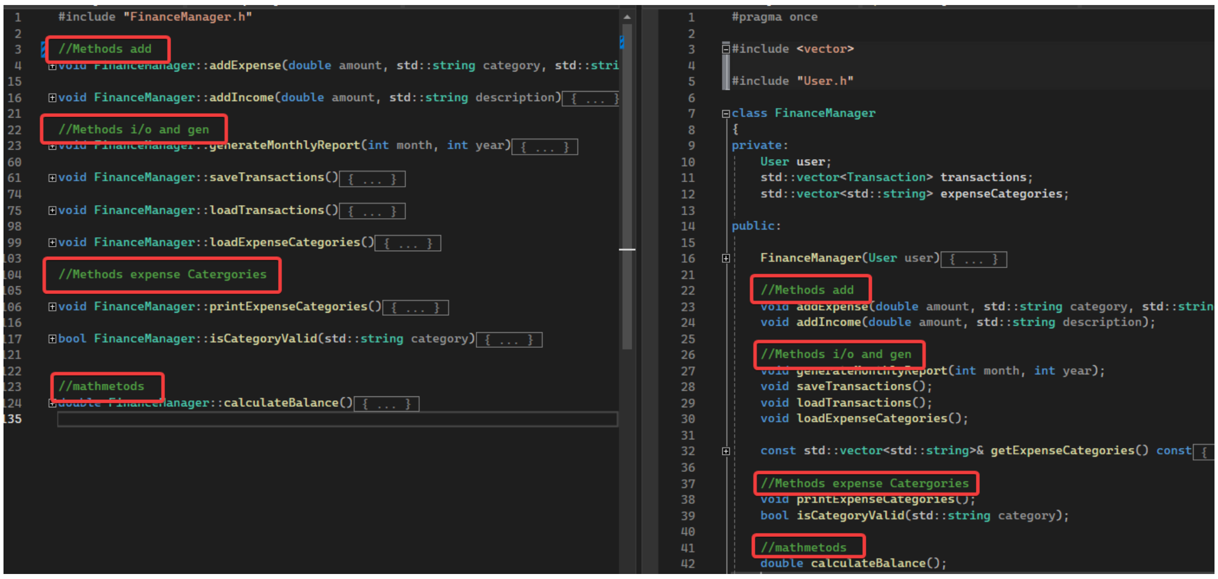
explanatory variables helped me with knowing what should I use. I did not wasted my time searching for "how do I called this variable?", names are clear and unique for me. Some people might be a little bit confused with those names but when the will see my POV it will be easy to use them for everyone



```cpp
int main()
{
    User user = User::loadUserFromFile();

    int choice, month, year, categoryChoice, retFlag;
    double income, amount, salary;
    string incomeDescription, category, description;
```

3. **Good comments**

Since my functions are self-explanatory I barely use comments. I mostly use comments to point the most important things to better catch the thing where is something. Also when it will come to the point of big algorithm comments might explain complexity of logic

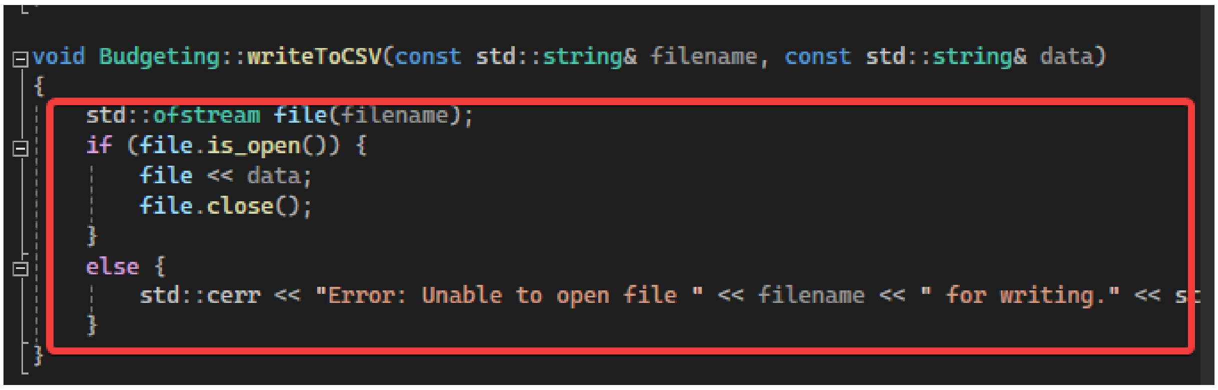


```cpp
#include "FinanceManager.h"

//Methods add
void FinanceManager::addExpense(double amount, std::string category, std::stri

void FinanceManager::addIncome(double amount, std::string description){ ... }

//Methods i/o and gen
void FinanceManager::generateMonthlyReport(int month, int year){ ... }

void FinanceManager::saveTransactions(){ ... }

void FinanceManager::loadTransactions(){ ... }

void FinanceManager::loadExpenseCategories(){ ... }

//Methods expense Catergories

void FinanceManager::printExpenseCategories(){ ... }

bool FinanceManager::isCategoryValid(std::string category){ ... }


//mathmetods
double FinanceManager::calculateBalance(){ ... }
```

```cpp
#pragma once

#include <vector>

#include "User.h"

class FinanceManager
{
private:
    User user;
    std::vector<Transaction> transactions;
    std::vector<std::string> expenseCategories;

public:

    FinanceManager(User user){ ... }

    //Methods add
    void addExpense(double amount, std::string category, std::strin
    void addIncome(double amount, std::string description);

    //Methods i/o and gen
    void generateMonthlyReport(int month, int year);
    void saveTransactions();
    void loadTransactions();
    void loadExpenseCategories();

    const std::vector<std::string>& getExpenseCategories() const{

    //Methods expense Catergories
    void printExpenseCategories();
    bool isCategoryValid(std::string category);

    //mathmetods
    double calculateBalance();
```

4. **Error handling**

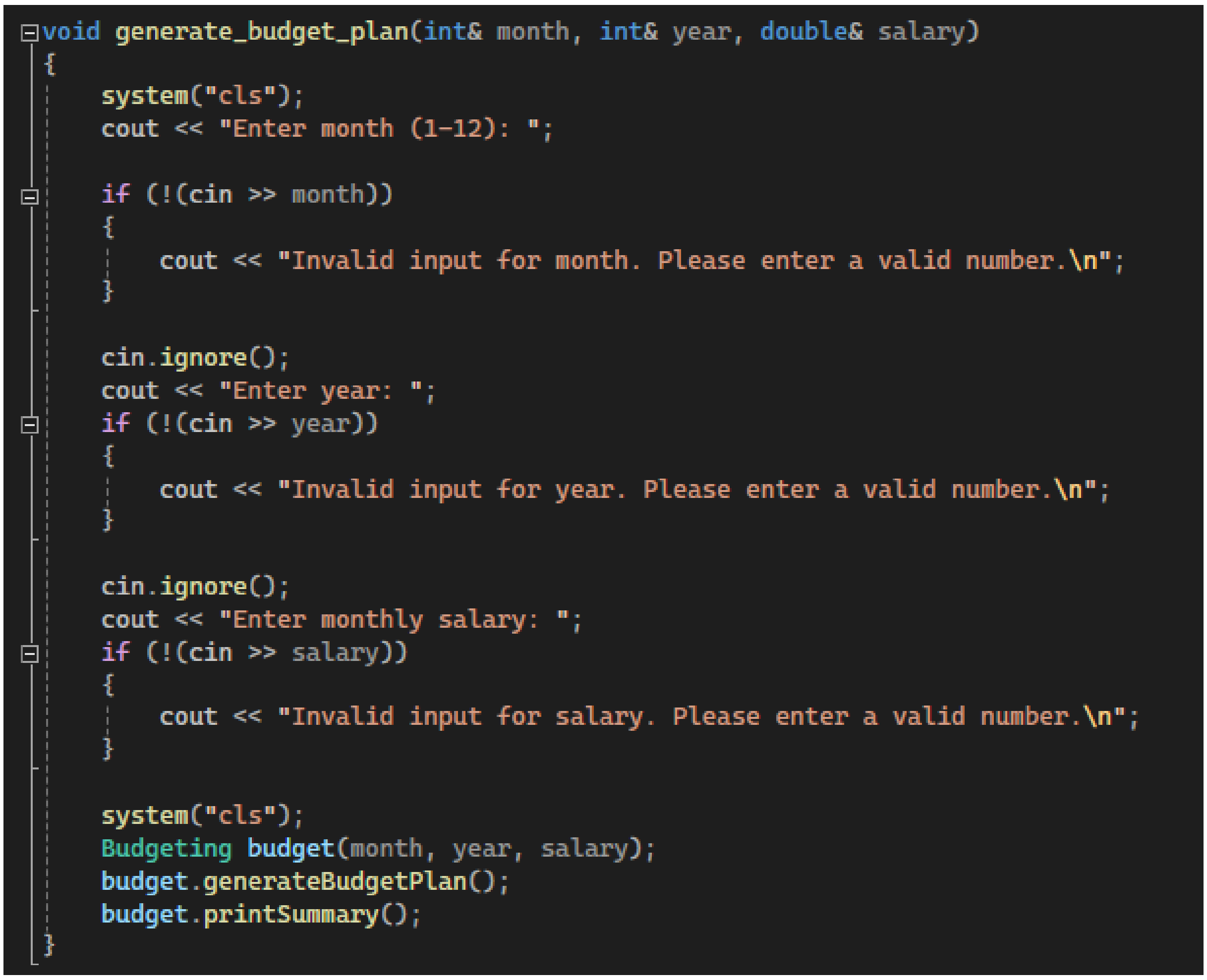
Error handling in my case I used for files, sometimes user is unavailable somehow to create file so it is important to know what exactly happened



```cpp
void Budgeting::writeToCSV(const std::string& filename, const std::string& data)
{
    std::ofstream file(filename);
    if (file.is_open()) {
        file << data;
        file.close();
    }
    else {
        std::cerr << "Error: Unable to open file " << filename << " for writing." << s
    }
}
```

5. **Intuitive Function names and clear intension of the function/method**

Intuitive function names explains me a lot with what is going on. It helps me plan better and use them by calling the thing what they will be doing. It also helps me find a bug when it comes to unit test because I know which function/functions are responsible for it



```cpp
void generate_budget_plan(int& month, int& year, double& salary)
{
    system("cls");
    cout << "Enter month (1-12): ";

    if (!(cin >> month))
    {
        cout << "Invalid input for month. Please enter a valid number.\n";
    }

    cin.ignore();
    cout << "Enter year: ";
    if (!(cin >> year))
    {
        cout << "Invalid input for year. Please enter a valid number.\n";
    }

    cin.ignore();
    cout << "Enter monthly salary: ";
    if (!(cin >> salary))
    {
        cout << "Invalid input for salary. Please enter a valid number.\n";
    }

    system("cls");
    Budgeting budget(month, year, salary);
    budget.generateBudgetPlan();
    budget.printSummary();
}
```