Sprawozdanie projektu pt. „Bank – udzielanie kredytu”

Data złożenia sprawozdania: 01.06.2023

Przedmiot: Programowanie Komputerowe

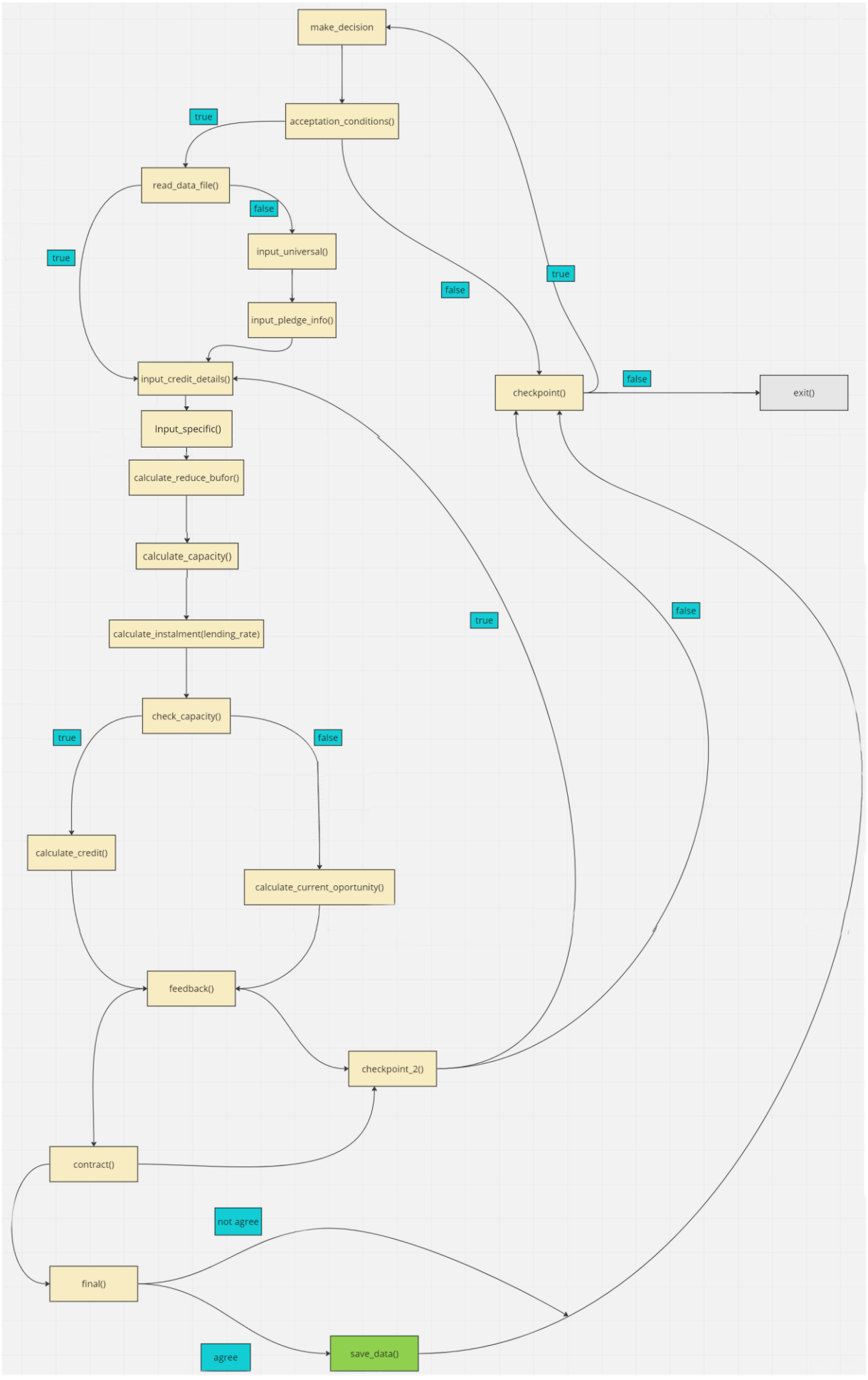
Autor: Karol Pitera

Grupa dziekańska: 1

Politechnika Śląska

Spis treści:

Pomocniczy schemat wywoływania funkcji programu



Wprowadzenie

W trakcie semestru realizowałem projekt pt. „Bank – udzielanie kredytu”, którego celem było stworzenie programu dającego możliwość udzielenia kredytu potencjalnemu klientowi banku.

W sprawozdaniu zwracam się do czytelników per Państwo, przyjąłem taką formę, ponieważ w podobny sposób będę się zwracał w przyszłości do odbiorców moich projektów.

Działanie programu zaprezentuję za pomocą opisu punktowego, zalecam czytać go wspierając się pomocniczym schematem wywoływanych funkcji zawartym na drugiej stronie:

Niebieskim kolorem wskazane będą funkcje ze schematu, odnoszące się do danego punktu.

W poniższym punktowym opisie występują przeskoki pomiędzy poszczególnymi punktami, czytelnik jest o nich informowany, gdyby jednak taka informacja się nie pojawiła, wtedy program przechodzi do kolejnego punktu.

„Wykonywanie Programu”

1) Użytkownik określa się jako klient indywidualny, lub klient reprezentujący firmę.

   Następnie wybiera jedną z wyświetlonych opcji kredytowych banku;

   make_decision()

2) Klient zostaje poproszony o akceptacje warunków kredytu, ma możliwość odmowy:

   - jeżeli zaakceptuje warunki, to następuje przejście do następnego punktu;

   - w przeciwnym wypadku nastąpi przeniesienie do punktu nr 18;
   acceptation_conditions()

3) (Opcjonalna) Możliwość wczytania danych z pliku (zwierającego dane personalne, miesięczne przychody i wydatki, oraz opcjonalny zastaw).
   - jeżeli plik zostanie wczytany to następuje przejście do punktu 6,
   - w przeciwnym wypadku program zostaje przekierowany do następnego punktu;

   *Proszę zachować szczególną ostrożność przy wpisywaniu poprawnej nazwy pliku. Program nie obsługuje polskich znaków takich jak: ą,ę itd.

*Dane powinny być zapisane w pliku tekstowym w odpowiedniej kolejności ukazanej w poniższym przykładzie:

legal_name: Jerzy Grebosz
age: 65
birthplace: Krakow
evidence_number: CPP211232
PESEL: 12345678911
monthly_income: 6000
monthly_spending: 3500
Would_you_like_pledge(write_true_or_false): true
*(opcional)pledge_name: bike
*(opcional)pledge_value: 10000

*Przykładowe pliki zawarte w projekcie z przeznaczeniem do testowania oprogramowania:
- test.txt
- test2.txt
- test3.txt

read_data_file()

4) Jeżeli dane nie zostały wczytane z pliku to klient jest poproszony o wpisanie danych personalnych, miesięcznych przychodów i wydatków.

   *funkcja została nazwana „input_universal()", przez fakt że dane w niej udzielone są potrzebne we wszystkich rodzajach kredytów.

   input_universal()

5) Następnie usługobiorca (opcjonalnie) ma możliwość zastawić dowolne dobro lub nadpisać wcześniejszy zastaw. W zależności od jego wartości zostanie zwiększona zdolność kredytowa klienta;

   input_pledge_info()

6) Wpisanie przez użytkownika kwoty kredytu o którą zamierza się ubiegać, oraz ilości lat lub miesięcy, w których zamierza spłacić swoje zobowiązanie;
   input_credit_details()

7) Udzielenie charakterystycznej informacji dla danej opcji kredytowej np. wpisanie nazwy banku, w którym klient ubiegający się o kredyt konsolidacyjny jest zadłużony; *input_specific()*

8) W tym miejscu program wykonuje obliczenia na których podstawie będzie oceniał zdolność kredytową klienta.

Obliczane informacje:

- *safety_bufor – bufor bezpieczeństwa* (obliczany w funkcji, *calculate_capacity()*) – jest to liczba równa 20% miesięcznych zarobków klienta. Zostaje ona odejmowana od zdolności kredytowej klienta w celu zapobiegania niewypłacalności interesantów (bank zakłada zdarzenia losowe mogące zmniejszyć miesięczny budżet klientów).

- *calculate_reduce_bufor()* – redukcja bufora, obliczana na podstawie wartości zastawu klienta, zmniejsza ona bufor bezpieczeństwa. Jeżeli spłata kredytu jest zabezpieczona pewnym dobrem np. działką budowlaną. Bank może pozwolić kredytobiorcy na wzięcie relatywnie większego kredytu.

- *calculate_capacity()* – funkcja oblicza zdolność kredytową kredytobiorcy (na podstawie miesięcznego dochodu, bufora bezpieczeństwa oraz ewentualnego zastawu).

- *calculate_instalment(lending_rate)* – funkcja oblicza miesięczną ratę kredytu z uwzględnieniem oprocentowania.

9) Sprawdzenie czy zdolność kredytowa jest większa od wysokości miesięcznej raty zobowiązania.
   - Jeżeli zdolność kredytowa okaże się wystarczająca, następuje przejście do kolejnego punktu,
   - W przeciwnym wypadku program zostaje przekierowany do pkt. 11.
   check_capacity()

10) Obliczenie wysokości kredytu do spłaty (z uwzględnieniem RRSO).
   calculate_credit()

   Następnie program przechodzi do pkt 12.

11) Obliczana zostaje maksymalna wysokość kredytu (z uwzględnieniem RRSO) jaką bank może udzielić klientowi przy obecnej zdolności kredytowej i wybranym okresie spłaty.
   calculate_current_oportunity(lending_rate)

12) Udzielenie informacji zwrotnej:

- Jeżeli zdolność kredytowa klienta jest wystarczająca w stosunku do wysokości kredytu to zostaje on o tym powiadomiony (przejście do punktu 14).

- W przeciwnym wypadku zostaje poinformowany o negatywnym wyniku weryfikacji oraz aktualnej kwocie kredytu, o który może się ubiegać. (przejście do punktu 13)

feedback()

13) Zapytanie klienta czy zamierza zmienić informacje kluczowe takie jak: wysokość kredytu, okres spłaty kredytu oraz ewentualny zastaw.

- Jeżeli tak, to działanie programu zostaje skierowane do pkt 6.

- W przeciwnym wypadku działanie programu jest przekierowane do pkt 18.

checkpoint_2()

14) Wypisane na ekran zostaje podsumowanie kredytu z najważniejszymi informacjami takimi jak: wysokość kredytu, okres spłaty kredytu, całkowita wysokość zobowiązania do pokrycia, miesięczna rata,
oraz opcjonalnie nazwa zastawionego dobra i jego oszacowana wartość.

Następnie wyświetlony jest kontrakt zawierający zobowiązania kredytobiorcy względem banku. Obejmuje on również konsekwencje grożące interesantowi za niewywiązywanie się z postanowień umowy.

contract()

15) Użytkownik zostaje poproszony o zaakceptowanie warunków zawartych w kontrakcie, klient ma możliwość braku akceptacji:

- jeżeli klient zaakceptuje kontrakt zostaje przeniesiony do punku 16;

- w przeciwnym wypadku program przechodzi do punktu 17

contract()

16) Klient jest powiadomiony o finalizacji kredytu zakończonej sukcesem, a najważniejsze informacje zostają zapisane w pliku o nazwie X_Imie Nazwisko.txt (gdzie X oznacza numer klienta).

*Wyjątkiem od reguły jest kredyt hipoteczny w którym po akceptacji warunków kontaktu, należy jeszcze wpłacić część własną (czyli 10% kwoty kredytu) dopiero potem następuje finalizacja kredytu (czyli uzyskanie środków przez klienta). Brak wpłaty części własnej jest równoznaczny z brakiem akceptacji kredytu.

final()

save_all_data()

17) Klient jest informowany, że zgoda na warunki kontraktu jest niezbędna, natomiast jego wcześniejszy kredyt zostaje anulowany.
final()

18) Następnie klient dostaje możliwość skorzystania z kolejnej/innej oferty banku lub zakończenia programu.

- jeżeli klient wpisze "bank offers" zostanie przeniesiony do punktu numer 1,

- natomiast gdy wpisz "!exit" program zostanie zakończony.

checkpoint()

Podsumowując warto zauważyć, że:

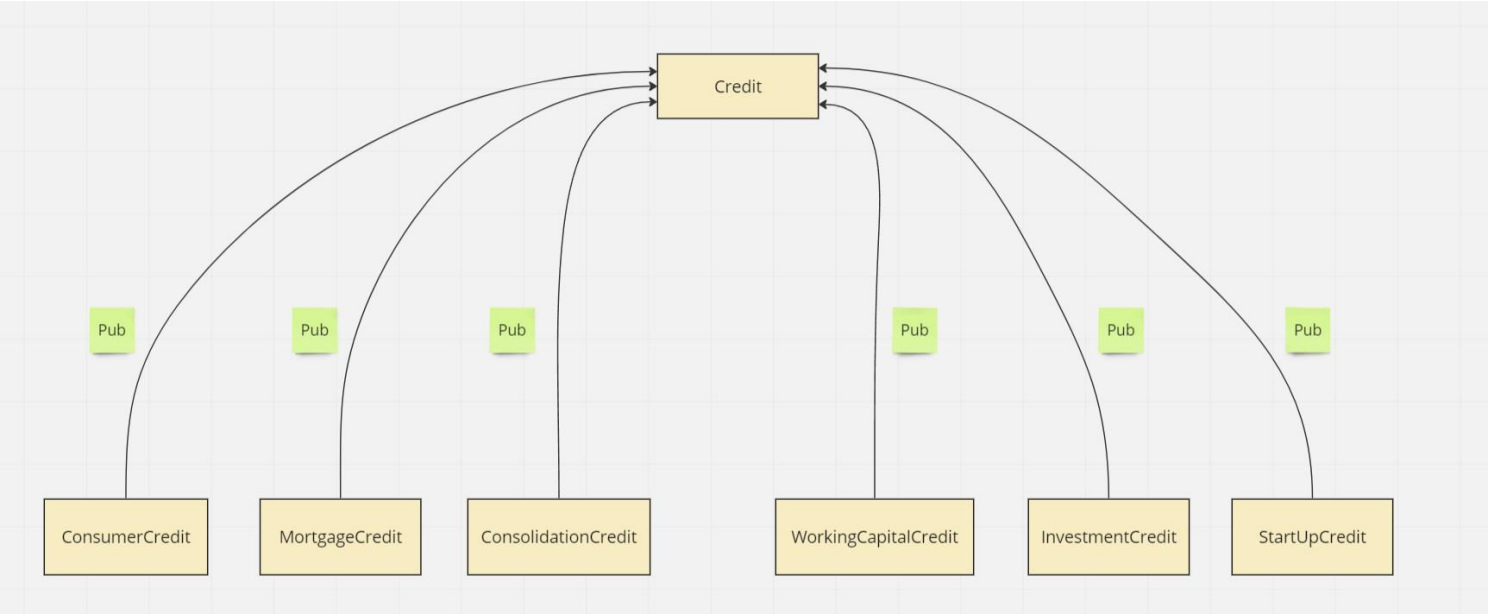
- dane zapisywane są do pliku jedynie, gdy kredyt zostanie udzielony, zatem do bazy danych będą trafiali jedyni klienci najistotniejsi, ułatwi to pracę pracownikom banku.

- program świadomie obniża zdolność kredytową klienta, przez co zwiększa prawdopodobieństwo, że w przyszłości kredyt zostanie spłacony na czas.

Zatem Pański bank będzie poświęcał mniej czasu, a co za tym idzie pieniędzy na odzyskanie należności od niestabilnych finansowo klientów.

- ukoronowaniem owego oprogramowania jest łatwość w jego rozbudowie. Każda opcja kredytowa przybywa tą samą drogę, zatem jeżeli Pański Bank będzie zamierzał wprowadzić nową opcje kredytową, wystarczy zmodyfikować powyższy program.

To pozwoli z kolei obniżyć koszty związane z zapleczem informatycznym.



Schemat dziedziczenia klas programu

Program zawiera 7 klas, jedną abstrakcyjną („Credit”) oraz 6 klas pochodnych.

Dziedziczenie odbywa się w sposób publiczny co pozwala na dostęp do publicznych składowych klasy „Credit” wewnątrz klas pochodnych.

Najważniejsze metody klasy abstrakcyjnej zostały opisane i wyszczególnione na stronach 3-7, wyjątkiem jest funkcja make_decision() będąca funkcją globalną.

```
Credit* make_decision()
{
    int target, kind;
    Credit* current = nullptr;

    cout << "\n\nWould you like recive a credit for individual customers or a company credit? ";
    cout << "\nTo select an individual credit, type in number: '1'.\nWhereas to choose a company credit, type in number: '2'.";
    cout << "\nEnter number: ";
    target = type_in_only_digit("1 or 2");
    cout << endl;
    if (target == 1)
    {
        cout << "Which credit kind you prefer?";
        cout << "\nPossible options:\n- nr 1: Mortage credit (RRSO: 10%),\n- nr 2: Consolidation credit (RRSO: 15%),\n- nr 3: Consumer credit (RRSO: 20%).";
        cout << "Type in number correct option: ";
        kind = type_in_only_digit("1, 2 or 3");
        cout << endl;
        switch (kind)
        {
        case 1:
            current = new MortgageCredit;
            break;
        case 2:
            current = new ConsolidationCredit;
            break;
        case 3:
            current = new ConsumerCredit;
            break;
        }
    }
```

Część funkcji make_decision demonstrująca tworzenie obiektu wybranej klasy zaalokowanego dynamicznie.

W funkcji „make_decision()” klient wybiera opcje kredytu np. (WorkingCapitalCredit, czyli kredyt obrotowy dla firm). To skutkuje powołaniem do życia obiektu tej klasy na który wskazuje pointer typu abstrakcyjnej klasy „current”. Ten wskaźnik zostanie zwrócony na końcu funkcji po czym zostanie przypisany do pointera „credit_kind” w funkcji executing_program(). Dzięki wskaźnikowi „credit_kind” uniwersalny interfejs dla każdego rodzaju kredytu wywoła metody z odpowiedniej klasy pochodnej, co pozwoli klientowi skorzystać z wybranej oferty banku.

```
Credit* credit_kind = make_decision();

int lending_rate = credit_kind->return_lending_rate();

bool acceptation_conditions = credit_kind->acceptation_conditions();
```

Część funkcji executing_program() *ukazującej wykorzystanie funkcji make_decision() oraz pointera credit_kind.*

Korzystanie z pamięci dynamiczniej niesie za sobą konieczność jej zwolnienia. Omawiany program czyści zaalokowaną pamięć pod koniec działania programu, kiedy jest ona już nie potrzebna.

```
        checkpoint_decision = credit_kind->checkpoint();

        delete credit_kind;
        credit_kind = nullptr;

    } while (checkpoint_decision);
}
```

Po wykonaniu funkcji checkpoint w której użytkownik podejmuje decyzje o skorzystaniu z kolejnej oferty banku lub zakończeniu działania programu, pamięć zaalokowana dynamicznie zostaje zwolniona. Natomiast wskaźnik, który na nią wskazywał zostaje ustawiony na wartość null

Klasy pochodne przechowują charakterystyczne dane takie jak oprocentowanie danej opcji kredytowej oraz uniwersalny zestaw odpowiednio nadpisanych funkcji wirtualnych.

Przykład:

```
class WorkingCapitalCredit : public Credit
{
    int lending_rate = 8;
    std::string bank_account_number;
    std::string NIP;

public:

    int return_lending_rate();
    bool acceptation_conditions();
    void input_specific();
    void final();
    void save_specific_data();
};
```

Charakterystyczne pola danej klasy:

Lending_rate – oprocentowanie;

Bank_account_number – numer konta bankowego klienta.

Stały zestaw odpowiednio nadpisanych metod wirtualnych.

Na podanym przykładzie widać, że klasa pochodna dziedziczy publicznie po klasie Credit, zatem będzie miała dostęp do jej publicznych składowych.

Funkcje globalne programu

Zacznę od podzielenia się z Państwem widokiem pliku nagłówkowego z funkcjami globalnymi (będę się do niego odnosił w dalszej części pracy)

```
Credit* make_decision();

void executing_program();

std::string type_in_legal_name();

std::string type_in_age();

std::string type_in_birthplace();

std::string type_in_evidence_number();

std::string type_in_PESEL();

int type_in_only_digit(std::string target);

std::string type_in_alpha_space(std::string target);

std::string type_in_digit_dash(std::string target);
```

Pierwsze dwie funkcje zostały już opisane powyżej:

- make_decision() – na stronie stronie 8;

- executing_program() – na stronach 3-7, opis jest zatytułowany „Wykonywanie programu”.

Kolejne funkcje są odpowiedzialne za pobranie danych od użytkownika, zawierają one podstawowe zabezpieczenia przed potencjalnymi błędami, jednak nie gwarantują wychwycenia wszystkich pomyłek klienta. Po pozytywnym przejściu weryfikacji funkcje zwracają pobrane dane.
Zaprezentuję Państwu dwie przykładowe funkcje pobierające dane. Pierwszą sprecyzowaną na konkretny rodzaj informacji, drugą stworzoną z myślą o pobieraniu znaków określonej klasy np. cyfr.

Przykład I
Funkcja: string type_in_legal_name():

```
string type_in_legal_name()
{
    bool correct = true;
    string name;
    do
    {
        cout << "\nType in your legal name: ";

        int alpha_counter = 0, surname_letters = 0, space_counter = 0;
        correct = true;

        getline(cin >> ws, name);

        for (int i = 0; i < name.size(); i++)
        {
            if (isalpha(name[i]))
            {
                alpha_counter++;
                surname_letters++;
            }
            if (isspace(name[i]))
            {
                surname_letters = 0;
                space_counter++;
            }
        }
        if (surname_letters < 2)
        {
            cout << "\Surname should include at least two letters";
            correct = false;
        }
        if (space_counter != 1)
        {
            cout << "\nLegal name should contain only one space between name and surname";
            correct = false;
        }
        if ((alpha_counter + 1) < name.size())
        {
            correct = false;
            cout << "\nLegal name can include only alphabets signs and one space sign";
        }
    } while (!correct);
    return name;
```

Funkcja ta prosi użytkownika o wpisanie konkretniej informacji jaką jest imię i nazwisko. Gdy użytkownik wpisze swoją odpowiedź, jej zawartość przechodzi weryfikacje podczas której sprawdzane jest czy spełnione zostały trzy warunki:

- długość nazwiska powinna zawierać przynajmniej dwie litery (tyle wynoszą najkrótsze polskie nazwiska),

- W ciągu znaków wymagana jest dokładnie jedna spacja (nie więcej, nie mniej),

- odpowiedź powinna zawierać jedynie litery oraz jedną spacje, inne znaki nie są dozwolone.

Jeżeli któryś z warunków nie zostanie spełniony, użytkownik jest poproszony o skorygowanie błędu na podstawie adnotacji np. „Nazwisko powinno zawierać przynajmniej dwie litery”.

Do sprawdzania czy dany znak pochodzi z określonej klasy znaków używam funkcji zawartych w bibliotece <ctype.h>

Przykład II
Funkcja: int type_in_only_digit(string target)

```
int type_in_only_digit(string target)
{
    bool correct = true;
    string chain;
    do
    {
        cout << "\nType in " << target << ": ";

        int digit_counter = 0;
        correct = true;
        getline(cin >> ws, chain);

        for (int i = 0; i < chain.size(); i++)
        {
            if (isdigit(chain[i]))
            {
                digit_counter++;
            }
        }
        if (chain.size() != digit_counter)
        {
            cout << "\nThe " << chain << " should include only digits";
            correct = false;

        }
    } while (!correct);
    float i_number = stoi(chain);
    return i_number;
}
```

Ta funkcja prosi użytkownika o wpisanie danej informacji złożonej tylko z cyfr której nazwa jest zapisana pod zmienną „target”. Jeżeli klient wpisze przez przypadek symbol, który nie jest cyfrą, program poprosi go o skorygowanie błędu.

W swoim projekcie tej funkcji używam do pobierania od użytkownika informacji o wysokości jego dochodów, wydatków oraz szacowanej wartości (opcjonalnego) zastawu.

Operacje na plikach oraz przeciążenie operatora

Na początku działania programu użytkownik jest zapytany czy zamierza wczytać dane zapisane w pliku tekstowym. Kolejność zapisywanych danych ma znaczenie i została podana na stronie 4.

```
void Credit::read_data_file()
{
    cout << "\n\nWould you like type your universal data from file?\
            \nIf you like write 'yes', otherwise write 'no': ";
    string answer = type_in_answer("yes", "no");

    if (answer == "yes")
    {
        string file_name;
        cout << "\n Type in file name with your data: ";
        getline(cin >> ws, file_name);

        ifstream file(file_name);
        if (file)
        {

            string temp, first_name, last_name;
            file >> temp >> first_name >> last_name;
            legal_name = first_name + " " + last_name;
            file >> temp >> age;
            file >> temp >> birthplace;
            file >> temp >> evidence_number;
            file >> temp >> PESEL;
            file >> temp >> monthly_income;
            file >> temp >> monthly_spending;
            file >> temp >> temp;
            if (temp == "true")
            {
                pledge = true;
                file >> temp >> pledge_name;
                file >> temp >> pledge_value;
            }
        }
        file.close();

        read_data = true;
    }
}
```

Funkcja odpowiedzialna za wczytanie danych z pliku

Jeżeli kredyt ostatecznie został udzielony, to najważniejsze informacje zostaną zapisane do pliku tekstowego nazwanego według wzoru: „x_imie nazwisko.txt” (gdzie x oznacza numer udzielonego kredytu przez program).

```
if (!credit_kind->lack_finalisation)
{
    ++(*credit_kind);
    credit_kind->save_all_data();
}
```

Instrukcja warunkowa sprawdzająca czy kredyt został sfinalizowany

W powyższym kodzie przy pomocy przeciążonego operatora „++” zostaje inkrementowany licznik udzielonych kredytów. Dzięki temu licznikowi każdy udzielony kredyt będzie zapisany w osobnym ponumerowanym pliku.

```
void Credit::operator ++()
{
    ifstream file("counter.txt");
    if (file)
    {
        file >> credit_counter;
    }
    file.close();

    credit_counter += 1;

    ofstream file2("counter.txt");
    if (file2)
    {
        file2 << credit_counter;
    }
    file2.close();
}
```

Instrukcja ukazująca przeciążenie operatora, którego zadaniem jest inkrementacja licznika credit_counter przechowywanego w pliku „counter.txt”

Po inkrementacji licznika następuje wywołanie metody „save_all_data()”. Ta funkcja wywołuje trzy inne metody, dwie pierwsze dostępne z poziomu klasy bazowej, natomiast trzecia jest nadpisana w klasie pochodnej.

```
void Credit::save_all_data()
{
    save_based_data();
    save_credit_data();
    save_specific_data();
}
```

Metoda: void save_all_data()

Zademonstrują Państwu jedną z owych trzech metod reszta działa na podobnej zasadzie.

```
void Credit::save_based_data()
{
    string counter = to_string(credit_counter);
    string name = counter + "_" + legal_name + ".txt";

    ofstream file(name);

    if (file)
    {
        file << "legal_name: " << legal_name;
        file << "\nage: " << age;
        file << "\nbirthplace: " << birthplace;
        file << "\nevidence_number: " << evidence_number;
        file << "\nPESEL: " << PESEL;
        file << "\nmonthly_income: " << monthly_income;
        file << "\nmonthly_spending: " << monthly_spending;

        file << "\ncredit_amount: " << credit_amount;
        file << "\ninstalment_amount: " << instalment_amount;
        if (pledge)
        {
            file << "\npledge_value: " << pledge_value;
            file << "\npledge_name: " << pledge_name;
        }
    }
    file.close();
}
```

Metoda void saved_based_data;

Zapisuje ona do pliku informacje podstawowe, czyli podane przez klienta (niezmodyfikowane przez program). Kolejne dwie metody zapisują kolejno informacje zmodyfikowane przez program takie jak całkowita suma kredytu (z uwzględnieniem RRSO) oraz informacje specyficzne dla danego kredytu np. biznes plan klienta zakładającego firmę.

Podziękowanie

Dziękuję Państwu za czas poświęcony na przeczytanie powyższego sprawozdania, jeżeli uważają Państwo, że mogłem coś zrobić lepiej, proszę o informację zwrotną na maila: *kp306682@student.polsl.pl* .

Pozdrawiam i życzę wszystkiego dobrego

Karol Pitera