helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

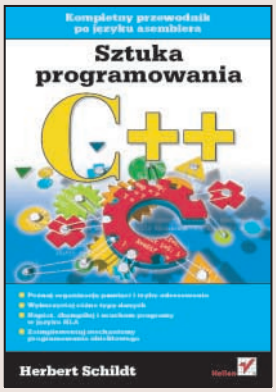
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

C++. Sztuka programowania



Autor: Herbert Schildt
Tłumaczenie: Małgorzata Koziej (rozdz. 2), Marcin Miklas (rozdz. 2, 7, 9), Marcin Samodulski (rozdz. 1, 3 – 6, 8)
ISBN: 83-7361-679-9
Tytuł oryginału: The Art of C++
Format: B5, stron: 384

Poznaj profesjonalne techniki programistyczne

Książka „C++. Sztuka programowania" przedstawia profesjonalne sposoby tworzenia aplikacji w języku C++. Nie opisuje podstaw języka — skupia się na tworzeniu praktycznych aplikacji z wykorzystaniem profesjonalnych technik programistycznych. Wykonując zawarte w książce przykłady, rozwiążesz różne zadania programistyczne i osiągniesz biegłość w posługiwaniu się językiem C++.

- Szablony STL, biblioteki i model obiektowy języka C++
- Mechanizm odzyskiwania pamięci
- Wykorzystanie wielowątkowości w aplikacjach
- Eksperymentalne elementy języka C++
- Aplikacje internetowe w C++
- Techniki sztucznej inteligencji
- Tworzenie własnego interpretera C++

O Autorze:
Herbert Schildt jest jednym z najpoczytniejszych autorów książek poświęconych programowaniu, konkretnie językom C, C++, Java i C#. Był członkiem komitetu ANSI/ISO, który dokonał standaryzacji języka C++. Jego książki poświęcone programowaniu zostały sprzedane w 3 milionach egzemplarzy na całym świecie i przetłumaczone na kilkanaście języków.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

O Autorze **9**

Wstęp **11**

Rozdział 1. Potęga C++ **13**
Zwięzła, lecz bogata składnia 14
Potężne biblioteki 14
Biblioteka szablonów STL 15
Programista ma władzę 16
Precyzyjne sterowanie 17
Przeciążanie operatorów 17
Przejrzysty, zwięzły model obiektowy 18
Dziedzictwo C++ 18

Rozdział 2. Prosty mechanizm odzyskiwania pamięci dla języka C++ **21**
Porównanie dwóch metod zarządzania pamięcią 22
Plusy i minusy ręcznego zarządzania pamięcią 23
Plusy i minusy mechanizmu odzyskiwania pamięci 24
Możesz mieć obie metody 25
Tworzenie mechanizmu odzyskiwania pamięci w języku C++ 25
Zrozumienie problemu 26
Wybór algorytmu odzyskiwania pamięci 26
Zliczanie referencji 27
Znacz i zamiataj 27
Kopiowanie 27
Który algorytm wybrać? 28
Implementowanie mechanizmu odzyskiwania pamięci 28
Wielowątkowy czy nie? 29
Kiedy odzyskiwać pamięć? 29
Co z auto_ptr? 30
Prosty mechanizm odzyskiwania pamięci w C++ 31
Omówienie klas mechanizmu odzyskiwania pamięci 41
GCPtr szczegółowo 42
Zmienne składowe klasy GCPtr 43
Funkcja findPtrInfo() 44
Definicja typu GCiterator 44
Konstruktor klasy GCPtr 44
Destruktor klasy GCPtr 46

Odzyskiwanie pamięci za pomocą funkcji collect() ... 46
Przeciążone operatory przypisania ... 48
Konstruktor kopiujący klasy GCPtr ... 50
Operatory wskaźnikowe i funkcja konwertująca ... 51
Funkcje begin() i end() ... 52
Funkcja shutdown() ... 53
Dwie funkcje pomocnicze ... 53
Klasa GCInfo ... 54
Klasa Iter ... 55
Jak używać klasy GCPtr? ... 57
Obsługa wyjątków alokacji pamięci ... 58
Bardziej interesujący przykład ... 59
Alokowanie i porzucanie obiektów ... 61
Alokowanie tablic ... 63
Użycie GCPtr z typami klasowymi ... 65
Większy program demonstracyjny ... 67
Testowanie obciążenia ... 72
Kilka ograniczeń ... 74
Pomysły do wypróbowania ... 75

Rozdział 3. Wielowątkowość w C++ ... 77
Czym jest wielowątkowość? ... 78
Wielowątkowość zmienia architekturę programu ... 78
Dlaczego język C++ nie ma wbudowanej obsługi mechanizmu wielowątkowości? ... 79
Jaki system operacyjny i kompilator wybrać? ... 80
Przegląd funkcji obsługi wątków w Windows ... 81
Tworzenie i zatrzymywanie wątku ... 81
Inne funkcje obsługi wątków dostępne w Visual C++ ... 82
Wstrzymywanie i wznawianie wątku ... 84
Zmiana priorytetu wątku ... 84
Pobieranie uchwytu głównego wątku ... 86
Zagadnienie synchronizacji ... 86
Tworzenie panelu kontrolnego wątków ... 90
Panel kontrolny wątku ... 91
Panel kontrolny wątku pod lupą ... 95
Prezentacja działania panelu kontrolnego ... 101
Wielowątkowy mechanizm odzyskiwania pamięci ... 106
Dodatkowe zmienne składowe ... 107
Konstruktor wielowątkowej klasy GCPtr ... 107
Wyjątek TimeOutExc ... 109
Wielowątkowy destruktor klasy GCPtr ... 110
Funkcja gc() ... 110
Funkcja isRunning() ... 111
Synchronizacja dostępu do listy gclist ... 111
Dwie inne zmiany ... 112
Kompletny kod wielowątkowego mechanizmu odzyskiwania pamięci ... 112
Użycie wielowątkowego mechanizmu odzyskiwania pamięci ... 123
Pomysły do wypróbowania ... 128

Rozdział 4. Rozszerzanie języka C++ ... 129
Dlaczego należy użyć translatora? ... 129
Eksperymentalne słowa kluczowe ... 131
Pętla foreach ... 131
Wyrażenie cases ... 132

Operator typeof ... 133
Pętla repeat-until ... 134
Translator eksperymentalnych elementów języka C++ ... 135
Użycie translatora ... 143
Jak działa translator? ... 144
Deklaracje globalne ... 144
Funkcja main() ... 145
Funkcje gettoken() i skipspaces() ... 146
Przekształcanie pętli foreach ... 149
Przekształcanie wyrażenia cases ... 152
Przekształcanie operatora typeof ... 154
Przekształcanie pętli repeat-until ... 155
Program demonstracyjny ... 157
Pomysły do wypróbowania ... 163

Rozdział 5. Program do ściągania plików z internetu ... 165
Biblioteka WinINet ... 166
Podsystem programu do ściągania pliku z internetu ... 167
Ogólny opis działania ... 171
Funkcja download() ... 172
Funkcja ishttp() ... 177
Funkcja httpverOK() ... 178
Funkcja getfname() ... 179
Funkcja openfile() ... 179
Funkcja update() ... 180
Plik nagłówkowy podsystemu do ściągania plików ... 181
Program prezentujący działanie podsystemu do ściągania plików ... 181
Program do ściągania plików z graficznym interfejsem użytkownika ... 183
Kod programu WinDL ... 183
Jak działa program WinDL? ... 188
Pomysły do wypróbowania ... 190

Rozdział 6. Obliczenia finansowe w C++ ... 191
Plan spłaty pożyczki ... 192
Obliczanie przyszłej wartości lokaty ... 193
Obliczanie początkowej wartości inwestycji wymaganej do otrzymania pożądanej przyszłej wartości ... 195
Obliczanie początkowej wartości inwestycji wymaganej do otrzymania określonych wypłat ... 196
Obliczanie maksymalnej wielkości regularnej wypłaty z danej lokaty ... 198
Obliczanie wartości pozostałego kredytu ... 200
Pomysły do wypróbowania ... 201

Rozdział 7. Rozwiązywanie problemów metodami sztucznej inteligencji ... 203
Reprezentacja i terminologia ... 204
Eksplozje kombinatoryczne ... 205
Strategie przeszukiwania ... 207
Ocenianie strategii przeszukiwania ... 207
Problem ... 208
Reprezentacja graficzna ... 208
Struktura FlightInfo i klasa Search ... 209
Przeszukiwanie w głąb ... 211
Funkcja match() ... 216
Funkcja find() ... 217
Funkcja findroute() ... 218

Wyświetlanie trasy ... 219
Analiza przeszukiwania w głąb ... 220
Przeszukiwanie wszerz ... 220
Analiza przeszukiwania wszerz ... 223
Dodawanie heurystyk ... 223
Przeszukiwania wspinaczkowe ... 224
Analiza przeszukiwania wspinaczkowego ... 228
Przeszukiwanie najmniejszego kosztu ... 229
Analiza przeszukiwania najmniejszego kosztu ... 231
Znajdowanie wielu rozwiązań ... 231
Usuwanie ścieżki ... 231
Usuwanie wierzchołków ... 232
Szukanie optymalnego rozwiązania ... 238
Wracamy do zgubionych kluczy ... 244
Pomysły do wypróbowania ... 247

Rozdział 8. Tworzenie własnego kontenera STL ... 249
Krótkie omówienie biblioteki STL ... 250
Kontenery ... 250
Algorytmy ... 251
Iteratory ... 251
Inne składniki biblioteki STL ... 251
Wymagania stawiane definiowanemu kontenerowi ... 252
Wymagania ogólne ... 252
Dodatkowe wymagania dla kontenerów sekwencyjnych ... 254
Dodatkowe wymagania dla kontenerów asocjacyjnych ... 254
Tworzenie kontenera tablicy dynamicznej ze zmiennymi indeksami ... 255
Jak działa tablica RangeArray? ... 255
Kompletna klasa RangeArray ... 257
Klasa RangeArray pod lupą ... 266
Kilka programów wykorzystujących tablicę RangeArray ... 279
Pomysły do wypróbowania ... 288

Rozdział 9. Miniinterpreter języka C++ ... 289
Interpretery kontra kompilatory ... 289
Opis ogólny interpretera Mini C++ ... 290
Charakterystyka interpretera Mini C++ ... 291
Ograniczenia interpretera Mini C++ ... 293
Nieformalna teoria języka C++ ... 294
Wyrażenia języka C++ ... 295
Definiowanie wyrażenia ... 295
Parser wyrażeń ... 297
Kod źródłowy parsera ... 297
Wyodrębnianie elementów leksykalnych z kodu źródłowego ... 309
Wyświetlanie błędów składniowych ... 314
Obliczanie wyrażenia ... 315
Interpreter Mini C++ ... 317
Funkcja main() ... 334
Wstępny przegląd kodu ... 335
Funkcja interp() ... 339
Obsługa zmiennych lokalnych ... 341
Wywoływanie funkcji zdefiniowanych przez użytkownika ... 343
Przypisywanie wartości zmiennym ... 345
Wykonywanie instrukcji if ... 347

Instrukcje switch i break ... 348
Przetwarzanie pętli while ... 350
Przetwarzanie pętli do-while ... 351
Pętla for ... 353
Obsługa instrukcji cin i cout ... 354
Biblioteka funkcji interpretera Mini C++ ... 356
Plik nagłówkowy mccommon.h ... 358
Kompilacja i konsolidacja interpretera Mini C++ ... 360
Demonstrowanie działania interpretera Mini C++ ... 360
Udoskonalanie interpretera Mini C++ ... 368
Rozszerzanie interpretera Mini C++ ... 369
Dodawanie nowych elementów języka C++ ... 369
Dodawanie elementów pomocniczych ... 370

Skorowidz ... 371

Rozdział 6.

Obliczenia finansowe w C++

Poza wszystkimi dużymi i skomplikowanymi aplikacjami, takimi jak: kompilatory, przeglądarki internetowe, edytory tekstów, bazy danych i programy księgowe, które zdominowały świat komputerów, istnieje pewna grupa programów będących zarówno pożytecznymi, jak i nieskomplikowanymi. Programy te wykonują różne obliczenia finansowe, na przykład wyznaczają wysokość raty kredytu, przyszłą wartość lokaty, pozostałą wartość kredytu. Żadne z wymienionych obliczeń nie są bardzo skomplikowane ani nie wymagają obszernego kodu, lecz uzyskujemy dzięki nim całkiem użyteczne informacje.

Ponieważ język C++ najlepiej nadaje się do tworzenia potężnych aplikacji systemowych, często nie docenia się jego możliwości w dziedzinie obliczeń finansowych. Jest to oczywiście błąd. Język C++ charakteryzuje się doskonałymi właściwościami w tym zakresie, gdyż oferuje szeroki zakres funkcji matematycznych i wydajny mechanizm arytmetycznych operacji zmiennoprzecinkowych. Ponadto, ze względu na fakt, że kod wykonywalny generowany na podstawie kodu C++ jest ekstremalnie wydajny, język C++ stanowi doskonały wybór dla programów wykonujących złożone analizy finansowe i modelowanie.

Aby pokazać, z jaką łatwością można wykonywać obliczenia finansowe posługując się językiem C++, w tym rozdziale stworzonych zostanie kilka małych programów, które obliczają:

1. Wysokość raty;
2. Przyszłą wartość lokaty;
3. Początkową wartość inwestycji wymaganą do otrzymania pożądanej przyszłej wartości;
4. Początkową wartość inwestycji wymaganą do otrzymania określonych wypłat;
5. Maksymalną wielkość regularnej wypłaty z danej lokaty;
6. Pozostałą kwotę kredytu.

Programy te mogą być wykorzystywane w niezmienionej postaci lub można je przystosować do konkretnych potrzeb. Pomimo tego, że są to najprostsze programy w tej książce, może się okazać, że będziesz ich używał najczęściej.

Plan spłaty pożyczki

Prawdopodobnie najczęściej wykonywanym obliczeniem finansowym jest wyznaczenie rat kredytu, przeznaczonego na przykład na samochód lub dom. Raty kredytu oblicza się za pomocą następującego wzoru:

$$\text{Rata} = (\text{oproc} * (\text{kwota} / \text{liczbaRatRocznie})) / (1 - ((\text{oproc} / \text{liczbaRatRocznie}) + 1)^{-\text{liczbaRatRocznie} * \text{liczbaLat}})$$

gdzie *oproc* określa oprocentowanie kredytu, *kwota* — wysokość kredytu, *liczbaRatRocznie* oznacza liczbę rat przypadającą na jeden rok, a *liczbaLat* określa długość spłaty kredytu w latach.

W poniższym programie w celu obliczenia wysokości rat przedstawiony wzór został użyty w ciele funkcji `regpay()`. Do funkcji tej przekazujemy oprocentowanie, wielkość pierwszej raty, długość spłaty kredytu w latach oraz liczbę rat przypadających na jeden rok. Funkcja zwraca wysokość raty.

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>

using namespace std;

// Obliczanie jednakowych rat kredytu.
double regpay(double principal, double intRate,
              int numYears, int payPerYear) {
  double numer;
  double denom;
  double b, e;

  intRate /= 100.0; // przekształć procenty na ułamki

  numer = intRate * principal / payPerYear;

  e = -(payPerYear * numYears);
  b = (intRate / payPerYear) + 1.0;

  denom = 1.0 - pow(b, e);

  return numer / denom;
}

int main() {
  double p, r;
  int y, ppy;
```

```
    cout.imbue(locale("polish"));
    cin.imbue(locale("polish"));

    cout << "Wprowadź kwotę kredytu: ";
    cin >> p;

    cout << "Wprowadź wysokość oprocentowania (wyrażoną w procentach): ";
    cin >> r;

    cout << "Wprowadź liczbę lat: ";
    cin >> y;

    cout << "Wprowadź liczbę spłat w roku: ";
    cin >> ppy;

    cout << "\nRata: " << fixed << setprecision(2)
         << regpay(p, r, y, ppy) << endl;

    return 0;
}
```

Aby obliczyć wysokość raty kredytu, wprowadź wymagane dane. Oto przykładowy przebieg programu:

```
Wprowadź wysokość kredytu: 1000
Wprowadź wysokość oprocentowania (wyrażoną w procentach): 9
Wprowadź liczbę lat: 5
Wprowadź liczbę spłat w roku: 12

Rata: 20.76
```

Zwróć uwagę na kilka elementów w ciele funkcji `main()`. Po pierwsze, obiekt `locale` związany ze strumieniem `cout` został ustawiony na *Polish*. Wykonano to przez wywołanie metody `imbue()` i przekazanie obiektu `locale` dla Polski. Dzięki temu zapewnione jest prawidłowe wyświetlanie sum pieniężnych zgodnie z zasadami obowiązującymi w języku polskim, przez co rozumiemy oddzielanie poszczególnych członów tysięcznych za pomocą znaku spacji oraz stosowanie przecinka do oddzielania części całkowitej liczby od jej części ułamkowej. Po drugie, przed wyświetleniem obliczonej wysokości raty, format liczby został zmieniony na `fixed` (stały) z precyzją 2 miejsc po przecinku z odpowiednim zaokrąglaniem. Jeśli to konieczne, do części ułamkowej dodawane są zera. We wszystkich prezentowanych w tym rozdziale programach do obliczeń finansowych zastosowane jest to samo podejście. Aby zmienić format liczb w celu dopasowania go do innego języka lub regionu, wystarczy zmienić język lub region obiektu `locale` przekazywanego do funkcji `imbue()`.

Obliczanie przyszłej wartości lokaty

Inną często obliczaną wartością jest przyszła wartość lokaty na podstawie początkowej wielkości lokaty, oprocentowania, rocznej częstotliwości kapitalizacji odsetek oraz długości lokaty wyrażonej w latach. Na przykład możesz chcieć się dowiedzieć, ile będziesz miał na koncie za 12 lat, jeżeli obecny stan Twego konta to 98 000 zł,

a średnie oprocentowanie roczne wynosi 6 procent. Przedstawiony tutaj program odpowie na to pytanie.

Aby obliczyć przyszłą wartość inwestycji, użyj następującego wzoru:

$$\text{Przyszła wartość} = \text{lokata} * ((\text{oproc} / \text{rocznaCzęstKapit}) + 1)^{\text{rocznaCzęstKapit} * \text{liczbaLat}}$$

gdzie *oproc* określa oprocentowanie lokaty, *lokata* to początkowa wysokość wkładu, *rocznaCzęstKapit* oznacza częstotliwość kapitalizacji odsetek w roku, a *liczbaLat* to długość lokaty wyrażona w latach. W przypadku naliczania odsetek raz do roku, częstotliwość kapitalizacji odsetek w roku wynosi oczywiście *1*.

W poniższym programie w celu obliczenia przyszłej wartości inwestycji przedstawiony wzór został użyty w ciele funkcji `futval()`. Do funkcji tej przekazujemy oprocentowanie lokaty, długość lokaty w latach oraz roczną częstotliwość kapitalizacji odsetek. Funkcja zwraca przyszłą wartość lokaty.

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>

using namespace std;

// Obliczanie przyszłej wartości lokaty.
double futval(double principal, double rateOfRet,
              int numYears, int compPerYear) {
  double b, e;

  rateOfRet /= 100.0; // przekształć procenty na ułamki

  b = (1 + rateOfRet/compPerYear);
  e = compPerYear * numYears;

  return principal * pow(b, e);
}

int main() {
  double p, r;
  int y, cpy;

  cout.imbue(locale("polish"));

  cout << "Wprowadź wielkość lokaty: ";
  cin >> p;

  cout << "Wprowadź wysokość oprocentowania lokaty (wyrażoną w procentach): ";
  cin >> r;

  cout << "Wprowadź liczbę lat: ";
  cin >> y;

  cout << "Wprowadź roczną częstotliwość kapitalizacji odsetek: ";
  cin >> cpy;

  cout << "\nPrzyszła wartość: " << fixed << setprecision(2)
       << futval(p, r, y, cpy) << endl;
```

```
  return 0;
}
```

Oto przykładowy przebieg programu:

```
Wprowadź wielkość lokaty: 10000
Wprowadź wysokość oprocentowania lokaty (wyrażoną w procentach): 6
Wprowadź liczbę lat: 5
Wprowadź roczną częstotliwość kapitalizacji odsetek: 12

Przyszła wartość: 13 488,50
```

Obliczanie początkowej wartości inwestycji wymaganej do otrzymania pożądanej przyszłej wartości

Czasami chcemy wiedzieć, ile musimy zainwestować, aby otrzymać w przyszłości jakąś określoną wartość. Na przykład jeżeli zbierasz pieniądze na prywatną szkołę dla dziecka i wiesz, że za 5 lat będziesz potrzebował 75 000 zł, zadasz pytanie: ile pieniędzy muszę przeznaczyć na lokatę oprocentowaną na 7 procent w skali roku, aby uzyskać założoną kwotę. Wykorzystując program zaprezentowany w tym podrozdziale, możesz znaleźć odpowiedź na to pytanie.

Wzór, na podstawie którego można obliczyć początkową wartość lokaty, wygląda następująco:

Początkowa wartość =
= wartośćDocelowa / (((oproc / rocznaCzęstKapit) + 1) $^{\text{rocznaCzęstKapit} * \text{liczbaLat}}$)

gdzie *oproc* określa oprocentowanie lokaty, *wartośćDocelowa* to założona przyszła wartość, którą chcemy uzyskać, *rocznaCzęstKapit* oznacza częstotliwość kapitalizacji odsetek w roku, a *liczbaLat* to długość lokaty wyrażona w latach. Jeżeli odsetki naliczane są raz do roku, częstotliwość kapitalizacji odsetek w roku wynosi oczywiście *1*.

W poniższym programie w celu obliczenia początkowej wysokości lokaty przedstawiony wzór został użyty w ciele funkcji `initval()`. Do funkcji tej przekazujemy założoną wartość docelową, oprocentowanie lokaty, długość lokaty w latach oraz roczną częstotliwość kapitalizacji odsetek. Funkcja zwraca początkową wartość lokaty.

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>

using namespace std;

// Obliczanie początkowej wartości lokaty
// wymaganej do uzyskania założonej przyszłej wartości.
double initval(double targetValue, double rateOfRet,
               int numYears, int compPerYear) {
```

```
  double b, e;

  rateOfRet /= 100.0; // przekształć procenty na ułamki

  b = (1 + rateOfRet/compPerYear);
  e = compPerYear * numYears;

  return targetValue / pow(b, e);
}

int main() {
  double p, r;
  int y, cpy;

cout.imbue(locale("polish"));

  cout << "Wprowadź pożądaną wartość: ";
  cin >> p;

  cout << "Wprowadź wysokość oprocentowania lokaty (wyrażoną w procentach): ";
  cin >> r;

  cout << "Wprowadź liczbę lat: ";
  cin >> y;

  cout << " Wprowadź roczną częstotliwość kapitalizacji odsetek: ";
  cin >> cpy;

  cout << "\nWymagana początkowa wartość lokaty: "
       << fixed << setprecision(2)
       << initval(p, r, y, cpy) << endl;

  return 0;
}
```

Oto przykładowy przebieg programu:

```
Wprowadź pożądaną wartość: 75000
Wprowadź wysokość oprocentowania lokaty (wyrażoną w procentach): 7
Wprowadź liczbę lat: 5
Wprowadź roczną częstotliwość kapitalizacji odsetek: 4

Wymagana początkowa wartość lokaty: 53 011,84
```

Obliczanie początkowej wartości inwestycji wymaganej do otrzymania określonych wypłat

Kolejną często obliczaną wielkością jest początkowa wartość lokaty, która zapewnia regularne wypłaty określonej sumy. Na przykład możesz zaplanować, że będziesz potrzebował w przyszłości 5 000 zł miesięcznie w charakterze emerytury wypłacanej

przez 20 lat. Pytanie, które powinieneś sobie zadać, brzmi: jak dużo muszę zgromadzić, aby zapewnić sobie taką emeryturę. Odpowiedź znajdziesz, wykorzystując następujący wzór:

$$\text{Początkowa wartość} = ((\text{wysWyp} * \text{liczbaWypRocznie}) / \text{oproc}) * (1 - (1 / (\text{oproc} / \text{liczbaWypRocznie} + 1)^{\text{liczbaWypRocznie} * \text{liczbaLat}}))$$

gdzie *oproc* określa wysokość oprocentowania lokaty, *wysWyp* to pożądana wysokość regularnych wypłat, *liczbaWypRocznie* oznacza zakładaną liczbę wypłat na rok, a *liczbaLat* to okres wypłacania wyrażony w latach.

W poniższym programie w celu obliczenia początkowej wysokości lokaty przedstawiony wzór został użyty w ciele funkcji `annuity()`. Do funkcji tej przekazujemy założoną wartość docelową regularnej wypłaty, oprocentowanie lokaty, okres wypłacania wyrażony w latach oraz roczną liczbę wypłat. Funkcja zwraca początkową wymaganą wartość lokaty.

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>

using namespace std;

// Obliczanie początkowej wartości inwestycji
// wymaganej do otrzymania określonych regularnych wypłat.
double annuity(double regWD, double rateOfRet,
               int numYears, int numPerYear) {

  double b, e;
  double t1, t2;

  rateOfRet /= 100.0; // przekształć procenty na ułamki

  t1 = (regWD * numPerYear) / rateOfRet;

  b = (1 + rateOfRet/numPerYear);
  e = numPerYear * numYears;

  t2 = 1 - (1 / pow(b, e));

  return t1 * t2;
}

int main() {
  double wd, r;
  int y, wpy;

cout.imbue(locale("polish"));

  cout << "Wprowadź pożądaną kwotę wypłaty: ";
  cin >> wd;

  cout << "Wprowadź wysokość oprocentowania lokaty (wyrażoną w procentach): ";
  cin >> r;
```

```
    cout << "Wprowadź liczbę lat: ";
    cin >> y;

    cout << "Wprowadź liczbę wypłat w roku: ";
    cin >> wpy;

    cout << "\nWymagana początkowa wysokość lokaty: "
         << fixed << setprecision(2)
         << annuity(wd, r, y, wpy) << endl;

    return 0;
}
```

Oto przykładowy przebieg programu:

```
Wprowadź pożądaną kwotę wypłaty: 5000
Wprowadź wysokość oprocentowania lokaty (wyrażoną w procentach): 6
Wprowadź liczbę lat: 20
Wprowadź liczbę wypłat w roku: 12

Wymagana początkowa wysokość lokaty: 697 903,86
```

Obliczanie maksymalnej wielkości regularnej wypłaty z danej lokaty

Innym rodzajem obliczeń finansowych jest wyznaczanie maksymalnej kwoty wypłaty (w sensie regularnych wypłat), jaką można otrzymywać przez określony czas, na podstawie danej lokaty. Załóżmy, że masz 500 000 zł na koncie i chciałbyś wiedzieć, ile możesz podejmować każdego miesiąca przez 20 lat, zakładając, że oprocentowanie lokaty wynosi 6 procent. Wzór do obliczania poszukiwanej wartości jest następujący:

$$\text{Maksymalna wysokość wypłaty} = \text{lokata} * (((\text{oproc} / \text{liczbaWypRocznie}) / (-1 + ((\text{oproc} / \text{liczbaWypRocznie}) + 1)^{\text{liczbaWypRocznie} * \text{liczbaLat}})) + (\text{oproc} / \text{liczbaWypRocznie}))$$

gdzie *oproc* określa wysokość oprocentowania lokaty, *liczbaWypRocznie* to zakładana liczba wypłat na rok, *liczbaLat* oznacza okres wypłacania wyrażony w latach, a *lokata* wskazuje początkową wysokość lokaty.

W poniższym programie w celu obliczenia maksymalnej wysokości wypłaty przedstawiony wzór został użyty w ciele funkcji `maxwd()`. Do funkcji tej przekazujemy początkową wartość lokaty, oprocentowanie lokaty, okres wypłacania wyrażony w latach oraz roczną liczbę wypłat. Funkcja zwraca maksymalną wartość wypłaty.

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>

using namespace std;
```

```
// Obliczanie maksymalnej wielkości regularnej wypłaty
// podejmowanej przez określony czas z lokaty o podanej wartości początkowej.
double maxwd(double principal, double rateOfRet,
             int numYears, int numPerYear) {

  double b, e;
  double t1, t2;

  rateOfRet /= 100.0; // przekształć procenty na ułamki

  t1 = rateOfRet / numPerYear;

  b = (1 + t1);
  e = numPerYear * numYears;

  t2 = pow(b, e) - 1;

  return principal * (t1/t2 + t1);
}

int main() {
  double p, r;
  int y, wpy;

cout.imbue(locale("polish"));

  cout << "Wprowadź początkową wartość lokaty: ";
  cin >> p;

  cout << "Wprowadź wysokość oprocentowania lokaty (wyrażoną w procentach): ";
  cin >> r;

  cout << "Wprowadź liczbę lat: ";
  cin >> y;

  cout << "Wprowadź liczbę wypłat w roku: ";
  cin >> wpy;

  cout << "\nMaksymalna kwota wypłaty: " << fixed << setprecision(2)
       << maxwd(p, r, y, wpy) << endl;

  return 0;
}
```

Oto przykładowy przebieg programu:

```
Wprowadź początkową wartość lokaty: 500000
Wprowadź wysokość oprocentowania lokaty (wyrażoną w procentach): 6
Wprowadź liczbę lat: 20
Wprowadź liczbę wypłat w roku: 12

Maksymalna kwota wypłaty: 3 582,16
```

Obliczanie wartości pozostałego kredytu

Często potrzebna jest informacja dotycząca tego, ile kredytu zostało do spłacenia. Można to łatwo obliczyć, znając początkową kwotę kredytu, jego oprocentowanie, wysokość raty oraz liczbę dokonanych wpłat. Aby poznać pozostałą kwotę kredytu, trzeba zsumować dokonane wpłaty, odejmując od każdej wpłaty odsetki, a następnie odjąć tak otrzymaną liczbę od początkowej kwoty kredytu.

Przedstawiona tutaj funkcja `balance()` oblicza pozostałą do spłacenia kwotę kredytu. Do funkcji tej przekazujemy początkową wartość kredytu, oprocentowanie lokaty, wielkość raty, liczbę spłat w roku i liczbę dokonanych wpłat. Funkcja zwraca pozostałą kwotę kredytu.

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <locale>

using namespace std;

// Oblicza pozostałą kwotę kredytu.
double balance(double principal, double intRate,
               double payment, int payPerYear,
               int numPayments) {

  double bal = principal;
  double rate = intRate / payPerYear;

  rate /= 100.0; // przekształć procenty na ułamki

  for(int i = 0; i < numPayments; i++)
    bal -= payment - (bal * rate);

  return bal;
}

int main() {
  double p, r, pmt;
  int ppy, npmt;

cout.imbue(locale("polish"));

  cout << "Wprowadź początkową kwotę kredytu: ";
  cin >> p;

  cout << "Wprowadź wysokość oprocentowania lokaty (wyrażoną w procentach): ";
  cin >> r;

  cout << "Wprowadź wielkość raty: ";
  cin >> pmt;
```

```
    cout << "Wprowadź liczbę spłat w roku: ";
    cin >> ppy;

    cout << "Wprowadź liczbę dokonanych wpłat: ";
    cin >> npmt;

    cout << "Pozostała kwota kredytu: " << fixed << setprecision(2)
         << balance(p, r, pmt, ppy, npmt) << endl;

    return 0;
}
```

Oto przykładowy przebieg programu:

```
Wprowadź początkową kwotę kredytu: 10000
Wprowadź wysokość oprocentowania lokaty (wyrażoną w procentach): 9
Wprowadź wielkość raty: 207.58
Wprowadź liczbę spłat w roku: 12
Wprowadź liczbę dokonanych wpłat: 30

Pozostała kwota kredytu: 5 558,19
```

Pomysły do wypróbowania

Istnieje wiele innych obliczeń finansowych, które mogą Ci się przydać, jak choćby obliczanie stopy oprocentowania lokaty lub wysokość regularnych wpłat prowadzących do uzyskania określonej przyszłej wartości. Możesz także wygenerować plan amortyzacji kredytu.

Być może zechcesz stworzyć większą aplikację obejmującą wszystkie przedstawione w tym rozdziale obliczenia. Użytkownik mógłby wybierać odpowiednią opcję z menu.