helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

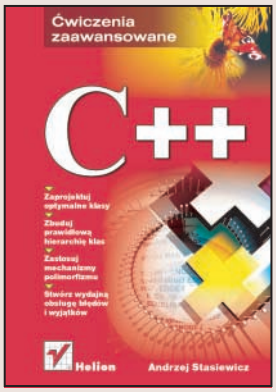
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

C++. Ćwiczenia zaawansowane



Autor: Andrzej Stasiewicz
ISBN: 83-7361-766-3
Format: B5, stron: 120

„C++. Ćwiczenia zaawansowane" to kontynuacja książki „C++. Ćwiczenia praktyczne" przeznaczona dla tych, którzy chcą pogłębiać swoją wiedzę o najpopularniejszym obecnie języku programowania. Przedstawia kolejne, niezwykle istotne zagadnienia związane z programowaniem w C++, mianowicie zasady programowania obiektowego. Przeczytasz o klasach, metodach i szablonach oraz poznasz w praktyce metody stosowania tych mechanizmów we własnych programach. Nauczysz się technik stosowania gotowych struktur danych, takich jak dynamiczna tablica, kolejka, lista oraz stos.

- Instalacja środowiska programistycznego
- Operacje zapisu i odczytu plików
- Klasy – definiowanie i stosowanie
- Konstruktorzy i destruktor klas
- Hierarchia klas i dziedziczenie
- Obsługa wyjątków
- Stosowanie szablonów

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

Spis treści

Wprowadzenie 5

Rozdział 1. Piszesz pierwsze programy 9
Instalacja środowiska programistycznego 9
Funkcja main() i ogólny szkic programu 11
Słowo kluczowe namespace, czyli przestrzenie nazw 14
Podsumowanie 16

Rozdział 2. Operacje na plikach 17
Otwieranie plików do zapisu i odczytu 17
Modyfikowanie standardowych operacji wejścia i wyjścia 20
Podsumowanie 23

Rozdział 3. Klasy, czyli Twoje własne typy danych 25
Deklarowanie i definiowanie klasy 25
Funkcje zaprzyjaźnione z klasami 28
Statyczne składniki klasy 30
Funkcje statyczne klasy 32
Pola bitowe w klasach 33
Słowo kluczowe this 34
Podsumowanie 35

Rozdział 4. Konstruktory i destruktor 37
Konstruktor bezargumentowy, zwany domyślnym 37
Konstruktory merytoryczne 38
Lista inicjalizacyjna konstruktora 39
Konstruktor kopiujący 40
Konstruktory konwertujące 42
Konstruktor a składniki stałe klasy 42
Destruktor 43
Gdy w klasie brakuje konstruktora… 44
Podsumowanie 46

Rozdział 5. Przeciążanie operatorów 47
Operatory zadeklarowane w klasie 47
Operator przypisania = ... 50
Operatory zadeklarowane poza klasą 52
Operator wyprowadzania danych << 53
Podsumowanie 54

Rozdział 6. Konwersje, czyli przekształcanie danych jednych typów w inne ... 55
Operator konwersji ... 55
Niejawne konwersje za pomocą operatora konwersji ... 57
Jednoargumentowe konstruktory konwersji ... 58
Niejawne konwersje za pomocą konstruktora ... 58
Konstruktor jednoargumentowy explicit ... 59
Operator static_cast ... 60
Operator const_cast ... 61
Operator dynamic_cast ... 62
Operator reinterpret_cast ... 63
Podsumowanie ... 63

Rozdział 7. Dziedziczenie, czyli hierarchie klas ... 65
Dziedziczenie po klasie a zawieranie klasy ... 65
Rodzaje dziedziczenia ... 67
Konstruktory klasy pochodnej ... 70
Konstruktor kopiujący klasy pochodnej ... 72
Operator przypisania w klasie pochodnej ... 73
Destruktor w klasie pochodnej ... 74
Funkcje wirtualne ... 75
Funkcje czysto wirtualne i klasy abstrakcyjne ... 76
Polimorfizm ... 77
Destruktor wirtualny ... 79
Podsumowanie ... 80

Rozdział 8. Obsługa sytuacji wyjątkowych ... 83
Ogólny schemat obsługi wyjątków ... 83
Zgłaszanie i identyfikacja wyjątków ... 84
Klasy wyjątków ... 86
Wyjątki i nowy styl pisania nagłówków funkcji ... 88
Hierarchia klas wyjątków ... 90
Wyjątki generowane przez funkcje biblioteczne ... 91
Podsumowanie ... 93

Rozdział 9. Szablony funkcji i klas ... 95
Szablony funkcji ... 95
Szablony klas ... 97
Domyślne typy — parametry szablonów ... 99
Podsumowanie ... 100

Rozdział 10. Kontener vector ... 101
Rozmiary wektorów ... 101
Iteratory ... 104
Wyszukiwanie elementu w wektorze ... 107
Sortowanie elementów wektora ... 108
Tablice wielowymiarowe ... 109
Tablice klas ... 110
Zewnętrzne funkcje relacji (predykaty) ... 114
Podsumowanie ... 116

Zakończenie ... 117

Rozdział 2.

Operacje na plikach

Otwieranie plików do zapisu i odczytu

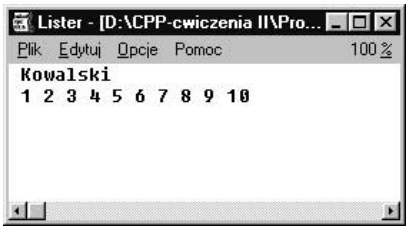
Ćwiczenie 2.1.

Otwórz plik dyskowy o nazwie test.txt. W pierwszej linii zapisz w nim swoje nazwisko, w drugiej 10 kolejnych liczb naturalnych, rozdzielonych spacjami (rysunki 2.1a i 2.1b):

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
 string tekst;
 cout << "Podaj nazwisko: ";
 cin >> tekst;
 ofstream plik( "test.txt");
 if( plik)
 {
   cout << "Otwarto plik test.txt ..." << endl;
   plik << tekst << endl;
   cout << "Wpisano do niego nazwisko ..." << endl;
   for( int i = 1; i <= 10; ++i)
   {
      cout << "Wpisano liczbe " << i << endl;
      plik << i << " ";
   }
 }
}
```



Rysunek 2.1a. *Efekt działania programu, wpisującego dane do pliku*



Rysunek 2.1b. *Zawartość pliku test.txt, utworzonego w programie*

Standardowy plik do zapisu jest reprezentowany przez obiekt typu `ofstream`. Otwarcie pliku następuje w chwili deklaracji obiektu `plik` za pomocą konstruktora, którego argumentem jest dyskowa nazwa pliku. Zapis w pliku realizuje standardowy operator strumieniowego wyjścia <<.

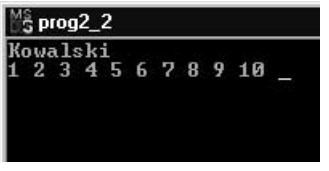
Ćwiczenie 2.2.

Napisz program, który odczyta dane z pliku utworzonego w poprzednim ćwiczeniu (rysunek 2.2):

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main()
{
 string tekst;
 int i, a;
 ifstream plik( "test.txt");
 if( plik)
 {
   plik >> tekst;
   cout << tekst << endl;
   for( i = 0; i < 10; ++i)
   {
      plik >> a;
      cout << a << " ";
   }
 }
}
```

Rysunek 2.2.
Efekt działania programu odczytującego dane z pliku tekstowego



Odczytywany plik jest reprezentowany przez obiekt typu `ifstream`. Standardowy odczyt danych jest wykonywany za pomocą operatora `>>`. Musisz zapamiętać, że operator ten pomija białe znaki *przed* znakami, składającymi się na daną (porównaj ćwiczenia 2.4 i 2.5).

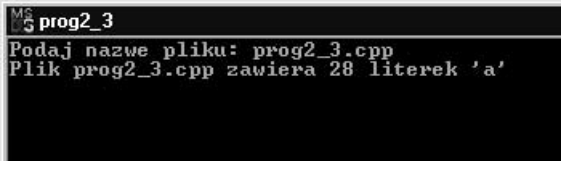
Ćwiczenie 2.3.

Napisz program, który odczyta wskazany plik tekstowy i zliczy znajdujące się w nim znaki „a" (rysunek 2.3).

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
 string nazwa;
 cout << "Podaj nazwe pliku: ";
 cin >> nazwa;
 int suma = 0;
 char znak;

 ifstream plik( nazwa.c_str());
 while( plik >> znak)
 {
   if( znak == 'a')
      ++suma;
 }
 cout << "Plik " << nazwa << " zawiera " << suma << " literek 'a'";
}
```

Rysunek 2.3.
Wynik zliczenia liter „a" w tekście źródłowym niniejszego programu



W konstruktorze strumienia plikowego:

```
ifstream plik( nazwa.c_str());
```

wywołujesz funkcję `c_str()`, która podaje klasyczny wskaźnik typu `char *` do tekstu zgromadzonego w obiekcie `nazwa`, mającego typ `string`.

Modyfikowanie standardowych operacji wejścia i wyjścia

Ćwiczenie 2.4.

Napisz program, który odczyta wskazany plik tekstowy i jego treść wyświetli na ekranie. Problemem jest właściwość pomijania przez standardowy operator wejścia odstępów przed istotnymi znakami (rysunek 2.4):

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
 string nazwa;
 cout << "Podaj nazwe pliku do wyswietlenia: ";
 cin >> nazwa;
 ifstream plik( nazwa.c_str());
 char znak;
 if( plik)
 {
   while( plik >> znak)
      cout << znak;
 }
}
```

Rysunek 2.4. *Niewłaściwy rezultat działania programu*



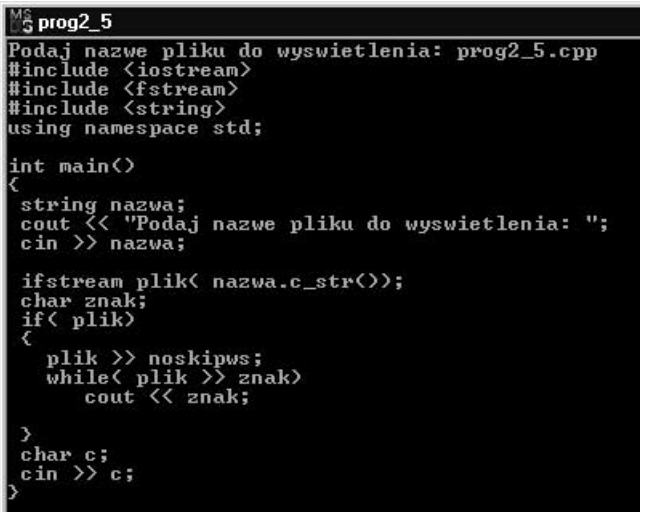
Program działa niewłaściwie, bo operator pobierania danych >> pomija białe znaki. Właściwość ta bardzo przydaje się przy wczytywaniu danych z klawiatury, ale tutaj nie jest potrzebna.

Ćwiczenie 2.5.

W poprzednim programie zmodyfikuj sposób współpracy operatora >> z obiektem plik, by nie pomijał białych znaków (rysunek 2.5):

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
 string nazwa;
 cout << "Podaj nazwe pliku do wyswietlenia: ";
 cin >> nazwa;
```

Rysunek 2.5.
Efekt działania programu modyfikującego pracę operatora >>




```
ifstream plik( nazwa.c_str());
char znak;
if( plik)
{
  plik >> noskipws;
  while( plik >> znak)
     cout << znak;
}
}
```

Ćwiczenie 2.6.

Niech program oczekuje dwóch parametrów, określanych w linii poleceń (porównaj ćwiczenie 1.3) — nazw plików wejściowego i wyjściowego. Treść pliku wejściowego niech zostanie zaszyfrowana i zapisana w pliku wyjściowym (rysunek 2.6):

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main( int argc, char *argv[])
{
 if( argc != 3)
 {
    cout << "W linii polecen podaj 2 pliki";
}
 else
 {
    char znak;
    ifstream in( argv[ 1]);
    ofstream out( argv[ 2]);
    if( in && out)
    {
       cout << "Szyfruje '" << argv[ 1] << "' i zapisuje '" << argv[ 2] << "'";
       in >> noskipws;
       while( in >> znak)
```



Rysunek 2.6. *Wynik szyfrowania pliku źródłowego z ćwiczenia 2.5*

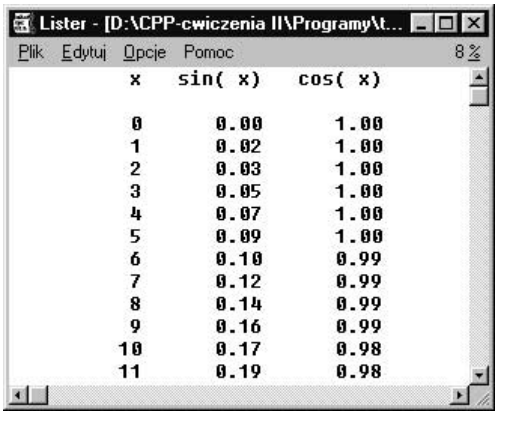
```
        {
          if( znak >= 32 && znak <= 126)
             ++ znak;      //szyfrowanie
          out << znak;
        }
      }
   }
}
```

Ćwiczenie 2.7.

Niech program wyprowadzi do pliku tablice.info wartości sinusa i cosinusa dla kątów z zakresu od 0 do 180 stopni co 1 stopień z dokładnością do dwóch miejsc po przecinku. Niech informacja będzie starannie sformatowana (rysunek 2.7):

```
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;
int main()
{
 cout << "Generacja pliku 'tablice.info'";
 ofstream out( "tablice.info");
 if( out)
 {
   int szer = 10;
   double x;
   out.width( szer);
   out << "x";
   out.width( szer);
   out << "sin( x)";
   out.width( szer);
   out << "cos( x)" << endl << endl;
```

Rysunek 2.7.
Tablice matematyczne, wygenerowane przez program




```
    out.precision( 2);   //dwa miejsca po przecinku
    out << showpoint;    //zawsze widać kropkę dziesiętną
    out << fixed;        //liczby nie w notacji wykładniczej
    for( int alfa = 0; alfa <= 180; ++alfa)
    {
       x = alfa * M_PI / 180.;
       out.width( szer);
       out << alfa;
       out.width( szer);
       out << sin( x);
       out.width( szer);
       out << cos( x) << endl;
    }
  }
}
```

Funkcja `width()` ustala szerokość pola dla najbliższej operacji zapisu, a `precision()` określa liczbę miejsc po przecinku w liczbie rzeczywistej. Manipulator `showpoint` nakazuje wyświetlanie kropki dziesiętnej, nawet gdy nie jest to konieczne, natomiast `fixed` wymusza stosowanie zapisu dziesiętnego, a nie wykładniczego w stylu *1.2e-2*.

Podsumowanie

Obiekty `ifstream` i `ostream` reprezentują wejście i wyjście do i z pliku dyskowego. Posiadają konstruktory, umożliwiające łatwe kojarzenie z plikiem za pomocą jego nazwy.

Odczytywanie i zapisywanie sformatowanych danych wykonują (między innymi) operatory >> i <<. Operatory te są przeciążone i potrafią pracować z danymi różnych typów. Potrafisz też dalej je przeciążać, definiując pracę z własnymi typami danych (porównaj ćwiczenie 5.8).

Obiekty reprezentujące wejście i wyjście posiadają kilka użytecznych funkcji formatujących (między innymi `width()`, `precision()`) oraz manipulatorów (między innymi `endl`, `showpoint`, `fixed`).