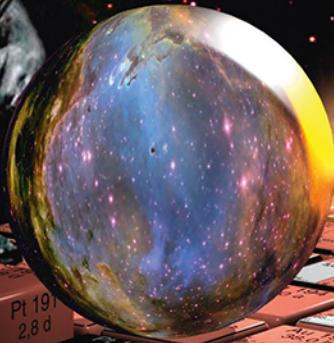


Jerzy Grębosz

OPUS MAGNUM C++11

Programowanie w języku C++



WYDANIE II
POPRAWIONE

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Helion SA dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki i opracowanie graficzne książki: Jerzy Grębosz

Zdjęcie Mgławicy Orzeł w grafice na okładce oraz zdjęcia łożnika Curiosity i powierzchni Marsa – wykorzystane w tytułach rozdziałów – dzięki uprzejmości NASA.

Wydanie drugie

ISBN: 978-83-283-6965-8

Copyright © Jerzy Grębosz 2020

Printed in Poland

Helion SA

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/ocpp12>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:

<ftp://ftp.helion.pl/przyklady/ocpp12.zip>

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści trzech tomów

0	Proszę tego nie czytać!	1
0.1	Zaprzyżnijmy się!	1
1	Startujemy!	8
1.1	Pierwszy program	8
1.2	Drugi program	13
1.3	Ćwiczenia	18
2	Instrukcje sterujące	20
2.1	Prawda – fałsz, czyli o warunkach	20
2.1.1	Wyrażenie logiczne.....	20
2.1.2	Zmienna logiczna <i>bool</i> w roli warunku.....	21
2.1.3	Stare dobre sposoby z dawnego C++	21
2.2	Instrukcja warunkowa <i>if</i>	22
2.3	Pętla <i>while</i>	26
2.4	Pętla <i>do...while</i>	27
2.5	Pętla <i>for</i>	28
2.6	Instrukcja <i>switch</i>	31
2.7	Co wybrać: <i>switch</i> czy <i>if...else</i> ?	33
2.8	Instrukcja <i>break</i>	36
2.9	Instrukcja <i>goto</i>	37
2.10	Instrukcja <i>continue</i>	39
2.11	Klamry w instrukcjach sterujących.....	40
2.12	Ćwiczenia	41
3	Typy	44
3.1	Deklaracje typu.....	44
3.2	Systematyka typów z języka C++	45
3.3	Typy fundamentalne.....	46
3.3.1	Typy przeznaczone do pracy z liczbami całkowitymi.....	46
3.3.2	Typy do przechowywania znaków alfanumerycznych.....	47
3.3.3	Typy reprezentujące liczby zmiennoprzecinkowe	47
3.3.4	<i>bool</i> – typ do reprezentacji obiektów logicznych.....	48
3.3.5	Kwestia dokładności	49
3.3.6	Jak poznać limity (ograniczenia) typów wbudowanych.....	51
3.4	Typy o precyzyjnie żądanej szerokości	55

3.5	InicjaLIZacja, czyli nadanie wartości w momencie narodzin.....	59
3.6	Definiowanie obiektów „w biegu”	60
3.7	Stałe dosłowne.....	62
3.7.1	Stałe dosłowne typu <i>bool</i>	63
3.7.2	Stałe będące liczbami całkowitymi	63
3.7.3	Stałe reprezentujące liczby zmiennoprzecinkowe.....	66
3.7.4	Stała dosłowna <i>nullptr</i> – dla wskaźników	67
3.7.5	Stałe znakowe	68
3.7.6	Stałe tekstowe, napisy, albo po prostu stringi	71
3.7.7	Surowe stałe tekstowe (napisy, stringi)	73
3.8	Typy złożone	76
3.9	Typ <i>void</i>	77
3.10	Zakres ważności nazwy obiektu a czas życia obiektu	78
3.10.1	Zakres: lokalny	78
3.10.2	Zakres: instrukcja.....	79
3.10.3	Zakres: blok funkcji	79
3.10.4	Zakres: obszar pliku	80
3.10.5	Zakres: obszar klasy	80
3.10.6	Zakres określony przez przestrzeń nazw	80
3.11	Zastąpienie nazw	85
3.12	Specyfikator (przydomek) <i>const</i>	87
3.13	Specyfikator (przydomek) <i>constexpr</i>	88
3.14	Obiekty <i>register</i>	92
3.15	Specyfikator <i>volatile</i>	92
3.16	<i>using</i> oraz <i>typedef</i> – tworzenie dodatkowej nazwy typu	93
3.17	Typy wyczerpieniowe <i>enum</i>	96
3.17.1	Dawne zwykłe <i>enum</i> a nowe zakresowe <i>enum class</i>	103
3.17.2	Kilka uwag dla wtajemniczonych	105
3.18	<i>auto</i> , czyli automatyczne rozpoznawanie typu definiowanego obiektu	106
3.19	<i>decltype</i> – operator do określania typu zadanego wyrażenia.....	109
3.20	Inicjalizacja z pustą klamrą { }, czyli wartością domniemaną	111
3.21	Przydomek <i>alignas</i> – adresy równe i równiejsze	113
3.22	Ćwiczenia	115

4 Operatory119

4.1	Operatory arytmetyczne	119
4.1.1	Operator %, czyli reszta z dzielenia (modulo)	120
4.1.2	Jednoargumentowe operatory + i –.....	121
4.1.3	Operatory inkrementacji i dekrementacji	121
4.1.4	Operator przypisania =	123
4.2	Operatory logiczne	124
4.2.1	Operatory relacji	124
4.2.2	Operatory sumy logicznej oraz iloczynu logicznego &&.....	125
4.2.3	Wykrzyknik !, czyli operator negacji	126
4.3	Operatory bitowe	127
4.3.1	Przesunięcie w lewo <<	128
4.3.2	Przesunięcie w prawo >>	129
4.3.3	Bitowe operatory sumy, iloczynu, negacji, różnicy symetrycznej.....	130
4.4	Różnica między operatorami logicznymi a operatorami bitowymi	130
4.5	Pozostałe operatory przypisania	132
4.6	Operator uzyskiwania adresu (operator &).....	133
4.7	Wyrażenie warunkowe	134
4.8	Operator <i>sizeof</i>	135
4.9	Operator <i>noexcept</i>	137
4.10	Deklaracja <i>static_assert</i>	137

4.11	Operator <i>alignof</i> informujący o najkorzystniejszym wyrównaniu adresu.....	139
4.12	Operatory rzutowania	141
4.12.1	Rzutowanie według tradycyjnych (niezalecanych) sposobów	141
4.12.2	Rzutowanie za pomocą nowych operatorów rzutowania	142
4.12.3	Operator <i>static_cast</i>	143
4.12.4	Operator <i>const_cast</i>	145
4.12.5	Operator <i>dynamic_cast</i>	146
4.12.6	Operator <i>reinterpret_cast</i>	147
4.13	Operator: przecinek	148
4.14	Priorytety operatorów	148
4.15	Łączność operatorów	151
4.16	Ćwiczenia	152

5 Typ *string* i typ *vector* – pierwsza wzmianka156

5.1	Typ <i>std::string</i> do pracy z tekstami	156
5.2	Typ <i>vector</i> – długi rząd obiektów.....	161
5.3	Zakresowe <i>for</i>	169
5.4	Ćwiczenia	172

6 Funkcje174

6.1	Definicja funkcji i jej wywołanie	174
6.2	Deklaracja funkcji	175
6.3	Funkcja często wywołuje inną funkcję.....	177
6.4	Zwracanie przez funkcję rezultatu.....	177
6.4.1	Obiekt tworzony za pomocą <i>auto</i> , a inicjalizowany rezultatem funkcji	179
6.4.2	O zwracaniu (lub niezwracaniu) rezultatu przez funkcję <i>main</i>	180
6.5	Nowy, alternatywny sposób deklaracji funkcji.....	181
6.6	Stos.....	183
6.7	Przesyłanie argumentów do funkcji przez wartość.....	184
6.8	Przesyłanie argumentów przez referencję.....	185
6.9	Pożyteczne określenia: l-wartość i r-wartość	188
6.10	Referencje do l-wartości i referencje do r-wartości jako argumenty funkcji	190
6.10.1	Który sposób przesyłania argumentu do funkcji wybrać?.....	197
6.11	Kiedy deklaracja funkcji nie jest konieczna?	198
6.12	Argumenty domniemane	199
6.12.1	Ciekawostki na temat argumentów domniemanych.....	202
6.13	Nienazwany argument	207
6.14	Funkcje <i>inline</i> (w linii).....	208
6.15	Przypomnienie o zakresie ważności nazw deklarowanych wewnątrz funkcji	212
6.16	Wybór zakresu ważności nazwy i czasu życia obiektu.....	212
6.16.1	Obiekty globalne	212
6.16.2	Obiekty automatyczne.....	213
6.16.3	Obiekty lokalne statyczne	214
6.17	Funkcje w programie składającym się z kilku plików	218
6.17.1	Nazwy statyczne globalne	222
6.18	Funkcja zwracająca rezultat będący referencją l-wartości	223
6.19	Funkcje rekurencyjne	228
6.20	Funkcje biblioteczne.....	237
6.21	Funkcje <i>constexpr</i>	240
6.21.1	Wymogi, które musi spełniać funkcja <i>constexpr</i> (w standardzie C++11)	242
6.21.2	Przykład pokazujący aspekty funkcji <i>constexpr</i>	243
6.21.3	Argumenty funkcji <i>constexpr</i> będące referencjami	252
6.22	Definiowanie referencji przy użyciu słowa <i>auto</i>	253
6.22.1	Gdy inicjalizatorem jest wywołanie funkcji zwracającej referencję.....	260

6.23	Ćwiczenia	263
7	Preprocesor	269
7.1	Dyrektywa pusta #	269
7.2	Dyrektywa <code>#define</code>	269
7.3	Dyrektywa <code>#undef</code>	271
7.4	Makrodefinicje	272
7.5	Sklejacz nazw argumentów, czyli operator <code>##</code>	274
7.6	Parametr aktualny makrodefinicji – w postaci tekstu	275
7.7	Dyrektywy kompilacji warunkowej	275
7.8	Dyrektywa <code>#error</code>	279
7.9	Dyrektywa <code>#line</code>	280
7.10	Wstawianie treści innych plików do tekstu skompilowanego właśnie pliku	280
7.11	Dyrektywy zależne od implementacji	282
7.12	Nazwy predefiniowane	282
7.13	Ćwiczenia	285
8	Tablice	288
8.1	Co to jest tablica	288
8.2	Elementy tablicy	289
8.3	Inicjalizacja tablic	291
8.4	Przekazywanie tablicy do funkcji	292
8.5	Przykład z tablicą elementów typu <i>enum</i>	296
8.6	Tablice znakowe	298
8.7	Ćwiczenia	306
9	Tablice wielowymiarowe	311
9.1	Tablica tablic	311
9.2	Przykład programu pracującego z tablicą dwuwymiarową	313
9.3	Gdzie w pamięci jest dany element tablicy	315
9.4	Typ wyrażeń związanych z tablicą wielowymiarową	315
9.5	Przesyłanie tablic wielowymiarowych do funkcji	317
9.6	Ćwiczenia	319
10	Wektory wielowymiarowe	321
10.1	Najpierw przypomnienie istotnych tu cech klasy <i>vector</i>	321
10.2	Jak za pomocą klasy <i>vector</i> budować tablice wielowymiarowe	322
10.3	Funkcja pokazująca zawartość wektora dwuwymiarowego	323
10.4	Definicja dwuwymiarowego wektora – pustego	325
10.5	Definicja wektora dwuwymiarowego z listą inicjalizatorów	326
10.6	Wektor dwuwymiarowy o żądanych rozmiarach, choć bez inicjalizacji	327
10.7	Zmiana rozmiaru wektora 2D funkcją <i>resize</i>	328
10.8	Zmiany rozmiaru wektora 2D funkcjami <i>push_back</i> , <i>pop_back</i>	329
10.9	Zmniejszanie rozmiaru wektora dwuwymiarowego funkcją <i>pop_back</i>	332
10.10	Funkcje mogące modyfikować treść wektora 2D	332
10.11	Wstawianie rzędu wektora 2D do funkcji pracującej z wektorem 1D	334
10.12	Całość przykładu definiującego wektory dwuwymiarowe	335
10.13	Po co są dwuwymiarowe wektory nieprostokątne	335
10.14	Wektory trójwymiarowe	337
10.15	Sposoby definicji wektora 3D o ustalonych rozmiarach	340
10.16	Nadawanie pustemu wektorowi 3D wymaganych rozmiarów	344
10.16.1	Zmiana rozmiarów wektora 3D funkcjami <i>resize</i>	344

10.16.2	Zmiana rozmiarów wektora 3D funkcjami <i>push_back</i>	346
10.17	Trójwymiarowe wektory 3D – nieprostokątne.....	347
10.18	Ćwiczenia.....	351

11 Wskaźniki – wiadomości wstępne353

11.1	Wskaźniki mogą bardzo ułatwić życie.....	353
11.2	Definiowanie wskaźników.....	355
11.3	Praca ze wskaźnikiem.....	356
11.4	Definiowanie wskaźnika z użyciem <i>auto</i>	359
11.5	Wyrażenie <i>*wskaźnik jest l-wartością</i>	360
11.6	Operator rzutowania <i>reinterpret_cast</i> a wskaźniki.....	360
11.7	Wskaźniki typu <i>void*</i>	363
11.8	Strzał na oślep – wskaźnik zawsze na coś wskazuje.....	365
11.8.1	Wskaźnik wolno porównać z adresem zero – <i>nullptr</i>	367
11.9	Ćwiczenia.....	367

12 Cztery domeny zastosowania wskaźników369

12.1	Zastosowanie wskaźników wobec tablic.....	369
12.1.1	Ćwiczenia z mechaniki ruchu wskaźnika.....	369
12.1.2	Użycie wskaźnika w pracy z tablicą.....	373
12.1.3	Arytmetyka wskaźników.....	377
12.1.4	Porównywanie wskaźników.....	379
12.2	Zastosowanie wskaźników w argumentach funkcji.....	380
12.2.1	Jeszcze raz o przesyłaniu tablic do funkcji.....	384
12.2.2	Odbieranie tablicy jako wskaźnika.....	384
12.2.3	Argument formalny będący wskaźnikiem do obiektu <i>const</i>	386
12.3	Zastosowanie wskaźników przy dostępie do konkretnych komórek pamięci.....	389
12.4	Rezerwacja obszarów pamięci.....	390
12.4.1	Operatory <i>new</i> i <i>delete</i> albo Oratorium Stworzenie Świata.....	391
12.4.2	Operator <i>new</i> a słowo kluczowe <i>auto</i>	395
12.4.3	Inicjalizacja obiektu tworzonoego operatorem <i>new</i>	395
12.4.4	Operatorem <i>new</i> możemy także tworzyć obiekty stałe.....	396
12.4.5	Dynamiczna alokacja tablicy.....	397
12.4.6	Tablice wielowymiarowe tworzone operatorem <i>new</i>	398
12.4.7	Umiejszczający operator <i>new</i>	401
12.4.8	„Przychodzimy, odchodzimy – cichuteńko, na...”.....	406
12.4.9	Zapas pamięci to nie studnia bez dna.....	408
12.4.10	Nowy sposób powiadomienia: rzucenie wyjątku <i>std::bad_alloc</i>	409
12.4.11	Funkcja <i>set_new_handler</i>	411
12.5	Ćwiczenia.....	413

13 Wskaźniki – runda trzecia.....417

13.1	Stałe wskaźniki.....	417
13.2	Stałe wskaźniki a wskaźniki do stałych.....	418
13.2.1	Wierzch i głębia.....	419
13.3	Definiowanie wskaźnika z użyciem <i>auto</i>	420
13.3.1	Symbol zastępczy <i>auto</i> a opuszczanie gwiazdki przy definiowaniu wskaźnika.....	423
13.4	Sposoby ustawiania wskaźników.....	425
13.5	Parada kłamców, czyli o rzutowaniu <i>const_cast</i>	427
13.6	Tablice wskaźników.....	431
13.7	Wariacje na temat C-stringów.....	433
13.8	Argumenty z linii wywołania programu.....	440
13.9	Ćwiczenia.....	443

14 Wskaźniki do funkcji445

14.1	Wskaźnik, który może wskazywać na funkcję	445
14.2	Ćwiczenia z definiowania wskaźników do funkcji	448
14.3	Wskaźnik do funkcji jako argument innej funkcji.....	454
14.4	Tablica wskaźników do funkcji.....	458
14.5	Użycie deklaracji <i>using</i> i <i>typedef</i> w świecie wskaźników	463
14.5.1	Alias przydatny w argumencie funkcji.....	463
14.5.2	Alias przydatny w definicji tablicy wskaźników do funkcji	464
14.6	Użycie <i>auto</i> lub <i>decltype</i> do automatycznego rozpoznania potrzebnego typu	465
14.7	Ćwiczenia	467

15 Przeladowanie nazwy funkcji469

15.1	Co oznacza przeladowanie	469
15.2	Przeladowanie od kuchni.....	472
15.3	Jak możemy przeladowywać, a jak się nie da?.....	472
15.4	Czy przeladowanie nazw funkcji jest techniką orientowaną obiektowo?.....	475
15.5	Linkowanie z modułami z innych języków	476
15.6	Przeladowanie a zakres ważności deklaracji funkcji	477
15.7	Rozważania o identyczności lub odmienności typów argumentów.....	479
15.7.1	Przeladowanie a typy tworzone z <i>using</i> lub <i>typedef</i> oraz typy <i>enum</i>	480
15.7.2	Tablica a wskaźnik	480
15.7.3	Pewne szczegóły o tablicach wielowymiarowych.....	481
15.7.4	Przeladowanie a referencja.....	483
15.7.5	Identyczność typów: <i>T</i> , <i>const T</i> , <i>volatile T</i>	484
15.7.6	Przeladowanie a typy: <i>T*</i> , <i>volatile T*</i> , <i>const T*</i>	485
15.7.7	Przeladowanie a typy: <i>T&</i> , <i>volatile T&</i> , <i>const T&</i>	486
15.8	Adres funkcji przeladowanej.....	487
15.8.1	Zwrot rezultatu będącego adresem funkcji przeladowanej.....	489
15.9	Kulisy dopasowywania argumentów do funkcji przeladowanych.....	491
15.10	Etapy dopasowania	492
15.10.1	Etap 1. Dopasowanie dokładne, bo konwersja niepotrzebna	492
15.10.2	Etap 1a. Dopasowanie dokładne, bo z tzw. trywialną konwersją.....	493
15.10.3	Etap 2. Dopasowanie z awansem (z promocją).....	494
15.10.4	Etap 3. Próba dopasowania za pomocą konwersji standardowych	496
15.10.5	Etap 4. Dopasowanie z użyciem konwersji zdefiniowanych przez użytkownika	498
15.10.6	Etap 5. Dopasowanie do funkcji z wielokropkiem	498
15.11	Wskaźników nie dopasowuje się inaczej niż dosłownie.....	498
15.12	Dopasowywanie wywołań z kilkoma argumentami	499
15.13	Ćwiczenia	500

16 Klasy503

16.1	Typy definiowane przez użytkownika.....	503
16.2	Składniki klasy	505
16.3	Składnik będący obiektem	506
16.4	Kapsułowanie	507
16.5	Ukrywanie informacji.....	508
16.6	Klasa a obiekt	511
16.7	Wartości wstępne w składnikach nowych obiektów. Inicjalizacja „w klasie”	513
16.8	Funkcje składowe	516
16.8.1	Posługiwanie się funkcjami składowymi	516
16.8.2	Definiowanie funkcji składowych	517
16.9	Jak to właściwie jest? (<i>this</i>).....	522
16.10	Odwołanie się do publicznych danych składowych obiektu	524

16.11	Zasłanianie nazw	525
16.11.1	Nie sięgaj z klasy do obiektów globalnych.....	528
16.12	Przeładowanie i zasłonięcie równocześnie	529
16.13	Nowa klasa? Osobny plik!.....	529
16.13.1	Poznajmy praktyczną realizację wieloplukowego programu	532
16.13.2	Zasada umieszczania dyrektywy <i>using namespace</i> w plikach	544
16.14	Przesyłanie do funkcji argumentów będących obiektami.....	544
16.14.1	Przesyłanie obiektu przez wartość	544
16.14.2	Przesyłanie przez referencję	546
16.15	Konstruktor – pierwsza wzmianka	547
16.16	Destruktor – pierwsza wzmianka	552
16.17	Składnik statyczny.....	556
16.17.1	Do czego może się przydać składnik statyczny w klasie?.....	565
16.18	Statyczna funkcja składowa.....	565
16.18.1	Deklaracja składnika statycznego mająca inicjalizację „w klasie”.....	570
16.19	Funkcje składowe typu <i>const</i> oraz <i>volatile</i>	576
16.19.1	Przeładowanie a funkcje składowe <i>const</i> i <i>volatile</i>	580
16.20	Struktura.....	580
16.21	Klasa będąca agregatem. Klasa bez konstruktora.....	581
16.22	Funkcje składowe z przydomkiem <i>constexpr</i>	583
16.23	Specyfikator <i>mutable</i>	590
16.24	Bardziej rozbudowany przykład zastosowania klasy	591
16.25	Cwiczenia	602

17 Biblioteczna klasa *std::string*607

17.1	Rozwiązanie przechowywania tekstów musiało się znaleźć	607
17.2	Klasa <i>std::string</i> to przecież nasz stary znajomy	609
17.3	Definiowanie obiektów klasy <i>string</i>	610
17.4	Użycie operatorów =, +, += w pracy ze stringami.....	615
17.5	Pojemność, rozmiar i długość stringu.....	616
17.5.1	Blizniacze funkcje <i>size()</i> i <i>length()</i>	616
17.5.2	Funkcja składowa <i>empty</i>	617
17.5.3	Funkcja składowa <i>max_size</i>	617
17.5.4	Funkcja składowa <i>capacity</i>	617
17.5.5	Funkcje składowe <i>reserve</i> i <i>shrink_to_fit</i>	619
17.5.6	<i>resize</i> – zmiana długości stringu „na siłę”	620
17.5.7	Funkcja składowa <i>clear</i>	622
17.6	Użycie operatora <i>[]</i> oraz funkcji <i>at</i>	622
17.6.1	Działanie operatora <i>[]</i>	623
17.6.2	Działanie funkcji składowej <i>at</i>	624
17.6.3	Przebieganie po wszystkich literach stringu zakresowym <i>for</i>	627
17.7	Funkcje składowe <i>front</i> i <i>back</i>	627
17.8	Jak umieścić w tekście liczbę?.....	628
17.9	Jak wczytać liczbę ze stringu?	630
17.10	Praca z fragmentem stringu, czyli z substringiem	633
17.11	Funkcja składowa <i>substr</i>	634
17.12	Szukanie zadanego substringu w obiekcie klasy <i>string</i> – funkcje <i>find</i>	635
17.13	Szukanie rozpoczynane od końca stringu.....	638
17.14	Szukanie w stringu jednego ze znaków z zadanego zestawu.....	639
17.15	Usuwanie znaków ze stringu – <i>erase</i> i <i>pop_back</i>	641
17.16	Wstawianie znaków do istniejącego stringu – funkcje <i>insert</i>	642
17.17	Zamiana części znaków na inne znaki – <i>replace</i>	644
17.18	Zagłądanie do wnętrza obiektu klasy <i>string</i> funkcją <i>data</i>	647
17.19	Zawartość obiektu klasy <i>string</i> a C-string	648

17.20	W porządku alfabetycznym, czyli porównywanie stringów	651
17.20.1	Porównywanie stringów za pomocą funkcji <i>compare</i>	652
17.20.2	Porównywanie stringów przy użyciu operatorów <code>==</code> , <code>!=</code> , <code><</code> , <code>></code> , <code><=</code> , <code>>=</code>	656
17.21	Zamiana treści stringu na małe lub wielkie litery	657
17.22	Kopiowanie treści obiektu klasy <i>string</i> do tablicy znakowej – funkcja <i>copy</i>	659
17.23	Wzajemna zamiana treści dwóch obiektów klasy <i>string</i> – funkcja <i>swap</i>	660
17.24	Wczytywanie z klawiatury stringu o nieznannej wcześniej długości – <i>getline</i>	661
17.24.1	Pułapka, czyli jak <i>getline</i> może Cię zaskoczyć	664
17.25	Iteratory stringu	668
17.25.1	Iterator do obiektu stałego	672
17.25.2	Funkcje składowe klasy <i>string</i> pracujące z iteratorami	673
17.26	Klasa <i>string</i> korzysta z techniki przenoszenia	678
17.27	Bryk, czyli „pamięć zewnętrzna” programisty	679
17.28	Ćwiczenia	687

18 Deklaracje przyjaźni694

18.1	Przyjaciele w życiu i w C++	694
18.2	Przykład: dwie klasy deklarują przyjaźń z tą samą funkcją	696
18.3	W przyjaźni trzeba pamiętać o kilku sprawach	698
18.4	Obdarzenie przyjaźnią funkcji składowej innej klasy	701
18.5	Klasy zaprzyjaźnione	703
18.6	Konwencja umieszczania deklaracji przyjaźni w klasie	705
18.7	Kilka otrzeźwiających słów na zakończenie	705
18.8	Ćwiczenia	706

19 Obsługa sytuacji wyjątkowych708

19.1	Jak dać znać, że coś się nie udało?	708
19.2	Pierwszy prosty przykład	710
19.3	Kolejność bloków <i>catch</i> ma znaczenie	712
19.4	Który blok <i>catch</i> nadaje się do złapania lecącego wyjątku?	713
19.5	Bloki <i>try</i> mogą być zagnieżdżane	715
19.6	Obsługa wyjątków w praktycznym programie	718
19.7	Specyfikator <i>noexcept</i> i operator <i>noexcept</i>	729
19.8	Ćwiczenia	732

20 Klasa-składnik oraz klasa lokalna734

20.1	Klasa-składnik, czyli gdy w klasie jest zagnieżdżona definicja innej klasy	734
20.2	Prawdziwy przykład zagnieżdżenia definicji klasy	741
20.3	Lokalna definicja klasy	752
20.4	Lokalne nazwy typów	755
20.5	Ćwiczenia	756

21 Konstruktory i destruktory758

21.1	Konstruktor	758
21.1.1	Przykład programu zawierającego klasę z konstruktorami	759
21.2	Specyfikator (przydomek) <i>explicit</i>	770
21.3	Kiedy i jak wywoływany jest konstruktor	771
21.3.1	Konstruowanie obiektów lokalnych	771
21.3.2	Konstruowanie obiektów globalnych	772
21.3.3	Konstrukcja obiektów tworzonych operatorem <i>new</i>	772
21.3.4	Jawne wywołanie konstruktora	773
21.3.5	Dalsze sytuacje, gdy pracuje konstruktor	776

21.4	Destruktor.....	776
21.4.1	Jawne wywołanie destruktora (ogromnie rzadka sytuacja).....	778
21.5	Nie rzucajcie wyjątków z destruktorów	778
21.6	Konstruktor domniemany	780
21.7	Funkcje składowe z przypiskami = <i>default</i> i = <i>delete</i>	781
21.8	Konstruktorowa lista inicjalizacyjna składników klasy	783
21.8.1	Dla wtajemniczonych: wyjątki rzucane z konstruktorowej listy inicjalizacyjnej.....	790
21.9	Konstruktor delegujący	794
21.10	Pomocnicza klasa <i>std::initializer_list</i> – lista inicjalizatorów.....	801
21.10.1	Zastosowania niekonstruktorowe.....	801
21.10.2	Konfuzja: lista inicjalizatorów a lista inicjalizacyjna.....	810
21.10.3	Konstruktor z argumentem będącym klamrową listą inicjalizatorów	811
21.11	Konstrukcja obiektu, którego składnikiem jest obiekt innej klasy	816
21.12	Konstruktory niepubliczne?	823
21.13	Konstruktory <i>constexpr</i> mogą wytwarzać obiekty <i>constexpr</i>	825
21.14	Ćwiczenia	835

22 Konstruktory: kopiujący i przenoszący838

22.1	Konstruktor kopiujący (albo inicjalizator kopiujący)	838
22.2	Przykład klasy z konstruktorem kopiującym.....	839
22.3	Kompilatorowi wolno pominąć niepotrzebne kopiowanie	844
22.4	Dlaczego przez referencję?.....	846
22.5	Konstruktor kopiujący gwarantujący nietykalność	847
22.6	Współodpowiedzialność.....	848
22.7	Konstruktor kopiujący generowany automatycznie	848
22.8	Kiedy powinniśmy sami zdefiniować konstruktor kopiujący?.....	849
22.9	Referencja do r-wartości daje zezwolenie na recykling.....	856
22.10	Funkcja <i>std::move</i> , która nie przenosi, a tylko rzutuje.....	859
22.11	Odebrana r-wartość staje się w ciele funkcji l-wartością	861
22.12	Konstruktor przenoszący (inicjalizator przenoszący).....	863
22.12.1	Konstruktor przenoszący generowany przez kompilator	868
22.12.2	Inne konstruktory generowane automatycznie.....	868
22.12.3	Zwrot obiektu lokalnego przez wartość? Nie używamy przenoszenia!.....	869
22.13	Tak zwana „semantyka przenoszenia”.....	870
22.14	Nowe pojęcia dla ambitnych: gl-wartość, x-wartość i pr-wartość	870
22.15	<i>decltype</i> – operator rozpoznawania typu bardzo wyszukanych wyrażeń.....	873
22.16	Ćwiczenia	878

23 Tablice obiektów880

23.1	Definiowanie tablic obiektów i praca z nimi	880
23.2	Tablica obiektów definiowana operatorem <i>new</i>	881
23.3	Inicjalizacja tablic obiektów	883
23.3.1	Inicjalizacja tablicy, której obiekty są agregatami	883
23.3.2	Inicjalizacja tablic, których elementy nie są agregatami	886
23.4	Wektory obiektów	890
23.4.1	Wektor, którego elementami są obiekty klasy będącej agregatem.....	892
23.4.2	Wektor, którego elementami są obiekty klasy niebędącej agregatem	894
23.5	Ćwiczenia	895

24 Wskaźnik do składników klasy896

24.1	Wskaźniki zwykłe – repetytorium.....	896
24.2	Wskaźnik do pokazywania na składnik-daną	897
24.2.1	Przykład zastosowania wskaźników do składników klasy.....	901
24.3	Wskaźnik do funkcji składowej.....	908

24.3.1	Przykład zastosowania wskaźników do funkcji składowych	910
24.4	Tablica wskaźników do danych składowych klasy	917
24.5	Tablica wskaźników do funkcji składowych klasy	918
24.5.1	Przykład tablicy/wektora wskaźników do funkcji składowych.....	919
24.6	Wskaźniki do składników statycznych są zwykłe	922
24.7	Ćwiczenia	923

25 Konwersje definiowane przez użytkownika.....925

25.1	Sformułowanie problemu	925
25.2	Konstruktory konwertujące	927
25.2.1	Kiedy jawnie, kiedy niejawnie	928
25.2.2	Przykład konwersji konstruktorem.....	933
25.3	Funkcja konwertująca – operator konwersji.....	935
25.3.1	Na co funkcja konwertująca zamieniać nie może.....	941
25.4	Który wariant konwersji wybrać?.....	942
25.5	Sytuacje, w których zachodzi konwersja.....	944
25.6	Zapis jawnego wywołania konwersji typów.....	945
25.6.1	Advocatus zapisu przypominającego: „wywołanie funkcji”	945
25.6.2	Advocatus zapisu: „rzutowanie”	946
25.7	Nie całkiem pasujące argumenty, czyli konwersje kompilatora przy dopasowaniu	946
25.8	Kilka rad dotyczących konwersji.....	951
25.9	Ćwiczenia	952

26 Przeładowanie operatorów.....954

26.1	Co to znaczy przeładować operator?	954
26.2	Przeładowanie operatorów – definicja i trochę teorii	956
26.3	Moje zabawki	960
26.4	Funkcja operatorowa jako funkcja składowa	961
26.5	Funkcja operatorowa nie musi być przyjacielem klasy	964
26.6	Operatory predefiniowane	964
26.7	Ile operandów ma mieć ten operator?	965
26.8	Operatory jednooperandowe	965
26.9	Operatory dwuoperandowe	968
26.9.1	Przykład na przeładowanie operatora dwuoperandowego	968
26.9.2	Przemienność	970
26.9.3	Choć operatory inne, to nazwę mają tę samą	971
26.10	Przykład zupełnie niematematyczny	971
26.11	Operatory postinkrementacji i postdekrementacji – koniec z niesprawiedliwością.....	981
26.12	Praktyczne rady dotyczące przeładowania	983
26.13	Pojedynek: operator jako funkcja składowa czy globalna?	985
26.14	Zasłona spada, czyli tajemnica operatora <<.....	986
26.15	Stałe dosłowne definiowane przez użytkownika	992
26.15.1	Przykład: stałe dosłowne użytkownika odbierane jako gotowane	996
26.15.2	Przykład: stałe dosłowne użytkownika odbierane na surowo	1005
26.16	Ćwiczenia	1008

27 Przeładowanie: =, [,], (), ->.....1012

27.1	Cztery operatory, które muszą być niestatycznymi funkcjami składowymi.....	1012
27.2	Operator przypisania = (wersja kopiująca)	1012
27.2.1	Przykład na przeładowanie (kopiującego) operatora przypisania	1014
27.2.2	Przypisanie „kaskadowe”	1021
27.2.3	Po co i jak zabezpieczamy się przed przypisaniem $a = a$	1023
27.2.4	Jak opowiedzieć potocznie o konieczności istnienia operatora przypisania?.....	1024
27.2.5	Kiedy kopiujący operator przypisania nie jest generowany automatycznie	1026

27.3	Przenoszący operator przypisania =	1026
27.4	Specjalne funkcje składowe i nierealna prosta zasada.....	1035
27.5	Operator [].....	1036
27.6	Operator ().....	1040
27.7	Operator →	1046
27.7.1	„Sprytny wskaźnik” wykorzystuje przeładowanie właśnie tego operatora	1048
27.8	Ćwiczenia	1055

28 Przeładowanie operatorów *new* i *delete* na użytek klasy1057

28.1	Po co przeładujemy operatory <i>new</i> i <i>new[]</i>	1057
28.2	Funkcja <i>operator new</i> i <i>operator new[]</i> w klasie K	1058
28.3	Jak się deklaruje operatory <i>new</i> i <i>delete</i> w klasie?.....	1061
28.4	Przykładowy program z przeładowanymi <i>new</i> i <i>delete</i>	1063
28.4.1	Gdy dopuszczamy rzucanie wyjątku <i>std::bad_alloc</i>	1064
28.4.2	Po staremu nadal można.....	1069
28.4.3	Rezerwacja tablicy obiektów naszej klasy <i>Twektorek</i>	1069
28.4.4	Nasze własne argumenty wysłane do operatora <i>new</i>	1071
28.4.5	👑 Operatory <i>new</i> i <i>delete</i> odziedziczone do klasy pochodnej.....	1073
28.4.6	A jednak polimorfizm jest możliwy	1075
28.4.7	Tworzenie i likwidowanie tablicy obiektów klasy pochodnej	1075
28.4.8	Operatory <i>new</i> , które nie rzucają wyjątku <i>std::bad_alloc</i>	1076
28.5	Rzut oka wstecz na przeładowanie operatorów	1081
28.6	Ćwiczenia	1082

29 Unie i pola bitowe1084

29.1	Unia	1084
29.2	Unia anonimowa.....	1086
29.3	Klasa uniopodobna (unia z metryczką)	1088
29.4	Gdy składnik unii jest obiektem jakiejś klasy	1090
29.5	Unia o składnikach mających swe konstruktory, destruktory itp.	1092
29.6	Pola bitowe	1099
29.7	Unia i pola bitowe upraszczają deszyfrowanie słów danych	1103
29.8	Ćwiczenia	1110

30 Wyrażenia lambda i wysłanie kodu do innych funkcji1114

30.1	Preludium: dwa sposoby przesłania kryterium oceniania.....	1114
30.1.1	Sposób I. Kryterium przekazane wskaźnikiem do funkcji (orzekającej)	1117
30.1.2	Sposób II. Kryterium umieszczone w obiekcie funkcyjnym.....	1119
30.1.3	Kryterium oceny z parametrem (czyli o wyższości funktorów).....	1121
30.1.4	Funkcja-algorytm biblioteczny <i>std::count_if</i>	1123
30.1.5	Co lepsze: funkcja orzekająca czy orzekający obiekt funkcyjny?.....	1126
30.2	Wyrażenie lambda	1128
30.3	Formy wyrażenia lambda	1133
30.3.1	Lista argumentów (formalnych).....	1134
30.3.2	Ciało wyrażenia lambda	1134
30.3.3	Typ rezultatu	1135
30.3.4	Lista wychwytywania.....	1136
30.3.5	Słowo kluczowe <i>mutable</i> w wyrażeniu lambda.....	1138
30.3.6	Specyfikacja dotycząca wyjątków rzucanych z wyrażenia lambda.....	1139
30.4	Wyrażenie lambda zastosowane w funkcji składowej.....	1139
30.5	Tworzenie (nazwanych) obiektów lambda słowem <i>auto</i>	1143
30.5.1	Tworzenie obiektów na lambdy słowem kluczowym <i>auto</i>	1144
30.5.2	Tworzenie (nazwanych) obiektów lambda szablonem <i>std::function</i>	1146

30.6	Stowarzyszenie martwych referencji	1151
30.7	Rekurencja przy użyciu wyrażenia lambda	1154
30.8	Wyrażenie lambda jako domniemana wartość argumentu.....	1158
30.9	Rzucanie wyjątków z wyrażenia lambda.....	1162
30.10	Vivat lambda!	1166
30.11	Ćwiczenia	1167

31 Dziedziczenie klas1170

31.1	Istota dziedziczenia.....	1170
31.2	Dostęp do składników	1173
31.2.1	Prywatne składniki klasy podstawowej.....	1173
31.2.2	Nieprywatne składniki klasy podstawowej.....	1175
31.2.3	Klasa pochodna też decyduje	1176
31.2.4	Deklaracja dostępu <i>using</i> , czyli udostępnianie wybiórcze	1178
31.3	Czego się nie dziedziczy.....	1181
31.3.1	„Niedziedziczenie” konstruktorów	1181
31.3.2	„Niedziedziczenie” operatora przypisania.....	1182
31.3.3	„Niedziedziczenie” destruktora	1182
31.4	Drzewo genealogiczne.....	1183
31.5	Dziedziczenie – doskonałe narzędzie programowania	1184
31.6	Kolejność wywoływania konstruktorów	1186
31.7	Przypisanie i inicjalizacja obiektów w warunkach dziedziczenia.....	1192
31.7.1	Klasa pochodna nie definiuje swojego kopiującego operatora przypisania	1192
31.7.2	Klasa pochodna nie definiuje swojego konstruktora kopiującego	1193
31.7.3	Inicjalizacja i przypisywanie według obiektu będącego <i>const</i>	1194
31.8	Przykład: konstruktor kopiujący i operator przypisania dla klasy pochodnej	1194
31.8.1	Jak zainstalować mechanizm kopiowania w klasie pochodnej	1200
31.8.2	Jak w klasie pochodnej zainstalować mechanizm przenoszenia	1204
31.9	Dziedziczenie od kilku „rodziców” (wielodziedziczenie)	1208
31.9.1	Konstruktor klasy pochodnej przy wielodziedziczeniu.....	1209
31.9.2	Ryzyko wieloznaczności przy wielodziedziczeniu	1212
31.9.3	Czy bliższe pokrewieństwo usuwa wieloznaczność?	1214
31.9.4	Poszlaki	1214
31.10	Sposób na „odziedziczenie” konstruktorów	1215
31.11	Pojedynek: dziedziczenie klasy contra zawieranie obiektów składowych	1222
31.12	Wspaniałe konwersje standardowe przy dziedziczeniu	1224
31.12.1	Panorama korzyści	1228
31.12.2	Czego się nie optać robić.....	1230
31.12.3	Tuzin samochodów nie jest rodzajem tuzina pojazdów	1231
31.12.4	Konwersje standardowe wskaźnika do składnika klasy	1235
31.13	Wirtualne klasy podstawowe.....	1237
31.13.1	Publiczne i prywatne dziedziczenie tej samej klasy wirtualnej	1241
31.13.2	Uwagi o konstrukcji i inicjalizacji w przypadku klas wirtualnych	1241
31.13.3	Dominacja klas wirtualnych.....	1245
31.14	Ćwiczenia	1246

32 Wirtualne funkcje składowe1253

32.1	Wirtualny znaczy: (teoretycznie) możliwy	1253
32.2	Polimorfizm	1260
32.3	Typy rezultatów różnych realizacji funkcji wirtualnej	1263
32.3.1	Zamiast „odpowiedni typ rezultatu” kompilator powie „kowariant”	1264
32.4	Dalsze cechy funkcji wirtualnej.....	1266
32.5	Wczesne i późne wiązanie.....	1268
32.6	Kiedy dla wywołań funkcji wirtualnych zachodzi jednak wczesne wiązanie?.....	1270
32.7	Kulisy białej magii, czyli jak to jest zrobione.....	1271

32.8	Funkcja wirtualna, a mimo to <i>inline</i>	1273
32.9	Destruktor? Najlepiej wirtualny!	1273
32.10	Pojedynek – funkcje przeładowane, zasłaniające się i wirtualne (zacierające się)	1275
32.11	Kontekstowe słowa kluczowe <i>override</i> i <i>final</i>	1277
32.11.1	Przykład użycia <i>override</i> i <i>final</i> , a także wirtualnych destruktorów	1279
32.12	Klasy abstrakcyjne.....	1290
32.13	Wprawdzie konstruktor nie może być wirtualny, ale... ..	1297
32.14	Rzutowanie <i>dynamic cast</i> jest dla typów polimorficznych.....	1303
32.15	POD, czyli Pospolite Stare Dane	1306
32.16	Wszystko, co najważniejsze	1310
32.17	Finis coronat opus	1312
32.18	Ćwiczenia	1312

33 Operacje wejścia/wyjścia – podstawy.....1316

33.1	Biblioteka <i>iostream</i>	1317
33.2	Strumień	1317
33.3	Strumienie zdefiniowane standardowo.....	1319
33.4	Operatory >> i <<.....	1320
33.5	Domniemania w pracy strumieni zdefiniowanych standardowo	1321
33.6	Uwaga na priorytet	1324
33.7	Operatory << oraz >> definiowane przez użytkownika	1325
33.7.1	Operatorów wstawiania i wyjmowania ze strumienia nie dziedziczy się	1330
33.7.2	Operatory wstawiania i wyjmowania nie mogą być wirtualne. Niestety.....	1331
33.8	Sterowanie formatem.....	1334
33.9	Flagi stanu formatowania	1334
33.9.1	Znaczenie poszczególnych flag sterowania formatem	1336
33.10	Sposoby zmiany trybu (reguł) formatowania	1341
33.11	Manipulatory	1341
33.11.1	Manipulatory bezargumentowe	1342
33.11.2	Manipulatory mające argumenty	1347
33.11.3	Manipulator <i>setw(int)</i>	1347
33.11.4	Manipulator <i>setfill</i>	1350
33.11.5	Manipulator <i>setprecision(int)</i>	1350
33.11.6	Manipulator <i>std::setbase(int)</i>	1352
33.11.7	Manipulatory <i>setiosflags</i> , <i>resetiosflags</i>	1353
33.11.8	Tabele z zestawieniem manipulatorów	1353
33.12	Definiowanie swoich manipulatorów	1355
33.12.1	Manipulator jako funkcja	1355
33.12.2	Definiowanie manipulatora z argumentem	1357
33.13	Zmiana sposobu formatowania funkcjami <i>setf</i> , <i>unsetf</i>	1360
33.14	Dodatkowe funkcje do zmiany parametrów formatowania	1366
33.14.1	Funkcja <i>width</i>	1367
33.14.2	Funkcja składowa <i>fill</i>	1368
33.14.3	Funkcja <i>precision</i>	1369
33.14.4	Funkcja <i>copyfmt</i>	1370
33.15	Nieformatowane operacje wejścia/wyjścia.....	1370
33.16	Omówienie funkcji wyjmujących ze strumienia.....	1372
33.16.1	Funkcje do pracy ze znakami i napisami.....	1372
33.16.2	Wczytywanie binarne – funkcja <i>read</i>	1378
33.16.3	Funkcja <i>ignore</i>	1379
33.16.4	Pożyteczne funkcje pomocnicze	1381
33.16.5	Funkcje wstawiające do strumienia.....	1383
33.17	Ćwiczenia	1385

34 Operacje we/wy na plikach 1390

34.1	Strumień płynący do lub od plików	1390
34.1.1	Otwieranie i zamykanie strumienia	1392
34.2	Błędy w trakcie pracy strumienia	1397
34.2.1	Flagi stanu błędu strumienia	1397
34.2.2	Funkcje do pracy na flagach błędu	1398
34.2.3	Kilka udogodnień dla sprawdzania poprawności	1399
34.2.4	Ustawianie i kasowanie flag błędu strumienia	1400
34.2.5	Trzy plagi, czyli „gotowiec”, jak radzić sobie z błędami	1404
34.3	Przykład programu pracującego na plikach	1408
34.4	Przykład programu zapisującego dane tekstowo i binarnie	1410
34.4.1	Zapis w trybie tekstowym	1414
34.4.2	Odczyt z pliku tekstowego	1415
34.4.3	Zapis danych w plikach binarnych	1417
34.4.4	Odczyt danych z pliku binarnego	1418
34.5	Strumień a technika rzucania wyjątków	1420
34.6	Wybór miejsca czytania lub pisania w pliku	1424
34.6.1	Funkcje składowe informujące o pozycji wskaźników	1425
34.6.2	Wybrane funkcje składowe do pozycjonowania wskaźników	1425
34.7	Pozycjonowanie w przykładzie większego programu	1428
34.8	Tie – harmonijna praca dwóch strumieni	1434
34.9	Ćwiczenia	1436

35 Operacje we/wy na stringach 1439

35.1	Strumień zapisujący do obiektu klasy <i>string</i>	1439
35.1.1	Przykłady ilustrujące użycie klasy <i>ostream</i>	1443
35.2	Strumień czytający z obiektu klasy <i>string</i>	1446
35.2.1	Prosty przykład użycia strumienia <i>istream</i>	1448
35.2.2	Strumień <i>istream</i> a wczytywanie parametrów-danych	1451
35.2.3	Wczytywanie argumentów wywoływania programu	1456
35.3	Ożenek: strumień <i>stringstream</i> czytający i zapisujący do stringu	1460
35.3.1	Przykładowy program posługujący się klasą <i>stringstream</i>	1461
35.4	Ćwiczenia	1465

36 Projektowanie programów orientowanych obiektowo 1467

36.1	Przegląd kilku technik programowania	1467
36.1.1	Programowanie liniowe (linearne)	1468
36.1.2	Programowanie proceduralne (czyli „orientowane funkcyjnie”)	1468
36.1.3	Programowanie z ukrywaniem (zgrupowaniem) danych	1468
36.1.4	Programowanie obiektowe – programowanie bazujące na obiektach	1469
36.1.5	Programowanie obiektowo orientowane (OO)	1469
36.2	O wyższości programowania OO nad Świętami Wielkiej Nocy	1470
36.3	Obiektowo orientowane: projektowanie	1473
36.4	Praktyczne wskazówki dotyczące projektowania programu techniką OO	1474
36.4.1	Rekonesans, czyli rozpoznanie zagadnienia	1475
36.4.2	Faza projektowania	1475
36.4.3	Etap 1. Identyfikacja zachowań systemu	1477
36.4.4	Etap 2. Identyfikacja obiektów (klas obiektów)	1477
36.4.5	Etap 3. Usystematyzowanie klas obiektów	1479
36.4.6	Etap 4. Określenie wzajemnych zależności klas	1480
36.4.7	Etap 5. Składanie modelu. Sekwencje działań obiektów i cykle życiowe	1482
36.5	Faza implementacji	1483
36.6	Przykład projektowania	1483

36.7	Rozpoznanie naszego zagadnienia	1484
36.8	Projektowanie	1488
36.8.1	Etap 1. Identyfikacja zachowań naszego systemu	1488
36.8.2	Etap 2. Identyfikacja klas obiektów, z którymi mamy do czynienia	1489
36.8.3	Etap 3. Usystematyzowanie klas obiektów z naszego systemu	1492
36.8.4	Etap 4. Określamy wzajemne zależności klas	1494
36.8.5	Etap 5. Składamy model naszego systemu	1496
36.9	Implementacja modelu naszego systemu	1501

37 Szablony – programowanie uogólnione1509

37.1	Definiowanie szablonu klas	1510
37.2	Prosty program z szablonem klas	1512
37.2.1	Ostrożnie z referencją jako parametrem aktualnym	1514
37.3	Szablon do produkcji funkcji	1515
37.4	Cudów nie ma. Sorry	1519
37.5	Jak rozmieszczać w plikach szablony klas?	1520
37.6	Tylko dla orłów	1521
37.7	Szablony klas, drugie starcie	1521
37.8	Co może być parametrem szablonu – zwiastun	1522
37.9	Rozbudowany przykład z szablonem klas	1522
37.9.1	Definiowanie funkcji składowych szablonu klas	1527
37.9.2	Składniki statyczne w szablonie klasy	1528
37.9.3	Obiekt klasy szablonej tworzony operatorem <i>new</i>	1530
37.9.4	Dyrektywa <i>using</i> składnikiem szablonu klas	1531
37.9.5	Przeładowany operator << w szablonie klas	1533
37.9.6	Jawne wywołanie destruktoru klasy szablonej	1534
37.10	Reguła SFINAE	1535
37.11	Kiedy kompilator sięga po nasz szablon klas?	1539
37.12	Co może być parametrem szablonu? Szczegóły	1540
37.13	Parametry domniemane	1549
37.13.1	Szablon klas z domniemanymi parametrami	1549
37.13.2	Domniemane parametry w szablonie funkcji	1550
37.14	Zagnieżdżenie a szablony	1552
37.14.1	Szablon funkcji składowych zagnieżdżony w szablonie klasy	1553
37.14.2	Szablon klasy zagnieżdżony w zwykłej klasie	1559
37.14.3	Szablon klasy z zagnieżdżoną definicją klasy	1561
37.15	Poradnik: jak pisać deklaracje przyjaźni w świecie szablonów	1563
37.15.1	Szablon obdarza przyjaźnią swój parametr	1569
37.16	Użytkownik sam może specjalizować szablony klas	1570
37.16.1	Kompletna (zupełna) specjalizacja szablonu klasy	1573
37.16.2	Częściowa specjalizacja szablonu klasy	1575
37.16.3	Częściowa specjalizacja pozwala wybrać parametry będące wskaźnikami	1577
37.17	Specjalizacja funkcji składowej szablonu klas	1581
37.18	Specjalizacja użytkownika szablonu funkcji	1583
37.19	Cwiczenia	1585

38 Posłowie1591

38.1	Per C++ ad astra	1591
------	------------------------	------

A Dodatek: Systemy liczenia1593

A.1	Dlaczego komputer nie liczy tak jak my?	1593
A.2	System szesnastkowy (heksadecymalny)	1599
A.3	Cwiczenia	1601

Skorowidz.....	1603
-----------------------	-------------



0 Proszę tego nie czytać!

0.1 Zaprzyjaźnijmy się!

Jeśli mamy ze sobą spędzić parę godzin, to proponuję – przejdźmy na „ty”. Moje nazwisko zobaczyłeś już na okładce, dodam więc tylko, że książka ta powstała na podstawie moich doświadczeń jako programisty w niemieckich, francuskich i włoskich instytutach fizyki jądrowej, z którymi mój macierzysty Instytut Fizyki Jądrowej im. Henryka Niewodniczańskiego Polskiej Akademii Nauk (w Krakowie) od lat prowadzi eksperymenty z dziedziny badania struktury jądra atomowego metodami spektroskopowymi. Eksperymenty te wymagają rozbudowanego oprogramowania, które współtworzyłem. Pierwotnie robiłem to w języku C klasycznym, ale gdy pojawił się język C++, nastąpił w tej pracy przełom. Ten niezwykły C++ szybko stał się jednym z najpopularniejszych i najważniejszych języków programowania i tak jest do dziś.

Jestem oczarowany jasnością, prostotą i logiką programowania w języku C++ i dlatego proponuję Ci spędzenie ze mną paru chwil w krainie programowania orientowanego obiektowo.

Będzie to niezwykła podróż. C++ ciągle się udoskonala i jest bardzo profesjonalny. Stosowany jest tam, gdzie konieczna jest duża wydajność. Najczęściej używa się go do tworzenia gier komputerowych, do obliczeń naukowych, technicznych, w medycynie, w przemyśle, w bankowości... Na lądzie, na wodzie, w powietrzu i w przestrzeni kosmicznej. Nie wierzysz?

Możliwe, że nawet nie zdajesz sobie sprawy, ile razy korzystałeś z efektów pracy języka C++. Znałe Ci zapewne programy Adobe Photoshop, Illustrator, Acrobat napisano przy użyciu języka C++. Komputerowe efekty w filmach „Gwiezdne wojny”, „Władca Pierścieni”, „Spider-Man” i innych wykonane zostały przez programy napisane w C++. W C++ napisano najistotniejsze procedury sterowania samolotami F-16 czy F-35. Niemal całe oprogramowanie sterowania silnikami okrętowymi olbrzymich kontenerowców i tankowców firmy MAN B&W Diesel A/S zostało napisane w C++.

NASA – amerykańska agencja lotów kosmicznych – posługuje się językiem C++ w wielu swych przedsięwzięciach. Zarówno w naziemnej kontroli lotów kosmicznych, jak i tam, w przestrzeni kosmicznej. Duża część oprogramowania w Międzynarodowej Stacji Kosmicznej została napisana w C++. Na orbicie okołoziemskiej krążą teleskopy kosmiczne, które potrafią swym zasięgiem penetrować tak odległe części

Wszechświata, że możemy obserwować najdawniejsze fazy jego powstania. I do tego użyto oprogramowania napisanego w C++.

W grafice na okładce tej książki nawiązuję właśnie do tego faktu.

Naprawdę więc nie tylko „na lądzie, wodzie i w powietrzu”. Nawet na powierzchni planety Mars. W marsjańskim łaziku Curiosity pracuje program w C++ analizujący obraz z kamer i planujący dalszą trasę łazika.

Taka jest terażniejszość języka C++; a przyszłość? Kto wie, może leży właśnie w Twoich rękach? Wiem, co mówię. Otrzymałem od czytelników moich książek wiele listów informujących, jak niezwykle potoczyły się ich dalsze, programistyczne losy. Teraz czas na Ciebie...

Książka ta jest następcą książki „Symfonia C++ Standard”

Na rynku księgarskim przez wiele lat obecna była moja książka „Symfonia C++ Standard”. Miała ok. 1200 stron, wiele wydań i cieszyła się ogromną poczytnością. Gdy pojawił się standard języka zwany C++11, należało napisać nową książkę. To właśnie efekt tej pracy.

Przy pisaniu tej nowej książki korzystałem oczywiście z niektórych fragmentów tamtej, a stary tekst musiał zostać zmodyfikowany i adaptowany do nowego standardu. Nowy standard wprowadził wiele nowych, ciekawych rozwiązań i należało je także w przystępny sposób opisać. W rezultacie obecna książka ma o 500 stron więcej. To nie jest już nowe wydanie „Symfonii”. To nowa książka. Ma też całkiem nowy tytuł. Nie chciałem popełnić tego samego błędu po raz drugi. Pierwsza moja książka miała tytuł „Symfonia C++”, a druga – tytuł dość podobny: „Symfonia C++ Standard”. W rezultacie wielu osobom, które chciały kupić tę nową książkę na aukcjach internetowych, sprzedawano tę starą, nadającą się wówczas już tylko na makulaturę.

Dlatego teraz postanowiłem zmienić tytuł tak wyraziście, żeby nie narażać czytelników na wspomniane nieporozumienia.

Dlaczego C++11, a nie C++17

Jeśli choć trochę interesujesz się językiem C++, to pewnie wiesz, że w chwili gdy obecne, drugie wydanie *Opusu* idzie do druku, powstały już standardy C++14 i C++17. Mógłbyś zapytać, dlaczego ta książka omawia standard C++11, a nie od razu C++14 lub C++17. Odpowiedź jest prosta: gdy pracowałem nad pierwszym wydaniem tej książki, mój kompilator ostrzegał mnie, że implementacja tych nowszych standardów jest w nim jeszcze „eksperymentalna i niepełna”. Nie mogłem więc w odpowiedzialny sposób kompilować w nim programów w standardzie C++17 ani tym bardziej opowiedzieć Ci o różnych aspektach danej nowej cechy, jeśli sam nie mogłem tego grunto-wnie sprawdzić. Z C++14 i C++17 musiałem więc wtedy jeszcze zaczekać. Z drugiej strony nie był to wielki problem, bo różnice między C++11, C++14 i C++17 są raczej niewielkie. Natomiast przejście ze starego standardu C++03 do C++11 było bardzo znaczące. Twórca tego języka, Bjarne Stroustrup, napisał wtedy: „**C++11 to jakby nowy język**”.

W końcu jednak przyszedł czas na omówienie C++14 i C++17. Napisałem więc dodatkowe rozdziały omawiające nowe cechy języka wprowadzone przez standardy C++14 i C++17.

Pierwotnie myślałem o rozszerzeniu tej książki, ale gdy zdałem sobie sprawę, że *Opus* jest i tak dość obszerny, postanowiłem wydać te zagadnienia w postaci odrębnej książki.

Powstał więc jakby osobny, czwarty tom uzupełniający tę książkę. Ma on tytuł *Opus magnum C++*. **Misja w nadprzestrzeń C++ 14/17.**

Jeśli spodoba Ci się mój sposób opowiadania o C++, to poszukaj *Misji w nadprzestrzeń* w księgarniach. Jest ona jakby ciągiem dalszym tej książki.

Wersja języka

Absolutnym autorytetem w sprawie C++11 był dla mnie dokument standardu języka C++ (ISO International Standard ISO/IEC 14882:2011 – Programming Language C++). W razie jakichkolwiek niejasności odsyłam do tego tekstu. (Uprzedzam jednak, że jest napisany w bardzo sformalizowanym stylu i nie nadaje się jako podręcznik do nauki).

Oczywiście nawet w C++11 można pisać programy „po staremu”. Można pisać nawet programy niemające z techniką orientowaną obiektowo nic wspólnego. Tylko że to tak, jakby zwiedzać Granadę, mając zamknięte oczy¹⁾.

Dlaczego ta książka jest taka gruba?

Nie wynika to wcale z tego, by język C++ był tak trudny. Uznałem tylko, że to, co uczy naprawdę, to przykłady. W książce więc oprócz „teorii” są dziesiątki przykładowych programów, a każdy jest szczegółowo omówiony. Przykładowi towarzyszą wydruki pokazujące wygląd ekranu po wykonaniu programu. To wszystko właśnie sprawia, że książka jest tak obszerna. Jednak wydruki te załączam w przeświadczeniu, że często łatwiej zorientować się, „co program robi”, rzucając okiem na taki obraz ekranu. Dopiero potem radzę analizować sam program.

Pióro

Ktoś kiedyś powiedział, że są dwa style pisania – pierwszy to: „popatrzcie, jaki ja jestem mądry”, a drugi to: „popatrzcie, jakie to proste”. Ja w tej książce wybieram ten drugi wariant w przeświadczeniu, że prędzej zaprowadzi do celu. Z drugiej strony wiem, że bezpośredni, wręcz kolokwialny styl tej książki, zupełnie naturalny w książkach na Zachodzie, w Polsce może zaskoczyć niektórych czytelników.

Dla kogo jest ta książka

Książka ta jest napisana z myślą o czytelnikach, którym nieobce są takie pojęcia jak program, instrukcja, dyskietka, monitor. Oczywiście najlepiej by było, gdybyś znał już jakiś inny, choćby najprostszy język programowania. Żebyś wiedział, co to jest komputer, dysk, program, instrukcja. Jeśli tak nie jest, to – jak sądzę – wystarczy Ci 20-minutowa rozmowa z kolegą, który już programuje, lub przeczytanie jakiejś strony internetowej tłumaczącej, co to znaczy programować komputer.

Pisząc tę książkę, musiałem najpierw określić sobie, dla kogo będzie ona przeznaczona. Otóż jest ona dla tzw. szerokiego grona czytelników. Pisałem ją tak, by była podręcznikiem dla 14-letniego programisty amatora, jak i dla zawodowego

1) „...kobieto, daj jałmużnę ślepcowi, bo nie ma większego nieszczęścia, niż być ślepcem w Granadzie...”

informatyka. Trudno te sprawy pogodzić, więc są w niej kompromisy w obu kierunkach. Co do jednego z takich kompromisów mam największej obaw:

Czołem, bracia angliści!

Gdy słyszałem, jak początkujący programiści wymawiają występujące w języku C++ angielskie słowa takie jak „unsigned”, „volatile”, „width”, postanowiłem w miejscach, gdzie słowa te pojawiają się po raz pierwszy, zamieścić adnotacje o ich wymowie. Wymowa podana jest nie w transkrypcji fonetycznej, ale za pomocą polskich liter. Wiem, że fakt ten może razić wielu czytelników. Proszę wtedy o wyrozumiałość – pamiętajcie, że ta książka ma służyć również małuczkiemu. Wymowa podana jest dyskretnie – w przypisie, w nawiasie, a w dodatku jeszcze w cudzysłowie – więc nie trzeba jej koniecznie czytać. Spodziewam się, że fakt zamieszczenia wymowy nie będzie przeszkadzał czytelnikom, którzy językiem angielskim posługują się na co dzień od lat, choć być może wzbudzi protest tych, którzy angielskiego nauczyli się przedwczoraj.

Od czasu, gdy pojawiła się pierwsza „Symfonia C++”, sytuacja ze znajomością języka angielskiego wśród naszych rodaków bardzo się zmieniła. W Krakowie nawet pani w kiosku na mojej ulicy mówi po angielsku. Wydawałoby się więc, że teraz można by już usunąć te przypisy o wymowie angielskich słów. No, ale przecież Kraków – z tysiącami zagranicznych turystów – nie jest w tym przypadku reprezentatywny. Chciałbym, żeby ta książka służyła także bardzo młodym ludziom w różnych, nawet odległych zakątkach Polski. Oni też mają prawo wzbudzić się do swego lotu do gwiazd.

A swoją drogą, to nawet wśród moich kolegów fizyków i programistów nie spotkałem poprawnie wymawiających słowo: „*width*”.

A teraz o strukturze tej książki

Książkę można podzielić na dwie zasadnicze części – tę klasyczną, opisującą zwykłe narzędzia programowania, i drugą (zaczynającą się od rozdziału o klasach) opisującą narzędzia do programowania obiektowo orientowanego. Po tym wszystkim następuje rozdział omawiający operacje wejścia/wyjścia, czyli sposoby pracy z takimi urządzeniami zewnętrznymi jak klawiatura, ekran, dyski magnetyczne. Wreszcie następuje rozdział o projektowaniu programu obiektowo orientowanego zawierający szczegółowy instruktaż. Uznałem te sprawy za bardzo ważne, gdyż często programista, mając w ręce to doskonałe narzędzie programowania, jakim jest C++, nie wie, co z nim począć. Na koniec zostawiłem rozdział o programowaniu uogólnionym (czyli innym niż orientowane obiektowo).

Książka ta opisuje sam język C++. Pierwotnie odczuwałem pokusę, aby opowiedzieć także o klasach z biblioteki standardowej C++. Gdy jednak zobaczyłem, jak bardzo książka się rozrasta, zdecydowałem, że to jednak temat na osobną książkę. Ta i tak jest bardzo gruba. Ograniczyłem się więc tutaj do omówienia potrzebnych Ci w pierwszej fazie nauki bezwzględnie najważniejszych klas bibliotecznych: wektorów, stringów i oczywiście strumieni wejścia/wyjścia. Są one opisane bardzo obszernie.

Metodą kolejnych przybliżeń

Nie liczę na to, że czytając tę książkę, zrozumiesz wszystko od razu. Lubię powtarzać, że mądrość polega między innymi na umiejętności odróżnienia spraw ważnych od mniej ważnych. Ucząc się nowego języka programowania, nie daj się zasypać szczegó-

łami. Dlatego w tekście wielokrotnie sugeruję, żebyś przy pierwszym czytaniu książki opuścił niektóre paragrafy. Nie wszystkie niuanse danego zagadnienia są ważne już na samym początku nauki. Lepiej najpierw mieć ogólny pogląd, a dopiero potem zagłębiać się w trudniejsze szczegóły. Te bardziej szczegółowe fragmenty poprzedza zwykle znaczek



Te miejsca opatrzyłem także adnotacją: „dla wtajemniczonych”. Są tam uwagi wybiegające nieco do przodu, ale rozumieją je czytelnicy, którzy mają za sobą już pierwsze czytanie – czyli czytelnicy już „wtajemniczeni”.

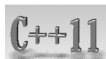
Nie szanuj tej książki. Przygotuj sobie kolorowe (tzw. niekryjące) flamastry (tzw. markery) i czytając, zakreślaj ważne dla Ciebie wyrazy czy myśli. Na marginesach pisz ołówkiem swoje uwagi i komentarze. Chociażby miały to być teksty typu: „Co za bzdura!” albo „Nie rozumiem, dlaczego...”, albo „Porównaj dwie strony wcześniej”. Ta aktywność zaprocentuje Ci natychmiast, bo mając tak osobisty stosunek do tekstu, łatwiej koncentrować na nim uwagę – co jest warunkiem *sine qua non* szybkiej nauki.

Jeśli znasz język C „klasyczny” (czyli tzw. ANSI C)

to zapewne pierwsze rozdziały będą dla Ciebie wyraźnie za łatwe. Mimo to radzę je przejrzeć chociaż pobieżnie, gdyż występują pewne różnice w językach C i C++ – nawet na tym etapie (oczywiście na korzyść C++).

Gorąco natomiast zachęcam do uważnego przeczytania rozdziałów o wskaźnikach. Z doświadczenia wiem, że te sprawy zna się zwykle najgorzej. Tymczasem zarówno w klasycznym C, jak i w C++ wskaźnik jest bardzo ważnym i pożytecznym narzędziem. Dlatego to zagadnienie tak rozbudowałem.

Znaki graficzne, piktogramy, numerki



Jeśli programowałeś w dawniejszych wersjach języka C++, a tę książkę czytasz, żeby poznać nowości w C++11, to zwróć uwagę na paragrafy, w których na marginesie pojawia się taki znak jak ten, który widzisz tu z lewej.



Czasem, gdy nowinka dotyczy tylko jednej linijki, ma on taką skróconą formę. Na marginesie występują też i inne znaczki, ale nie ma sensu o nich pisać, bo ich znaczenie będzie dla Ciebie oczywiste. Natomiast w tekstach programów spotkasz miejsca oznaczone czarnymi numerkami.

- 23 Są one po to, żeby potem łatwo było nam te fragmenty programu omawiać. Dodatkowo są też numerki drugiego typu. Występują one we fragmentach, gdzie pokazany jest wygląd ekranu komputera, na którym nasz program wypisał jakiś tekst. Właśnie tam (czasami) pojawiają się numerki „kwadratowe” 23. Jak widać, liczba jest ta sama, ale teraz jest ona na kwadratowym tle, które (umówmy się) oznacza ekran komputera. W ten sposób oznaczam, że dany tekst jest ekranowym rezultatem pracy instrukcji programu wcześniej oznaczonej jako 23.

Tych kwadratowych numerków nie stosuję często, bo zwykle dokładnie wiadomo, z której instrukcji pochodzi dany tekst. Czasem jednak obecność takiego kwadratowego numerka się nam przyda.

W programach występują nazwy zmiennych. Celowo wymyślam dla nich polskie nazwy, bo wtedy wyraźnie widać, co w programie jest naszą twórczością, a co stanowią

(angielskie) słowa kluczowe języka C++. Zatem w naszych programach będziesz mógł na przykład spotkać się ze zmiennymi o nazwach przelicznik, moc_dopuszczalna, dzwiek. Oczywiście ten ostatni wyraz to dźwięk, ale tych polskich liter nie wolno mi użyć w programie. Nazwy zmiennych mają składać się z liter alfabetu angielskiego. Dlatego trzeba jednak pisać trochę dziwnie: dzwiek. Wkrótce się z tym oswoisz.

Z drugiej strony jednak, gdy potem rozmawiamy o programie, to pozwalam sobie tych polskich liter tam używać, bo w polskim zdaniu wygląda to czytelniej. Zatem piszę, że „do funkcji fun wysyłamy argument dźwięk”. W tych opisach pozwalam sobie nawet na coś więcej: pozwalam sobie na fleksję, czyli końcówki odmiany przez przypadki. Na przykład piszę, że: „inkrementujemy moc_dopuszczalną” lub że „do dźwięku przypisujemy wartość 330”.

Pamiętaj, że te zabiegi spolszczające wolno mi robić poza programem, w jego opisie. Natomiast w samym programie nazwa zmiennej musi być zawsze bez tych polskich liter – i zawsze w mianowniku.

Szukajcie, a znajdziecie

Na koniec uwaga zecerska. W tekście książki czasem występują przypisy u dołu strony. Jeśli – z przyczyn typograficznych – jakiś przypis nie będzie mógł się pojawić u dołu strony, należy go szukać na stronie następnej.

Internetowe wsparcie

Oczywiście na pewno niejedno da się w tej książce poprawić. Jeśli będziesz miał jakieś uwagi, to proszę, przyślij mi je pocztą elektroniczną na adres:

jerzy.grebosz@ifj.edu.pl

Z góry wyrażam moją wdzięczność za nawet najdrobniejsze uwagi. Proszę, napisz nawet o zauważonych błędach literowych czy swoich odczuciach, że jakiś fragment jest niejasny albo stanowczo za trudny.

Kod źródłowy przykładowych programów z tej książki można sobie pobrać ze strony WWW wydawnictwa, a także z mojej strony WWW w Internecie. Obecny adres mojej strony to:

<https://www.ifj.edu.pl/private/grebosz/>

Nawet jeśli ten adres się kiedyś zmieni, łatwo będzie znaleźć nową lokalizację za pomocą internetowej wyszukiwarki, podając hasło „Opus Magnum C++11” lub moje nazwisko.

Na mojej stronie WWW możesz też szukać odpowiedzi do ćwiczeń, które są zamieszczone na końcu poszczególnych rozdziałów tej książki. Na stronach WWW zawierających materiały uzupełniające do tej książki znajdziesz też ewentualną erratę tworzoną na bieżąco z uwag nadsyłanych przez czytelników.

Uwaga do wydania drugiego

Obecne wydanie **drugie** różni się tym od wydania pierwszego, że do tekstu wprowadzono poprawki zgromadzone dotychczas w erracie do wydania pierwszego.

Możesz ją znaleźć na stronie: <https://www.ifj.edu.pl/private/grebosz/opus.html>

Na koniec wypada mi się wytłumaczyć z tytułu tego wstępu. Chciałem tu poruszyć parę ważnych, ale i trochę nudnych spraw. Wiedząc, że czytelnicy najczęściej opuszczają wstępy, dałem taki tytuł, podejrzewając, że zakazany owoc najlepiej smakuje.



Jeśli jesteś moim dawnym przyjacielem, czytelnikiem moich poprzednich książek, to pewnie zapytasz, dlaczego tak długo musiałeś czekać na tę nową książkę. Poprzednią napisałem przecież ponad 10 lat temu.

Sprawa jest przykra. Pisanie książki to dwa lata pracy. Bycie autorem to rodzaj samotności. Pamiętam, jak wiele propozycji wypraw w góry z przyjaciółmi musiałem odrzucać, bo obowiązek nakazywał mi siedzieć i pracować nad kolejnymi rozdziałami. W końcu jednak praca się kończy, książka idzie do wydawnictwa, a potem do czytelników. To jest dla mnie ogromna radość. Nagle jednak okazuje się, że jakiś mój przyjaciel-czytelnik postanowił „spiratować” książkę i umieścić ją w Internecie. Gdy zwracam się do administratorów danej strony, którzy za pieniądze pozwalają ludziom ściągać tę moją pracę – śmieją mi się w twarz. Usuwają ten link, a za dwa dni znowu on wraca. Znowu zwracam się do nich, a oni znowu się śmieją: „I co nam zrobisz?”. W takiej sytuacji nie chce się pisać następnych książek. Też byś tego nie robił.

A jednak napisałem tę, którą trzymasz teraz w dłoniach. Dlaczego? Dlatego że coś się zaczęło zmieniać. Ludzie stali się świadomi, że okradanie polskich autorów sprawia, iż polskich książek jest mniej i są gorszej jakości. Zatem na dłuższą metę tracą inni czytelnicy. Stała się rzecz niezwykła: ostatniej *Symfonii Standard* nie „spiratowano”. Na różnych forach internetowych widziałem nawet, jakie gromy rzucano na tych, którzy domagali się pirackich kopii. Może więc nie wszystko stracone? Postanowiłem zaryzykować. Zaryzykować nowe spotkanie z nowymi (i starymi) przyjaciółmi-czytelnikami.

A zatem wyruszamy razem w wielką przygodę, do gwiazd. Proszę zapiąć pasy, bo...





18

Deklaracje przyjaźni

Funkcja zaprzyjaźniona z klasą to funkcja, która – mimo że nie jest składnikiem tej klasy – ma dostęp do jej wszystkich (nawet prywatnych) składników.

18.1 Przyjaciele w życiu i w C++

Wyobraź sobie taką sytuację. W Twoim domu jest dużo roślin. Rośliny te są prywatnym składnikiem obiektu klasy `dom`. Pewnego dnia wyjeżdżasz na wakacje na Majorkę. Chcesz jednak, by kwiatki Ci nie „zdechły”. Masz dwa wyjścia:

- ❖ Ewentualność pierwsza: sprawić, by kwiatki stały się publiczne, czyli wystawić je na klatkę schodową (kwiatki globalne). Każdy wtedy może wykonać na nich funkcję „podlewanie”. Ryzykujesz jednak, że ktoś nieproszony wykona na nich funkcję „modyfikacja”, czyli przerobi je na pokarm dla swojego królika czy węża boa. Wyjście – jeśli kochasz swoje kwiatki – nie jest dobre.
- ❖ Ewentualność druga: masz zaufanego przyjaciela. Dajesz mu klucze do swojego mieszkania i prosisz go, by podlewał kwiatki. Przyjaciel ma dostęp do wszystkich Twoich prywatnych składników (np. brylantowej kolii w komodzie), ale ponieważ mu ufasz, więc nie boisz się o nic. Przyjaciel przychodzi co drugi dzień i podlewa. Jak dotąd obrazek jest idylliczny.

Wścibscy sąsiedzi zauważają jednak, że ktoś obcy wchodzi do Twojego domu i dzwoni na policję, która pewnego popołudnia urządza zasadzkę – wykopuje przed drzwiami trzymetrowy dół i nakrywa go gałęziami. Przyjaciel wpada w zasadzkę. Policja zaciera ręce, że złapała złodzieja. Co prawda przyjaciel krzyczy coś z dna dołu o przyjaźni, ale nikt go nie słucha. I słusznie, prawdziwy złodziej krzyczałby to samo. Nadjeżdża specjalnym wozem nadinspektor. Ocenia sytuację i mówi:

„Chwileczkę: oto mam w ręce definicję klasy pod tytułem `Dom_Czytelnika` i widzę, że na liście składników jest deklaracja, iż pana `X` uznaje się za przyjaciela `Domu_Czytelnika`. Ma on zatem prawo zrobić wszystko ze składnikami tej klasy. Proszę go więc wyciągnąć z dołu, bo jeszcze kwiatki zwiędną”.

Przełożmy ten obrazek na język pojęć C++

Konstruując klasę, ustalamy, że pewne składniki będą prywatne. Mogą więc na nich pracować funkcje składowe tej klasy. Inne nie.



W pewnych sytuacjach jednak może być korzystne, by jakaś funkcja spoza zakresu tej klasy miała także dostęp do składników prywatnych. Robi się to bardzo prosto. Wewnątrz definicji klasy wystarczy umieścić deklarację tej funkcji poprzedzoną słowem *friend*¹⁾. Dzięki temu zwykła funkcja ma prawo dostępu do prywatnych składników klasy. To tak, jakby składniki te stały się dla niej publiczne.

Ważne jest, że to nie funkcja ma twierdzić, iż jest zaprzyjaźniona. To klasa ma zadeklarować, że przyjaźni się z funkcją i tym samym nadaje jej prawo dostępu do swoich składników prywatnych. Zatem słowo *friend* pojawia się tylko wewnątrz definicji klasy.

Funkcja zaprzyjaźniona ma oczywiście także na mocy tej przyjaźni dostęp do składników *protected*.

Przykład deklaracji przyjaźni z funkcją:

Oto klasa *Tpionek*, w której jest deklaracja przyjaźni z funkcją *raport*:

```
class Tpionek
{
private:
    int kolor, pozycja;
    // dotychczasowe deklaracje
    // .....
    friend void raport(Tpionek );
};
```

Sama funkcja jest gdzieś w programie zdefiniowana następująco:

```
void raport (Tpionek p)
{
    cout << p.kolor << " pionek jest na pozycji " << p.pozycja << endl;
}
```

Funkcja *raport* wywoływana jest z argumentem typu *Tpionek*. Najważniejsze jest, że:

Wewnątrz tej zaprzyjaźnionej funkcji *raport* możemy odwoływać się do prywatnych składników obiektu klasy *Tpionek*. To tak, jakby te składniki były publiczne. To wszystko.

Jeśli chcemy, by funkcja wypisała dane o jakimś *Tpionku*, to po prostu wysyłamy go jako argument funkcji.

```
Tpionek niebieski;    // definicja obiektu
...
raport(niebieski);    // wywołanie funkcji
```

Może Ci się nasunąć pytanie: „Skoro chcemy, by funkcja pracowała na danych składowych klasy, to dlaczego nie zrobić z niej po prostu funkcji składowej tej klasy?”.

Pochwalam ten pomysł. Tak właśnie powinno być w tym przypadku. Lepiej mieć funkcję jako składnik, bo łatwiej wtedy nad nią panować . Tak samo jak lepiej, by to domownik podlewał kwiatki, niż przychodził w tym celu ktoś obcy.

Funkcje zaprzyjaźnione mają pewne cechy, które je wyróżniają i czynią z nich bardzo dobre narzędzie

Najważniejsza cecha to:

- 1) ang. *friend* – przyjaciel [czytaj: „*frend*”]

Dzięki deklaracji przyjaźni możemy nadać dostęp do prywatnych składników naszej klasy nawet takiej funkcji, która nie mogłaby być funkcją składową naszej klasy z powodów zasadniczych.

Na przykład dlatego, że:

- jest już funkcją składową innej klasy
- albo musi być funkcją globalną (np. przeładowany operator <<).

Funkcja może być przyjacielem więcej niż jednej klasy. (Tak się dzieje, gdy więcej niż jedna klasa zadeklaruje z nią przyjaźń). Wtedy taka funkcja może mieć dostęp do prywatnych składników *kilku* klas.

18.2 Przykład: dwie klasy deklarują przyjaźń z tą samą funkcją

W programie mamy dwie klasy. Klasa Tpunkt opisuje współrzędne jakiegoś punktu. Klasa Tkwadrat opisuje współrzędne lewego dolnego rogu kwadratu i długość jego boku. W obu są deklaracje przyjaźni z funkcją globalną o nazwie sedzia.



```
#include <iostream>
#include <string>
using namespace std;
//-----
class Tkwadrat; // deklaracja zapowiadająca ❶
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
class Tpunkt
{
    int x, y;
    string nazwa;
public:
    Tpunkt(int a, int b, string opis);
    void ruch(int n, int m)
    {
        x += n;
        y += m;
    }
    // ...może coś jeszcze...

    friend int sedzia(Tpunkt & p, Tkwadrat & k); ❷
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
class Tkwadrat
{
    int x, y;
    int bok;
    string nazwa;
public:
    Tkwadrat(int a, int b, int dd, string opis);
    // ...może coś jeszcze...

    friend int sedzia (Tpunkt & p, Tkwadrat & k); ❸
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
Tpunkt::Tpunkt(int a, int b, string opis) // konstruktor
```

```

{
    x = a;
    y = b;
    nazwa = opis;
}
//*****
Tkwadrat::Tkwadrat(int a, int b, int dd, string opis)    // konstruktor
{
    x = a;
    y = b;
    bok = dd;
    nazwa = opis;
}
//*****
// Z tą funkcją przyjaźnią się obie klasy.
int sedzia (Tpunkt & pt, Tkwadrat & kw)
{
    if( (pt.x >= kw.x) && (pt.x <= (kw.x + kw.bok) )
        &&
        (pt.y >= kw.y) && (pt.y <= (kw.y + kw.bok) )
        )
    {
        cout << pt.nazwa << " leży na tle " << kw.nazwa << endl;
        return 1;
    }
    else {
        cout << "AUT! " << pt.nazwa << " jest na zewnątrz " << kw.nazwa << endl;
        return 0;
    }
}
//*****
int main()
{
    Tkwadrat    bo(10, 10, 40, "boiska");
    Tpunkt      pi(20, 20, "pilka");

    sedzia(pi, bo );
    cout << "kopiemy pilke!\n";
    while(sedzia(pi, bo))
    {
        pi.ruch(20,20);
    }
}

```

4

5



Po wykonaniu programu na ekranie zobaczymy:

```

pilka leży na tle boiska
kopiemy pilke!
pilka leży na tle boiska
pilka leży na tle boiska
AUT! pilka jest na zewnątrz boiska

```



Przyjrzyjmy się ciekawszym miejscom programu

- Wewnątrz definicji klasy Tpunkt widzimy deklarację przyjaźni. Klasa Tpunkt stwierdza tutaj, że ma zaufanie do funkcji sedzia.

Bardzo ważna uwaga:

Zauważ, że na liście argumentów tej funkcji jest nazwa klasy Tkwadrat. Do tej pory klasa ta jeszcze nie została zdefiniowana. Pamiętamy jednak, że w C++ każda nazwa, zanim zostanie użyta po raz pierwszy, musi zostać zadeklarowana.



Jak ten problem rozwiązać?

- ❶ Oto rozwiązanie. Jest to tak zwana **deklaracja zapowiadająca** (zwiastująca). Mówi ona: „Jakby co, to nazwa Tkwadrat jest nazwą klasy”. To wszystko. Nie ma tu nic więcej na temat wewnętrznej struktury klasy Tkwadrat, ale to nie szkodzi, bo w momencie deklaracji przyjaźni te detale nie są jeszcze kompilatorowi potrzebne.
- ❸ To deklaracja przyjaźni w drugiej klasie. Podobnie: klasa Tkwadrat stwierdza tutaj, że ma zaufanie do funkcji sędzia.
- ❹ Oto definicja funkcji sędzia. Jest zdefiniowana jak najzwyklejsza funkcja.

Różnica polega tylko na tym, że funkcja ta pracuje sobie na prywatnych składnikach obiektów obu klas – tak jakby były one publiczne.

Czym zajmuje się funkcja sędzia? Łatwo się zorientować, że po prostu sprawdza, czy obiekt klasy Tpunkt leży na tle obiektu klasy Tkwadrat („czy piłka leży na boisku”). Robi to przez porównanie współrzędnych punktu z obszarem zajmowanym przez obiekt klasy Tkwadrat.

- ❺ W funkcji main korzystamy z tej funkcji. Zdefiniowaliśmy dwa obiekty i wywołujemy funkcję sędzia. Zauważ, że obiekty te wysyłamy do funkcji jako zwykłe argumenty – nie ma tu żadnego zapisu w stylu

obiekt.funkcja()

bowiem funkcja sędzia nie jest funkcją składową żadnej klasy.



18.3 W przyjaźni trzeba pamiętać o kilku sprawach

Funkcja jest zaprzyjaźniona z klasą, a nie tylko z jakimś konkretnym obiektem danej klasy. To znaczy, że funkcja zaprzyjaźniona otrzymuje prawa przyjaciela w stosunku do **wszystkich** obiektów tej klasy.

- ✧ Deklaracja przyjaźni tylko deklaruje przyjaźń – i nic więcej. Konkretnie: nazwa funkcji zaprzyjaźnionej nie staje się przez to nazwą z zakresu tej klasy.

Po prostu w deklaracji przyjaźni tylko rozmawiamy z kompilatorem, tłumacząc mu, że taka funkcja ma dostęp...

- ✧ Funkcja zaprzyjaźniona nie jest składnikiem klasy, dlatego nie ma wskaźnika `this` do obiektów klasy, która obdarza ją przyjaźnią.

Dla nas oznacza to, że jeśli chcemy w ciele tej funkcji odnieść się do składnika jakiegoś obiektu klasy, która uznaje nas za przyjaciela, musimy powiedzieć:

- ❖ jak ten obiekt się nazywa (wtedy posługujemy się składnią *obiekt.składnik*)
- ❖ lub pokazać na niego wskaźnikiem (wtedy posługujemy się składnią *wskaźnik->składnik*).



Powtarzam więc wniosek:

Funkcja zaprzyjaźniona to zwykła funkcja, której wyjątkowo nie obowiązują słowa *private* i *protected* w klasach uznających ją za przyjaciela.

- ✧ Zwykle wewnątrz klasy funkcja zaprzyjaźniona jest tylko deklarowana. Jest to jedynie deklaracja przyjaźni. Nie ma znaczenia, w którym miejscu klasy (*public*, *protected*, *private*) taka deklaracja nastąpiła. Słowa *public*, *protected* i *private* nie mają na to wpływu. Przyjacielem albo się jest, albo nie jest.
- ✧ Może się tak zdarzyć, że kompilator (pracując nad jakimś plikiem) zobaczy deklaracje pewnej funkcji po raz pierwszy dopiero w miejscu deklaracji przyjaźni. Nie jest to błąd, ale uwaga: w tym miejscu kompilator uzna, że chodzi o jakąś funkcję globalną, dostępną ogólnie – także z innych plików tego programu.
Jeśli jednak zostanie przez nas oszukany, czyli gdzieś dalej zobaczy, że definiujemy tę funkcję jako funkcję *static* (a więc widzialną tylko dla jednego konkretnego pliku), zasygnalizuje błąd.
Nie byłoby problemu, gdybyśmy wcześniej zamieścili deklarację tej funkcji jako *static*, bo wtedy przy deklaracji przyjaźni kompilator już wiedziałby, z czym ma do czynienia, i nie musiałby niczego zakładać w ciemno, a potem zmieniać zdania.

Jak to zrobić w naszym niedawnym programie?

U nas w punkcie ② w deklaracji przyjaźni po raz pierwszy pojawia się nazwa funkcji sędzia, nieznana jeszcze kompilatorowi. Skoro nie było jeszcze deklaracji tej funkcji, kompilator zakłada, że chodzi o jakąś funkcję sędzia z zakresu globalnego.

Lepszą praktyką jest jednak deklarowanie wszystkich funkcji, które potem mają wystąpić w deklaracji przyjaźni.

U nas polegałoby to na postawieniu deklaracji funkcji sędzia pod liniijką ①. Niestety w funkcji sędzia jeden z argumentów jest typu *Tpunkt*, a także ten typ jest tu jeszcze kompilatorowi nieznan. Rozwiązanie jest proste: i on powinien mieć deklarację zapowiadającą. Łącznie więc: w programie, w miejscu ①, dobrze by było mieć takie deklaracje:

```
class Tkwadrat;           // deklaracja zapowiadająca ①
class Tpunkt;           // deklaracja zapowiadająca
int sedzia (Tpunkt & pt, Tkwadrat & kw); // deklaracja funkcji
```

Wiele kompilatorów nie wymaga surowo wcześniejszych deklaracji funkcji, które mają zostać potem obwołane przyjaciółmi, i wybaczy nam ich brak. No ale według standardu C++ te deklaracje powinny jednak być.

Przyjaciół-rezydent, czyli: funkcja zaprzyjaźniona „goszcząca” w klasie

Możemy tak zrobić, że wewnątrz klasy jest nie tylko deklaracja funkcji zaprzyjaźnionej, ale wręcz jej definicja (czyli całe ciało funkcji). Mimo że jest ona umieszczona „w środku” ciała klasy, funkcja jest nadal tylko przyjacielem, a nie składnikiem.

Taka definicja funkcji zaprzyjaźnionej ma następujące konsekwencje:

- ❖ funkcja zaprzyjaźniona jest typu *inline*,
- ❖ funkcja leży w zakresie **leksykalnym** deklaracji tej klasy; oznacza to, że można w definicji tej zaprzyjaźnionej funkcji:

18.4 Obdarzenie przyjaźnią funkcji składowej innej klasy

Funkcja zaprzyjaźniona może być zwykłą funkcją, a może być też funkcją składową zupełnie innej klasy.

Oto tak zmodyfikowany poprzedni przykład, że funkcja sędzia jest składnikiem klasy Tkwadrat, a klasa Tpunkt deklaruje z tą funkcją przyjaźń.



```

#include <iostream>
#include <string>
using namespace std;
//-----
class Tpunkt;                // deklaracja zapowiadająca           ❶
/////////////////////////////////////////////////////////////////
class Tkwadrat
{
    int x, y;
    int bok;
    string nazwa;
public:
    Tkwadrat(int a, int b, int dd, string opis);
    // ...może coś jeszcze...

    int sędzia (Tpunkt & p);                                     ❷
};
/////////////////////////////////////////////////////////////////
class Tpunkt
{
    int x, y;
    string nazwa;
public:
    Tpunkt(int a, int b, string opis);
    void ruch(int n, int m) {
        x += n;
        y += m;
    }
    // ...może coś jeszcze...

    friend int Tkwadrat::sędzia(Tpunkt & p);                    ❸
};
/////////////////////////////////////////////////////////////////
Tpunkt::Tpunkt(int a, int b, string opis)                        // konstruktor
{
    x = a;
    y = b;
    nazwa = opis;
}
//*****
Tkwadrat::Tkwadrat(int a, int b, int dd, string opis)          // konstruktor
{
    x = a;
    y = b;
}
    
```

```

    bok = dd;
    nazwa = opis;
}
//*****
int Tkwadrat::sedzia (Tpunkt & pt)           ❷
{
    if( (pt.x >= x) && (pt.x <= (x + bok) )   ❸
        &&
        (pt.y >= y) && (pt.y <= (y + bok) )
    )
    {
        cout << pt.nazwa << " lezy na tle " << nazwa << endl;
        return 1;
    }
    else {
        cout << "AUT!" << pt.nazwa << " jest na zewnatrz " << nazwa << endl;
        return 0;
    }
}
//*****
int main()
{
    Tkwadrat    bo(10,10, 40, "boiska");
    Tpunkt      pi( 20, 20, "piłka");
    bo.sedzia(pi);                               ❹
}

```



Po wykonaniu programu na ekranie pojawi się:

piłka lezy na tle boiska



Komentarz

- ❷ To jest deklaracja zwykłej funkcji składowej w klasie Tkwadrat. Argumentem jest obiekt klasy Tpunkt, stąd też konieczna była deklaracja zapowiadająca tę klasę ❶.
- ❸ To deklaracja przyjaźni. Tutaj jednak ważna uwaga. Jeśli przyjacielem ma być *funkcja składowa z innej klasy*, to ta klasa musi być już w tym momencie znana kompilatorowi. Dlatego najpierw w programie umieszczona jest deklaracja klasy Tkwadrat (z funkcją sędzia), a potem dopiero deklaracja klasy Tpunkt i niniejsze ogłoszenie przyjaźni.
- ❹ Oto definicja funkcji sędzia. Na pewno już przy deklaracjach zauważyłeś, że zmieniła się lista argumentów. Teraz argumentem jest tylko obiekt klasy Tpunkt. A co z obiektem klasy Tkwadrat, funkcja go przecież także potrzebuje?! Zapominasz, że teraz funkcja jest funkcją składową klasy Tkwadrat, a więc jest wywoływana na rzecz obiektu klasy Tkwadrat. Zresztą spójrz poniżej.
- ❺ Tak właśnie w main wywołujemy funkcję sędzia. Obiekt klasy Tpunkt wysyłany jest jako argument, a obiekt klasy Tkwadrat – przez ukryty wskaźnik this.
- ❻ Z faktu, iż funkcja sędzia jest funkcją składową klasy Tkwadrat, wynika, że odnosząc się do danej składowej swojej klasy, można posługiwać się zapisem: *składnik*, a nie zapisem: *obiekt.składnik* – widać to wyraźnie w tej linijce.
W stosunku do składników obiektu klasy zaprzyjaźnionej Tpunkt stosujemy tu zapis pt.x, ale w stosunku do składników swojej klasy: x, bok. Przedtem w tym miejscu było konieczne kw.x, kw.bok.

To dlatego, że przecież gdy piszemy `x`, to jest tam naprawdę `this->x`.

„*Coś kręcisz!*” – zawołałeś zapewne – „*parę stron wcześniej wmawiałeś mi, że funkcja zaprzyjaźniona z klasą nie zawiera wskaźnika `this!`*”.

Podtrzymuję to!

|| Funkcja `F` zaprzyjaźniona z klasą `K` nie zawiera wskaźnika `this` do klasy `K`, która uznaje ją za przyjaciela, bo nie jest składnikiem tej klasy.

Jeśli jednak sama funkcja `F` jest zwykłą funkcją składową jakiejś innej klasy, to zawiera wskaźnik `this` do obiektu *swojej* klasy. Za jego pomocą pracuje przecież na swoich składnikach.

Posłużmy się analogią do podlewania kwiatków. Załóżmy, że to Ty jesteś osobą podlewającą kwiatki i Twoja znajoma Perfidia zadeklarowała z Tobą przyjaźń.

Gdy określasz swoje czynności, to mówisz: „Idę podlać kwiatki Perfidii”. Jak do tej pory rzeczywistość nie ma wskaźnika `this`. Mówisz przecież o składnikach tych klas `Perfidia.kwiatki`.

Teraz uwaga: okazuje się, Czytelniku, że dostałeś mieszkanie i nie mieszkasz już więcej pod mostem. Nie jesteś już funkcją globalną, tylko należysz do klasy pod nazwą „`mój_dom`”. Załóżmy, że Perfidia deklaruje Cię nadal jako swojego przyjaciela.

Mówisz: podlewam „kwiatki Perfidii” (podlewam `Perfidia.kwiatki`).

Możesz jednak powiedzieć też: „podlewam kwiatki”, myśląc o podlewaniu *swoich* kwiatków

kwiatki czyli `this->kwiatki`

gdzie `this` oznacza wskaźnik do obiektu „`mój_dom`”.



Jeśli projektujesz program i widzisz, że przydałoby się, żeby funkcja miała dostęp do składników prywatnych dwóch klas, to masz do wyboru jedno z rozwiązań:

obie klasy deklarują tę funkcję (globalną) jako zaprzyjaźnioną,
funkcja jest składnikiem jednej klasy, a druga klasa deklaruje ją jako funkcję zaprzyjaźnioną.

Który z wariantów wybrać, decydujesz, rozważając wspomniane zalety funkcji zaprzyjaźnionej z cechami funkcji składowej. O podejmowaniu takich wyborów porozmawiamy jeszcze w jednym z następnych rozdziałów. („Przeładowanie operatorów” – str. 954).

18.5 Klasy zaprzyjaźnione

Klasa `K` może deklarować przyjaźń z więcej niż jedną funkcją składową klasy `M`. Może nawet deklarować przyjaźń ze wszystkimi funkcjami klasy `M`. Jest to trochę tak, jakbyś wytrwale zadeklarował przyjaźń klasy `K` z każdą funkcją składową klasy `M`. Łatwo te deklaracje zrobić, ale wymaga to dużo pisania.



Zamiast tego możemy zadeklarować, że klasa `K` uznaje za przyjaciela *całą* klasę `M`.

```
class K
{
    friend class M;
    // ...
};
```

Od tej pory *wszystkie* funkcje składowe klasy M mają dostęp do prywatnych składników klasy K deklarującej tę przyjaźń. To już Cię nie dziwi – przyzwyczaiłeś się do tego przy okazji funkcji zaprzyjaźnionych. Jest tu jednak coś jeszcze, coś, czego przy funkcjach być nie mogło. Załóżmy, że mamy zwykłą klasę K, która deklaruje przyjaźń z klasą PRZYJACIEL.



Klasa PRZYJACIEL może używać składników prywatnych klasy K nie tylko w swych funkcjach składowych, ale **także przy inicjalizacji swych składników, nawet tych statycznych**. Jak pewnie jeszcze pamiętasz, ich definicje są umieszczane osobno, jakby na zewnątrz definicji klasy PRZYJACIEL.

|| Nawet więc tam, jakby na zewnątrz ciała klasy PRZYJACIEL, przyjaciel może skorzystać z wartości prywatnego składnika klasy K.

✧ Jeśli klasa K ma w sobie jakieś definicje typów enum lub typedef czy using, to klasa PRZYJACIEL też może z nich skorzystać przy deklaracji swoich składników.

✧ Mówiliśmy o tym, że definicję *funkcji* zaprzyjaźnionej z klasą K można umieścić nawet w samym miejscu deklaracji przyjaźni, czyli wewnątrz klasy K (funkcja: przyjaciel – rezydent, str. 699).

|| Jednak z klasą-przyjacielem tej sztuczki zrobić się nie da. Ta klasa-przyjaciel musi mieć swoją definicję gdzieś na zewnątrz.

Deklaracja przyjaźni jest oczywiście jednostronna

Wyraża ją klasa K wobec klasy PRZYJACIEL i już. Natomiast klasa PRZYJACIEL nie wyraża niczego szczególnego w stosunku do klasy K. Konkretnie – wcale jej nie upoważnia do grzebania w swoich składnikach prywatnych.

Dwie klasy mogą się przyjaźnić także z wzajemnością

Jedyną możliwością zadeklarowania takiej przyjaźni jest właśnie deklaracja przyjaźni z klasą jako całością sposobem, jaki pokazaliśmy.

|| Nie ma możliwości zadeklarowania w jednej klasie, że przyjaźni się ona z funkcjami innej klasy, a w tej innej klasie – że przyjaźni się z wybranymi funkcjami klasy pierwszej.

To z powodu, o którym już wspomnieliśmy: jeśli deklarujemy przyjaźń z funkcją, która jest funkcją składową innej klasy, to kompilator życzy sobie już znać deklarację tej klasy (na przykład, żeby sprawdzić, czy taka funkcja rzeczywiście tam jest).

Żebyśmy się nie wiem jak gimnastykowali, to zawsze definicja jednej klasy będzie znana wcześniej od drugiej, bo tak przecież piszemy tekst programu. Siłą rzeczy klasa, która definiowana jest wcześniej, musiałaby zawierać w sobie deklaracje przyjaźni z funkcjami składowymi klasy drugiej, chwilowo jeszcze nieznanymi. (Sama deklaracja zapowiadająca klasę nie wystarcza kompilatorowi – musi on znać wnętrze klasy).

Nie ma problemu. Wyjście z tego błędnego koła załatwia nam deklaracja przyjaźni z całą klasą.

```
class Tdruga;           // deklaracja zapowiadająca
```

```
class Tpierwsza {
    friend class Tdruga;
    // ...reszta ciała klasy pierwszej
```

```
};

class Tdruga {
    friend class Tpierwsza;
    // ...reszta ciała klasy drugiej
};
```

Przyjaźń nie jest przechodnia

|| Przyjaciel mojego przyjaciela nie jest moim przyjacielem.

Inaczej mówiąc: jeśli klasa A deklaruje przyjaźń z klasą B, natomiast klasa B deklaruje przyjaźń z klasą C, to wcale nie oznacza, że klasa A uznaje klasę C za swojego przyjaciela.

Gdyby o to chodziło, to należałoby w klasie A zamieścić deklarację takiej przyjaźni:
friend class C;

Przechodniość przyjaźni byłaby bardzo niebezpieczna. Zresztą w życiu także się nią nie posługujemy.

Przyjaźń nie jest dziedziczna

|| Przyjaciel mojej prababki nie jest moim przyjacielem.

👑 Wtajemniczeni wiedzą, że klasa może mieć „potomstwo” (klasy pochodne). Przyjaźń nie jest dziedziczna – jeśli jakaś klasa chce mieć przyjaciela, to powinna to powiedzieć wyraźnie sama.

W deklaracji przyjaźni nie mogą pojawić się przydomki (specyfikatory)...

✧ ...określające sposób, w jaki przyjaciel został (przez kompilator) umieszczony w pamięci. Te niedozwolone przydomki to static, register, extern, thread_local i mutable.

W życiu codziennym jest podobnie. Wypada powiedzieć: „Przyjaźnię się z Tomaszem”, a nie wypada powiedzieć: „Przyjaźnię się z Tomaszem, bo ma wille z basenem” albo „Przyjaźnię się z Tomaszem, bo mieszka za granicą”.

18.6 Konwencja umieszczania deklaracji przyjaźni w klasie

Spotyka się często konwencję definiowania klasy w taki sposób, że najpierw w definicji klasy wyszczególnia się wszystkie składniki publiczne (czyli widziane z zewnątrz klasy). Dopiero dalej występują składniki prywatne, czyli takie, o których zwykły użytkownik klasy nie musi już wiedzieć. (Używając praktyki automatycznej, użytkownik nie musi wiedzieć o wszystkich jej elementach elektronicznych i mechanicznych).

Funkcje zaprzyjaźnione są tym, co powinno się od razu zauważyć, patrząc na definicję klasy, więc przyjęło się umieszczać je na samym początku, na samej górze definicji klasy.

Jak mówię – jest to tylko konwencja, która może czasem ułatwić „czytanie” definicji klas.

18.7 Kilka otrzeźwiających słów na zakończenie

Poznaliśmy tu nowe narzędzie pozwalające na dostęp do schowanych składników klasy. Nie daj się jednak ponieść. Przyjaźń jest przecież naruszeniem czegoś, z czego


jesteśmy bardzo dumni, czyli schowania części danych w klasie. Schowania po to, żebyśmy w przypadku gdy program „chodzi źle”, nie martwili się: „Któż to zmienił mi wartość tego składnika bez mojej wiedzy...”. Im mniej przyjaciół, tym łatwiej panować nad działaniem danej klasy.

Zatem szalu nie ma, przesadzanie z przyjaźnią jest złą praktyką.

Naprawdę więc deklaracje przyjaźni będziemy stosowali głównie w sytuacjach:

- ❖ gdy będziemy chcieli, aby obiekt cout mógł wypisywać na ekranie treść składników obiektu naszej klasy (obdarzymy wtedy przyjaźnią klasę ostream),
- ❖ gdy będziemy chcieli, żeby na obiektach naszej klasy mogły sprawnie pracować globalne funkcje tzw. operatorowe (poznamy je na str. 954).

18.8 Ćwiczenia

- I** Przyjaźń polega na tym, że dana klasa K udziela zezwolenia innej funkcji/klasie:
- a) na dostęp do pracy z obiektami klasy K,
 - b) na modyfikację jej składników w klasie K,
 - c) na dostęp do składników niepublicznych w obiektach klasy K,
 - d) na dostęp do składników niepublicznych w wyznaczonych obiektach klasy K.
- II** Klasa K oznajmia przyjaźń z funkcją/klasą P. Słowo friend umieszczone jest:
- a) w deklaracji funkcji/klasy P uznanej za przyjaciela,
 - b) w klasie K, która ogłasza przyjaźń z funkcją/klasą P,
 - c) w klasie/funkcji P, w instrukcjach, które odnoszą się do prywatnych składników klasy K.
- III** Przyjaźń może deklarować:
- a) klasa, b) funkcja globalna, c) funkcja składowa, d) każde z wymienionych.
- IV** Co może być przyjacielem?
- a) funkcja globalna, c) funkcja składowa,
 - b) funkcja statyczna, d) klasa.
- V** Jedna funkcja może być przyjacielem
- a) tylko jednej klasy, b) nawet wielu klas.
- VI** Czy w deklaracji przyjaźni wyrażonej w klasie K wobec funkcji f, funkcja ta musi być wcześniej zadeklarowana?
- VII** Jeśli funkcja-przyjaciel klasy K odnosi się do składników klasy K, to skąd wiadomo, którego obiektu klasy K to dotyczy?
- a) działa to na wszystkie obiekty klasy K,
 - b) funkcja musi powiedzieć, o który konkretny obiekt jej chodzi.
- VIII** Przyjaciel klasy K otrzymuje prawo dostępu do składników niepublicznych klasy K
- a) we wszystkich obiektach klasy K,
 - b) w wybranym obiekcie klasy K.
- IX** Jakie ma konsekwencje fakt, że funkcja-przyjaciel klasy K ma swoją definicję dołączoną do deklaracji przyjaźni w klasie K („funkcja-rezydent”)?
- X**  W lokalnej klasie KL umieszczamy deklarację przyjaźni z funkcją f i równocześnie definicję tej funkcji f („funkcja-rezydent”). Czy funkcja f może zwracać rezultat będący obiektem typu KL?

- XI** Klasa K deklaruje przyjaźń z funkcją składową klasy P. Jaka deklaracja klasy K musi poprzedzać definicję tej klasy?
a) wystarczy deklaracja zapowiadająca K, b) konieczna jest definicja klasy K.
- XII** Wybierz wszystkie poprawne zakończenia następującego zdania: Klasa K deklaruje przyjaźń z funkcją składową klasy P; ta funkcja składowa:
a) jest wywoływana na rzecz obiektu klasy K, która obdarzyła ją przyjaźnią,
b) jest wywoływana na rzecz obiektu swojej klasy P,
c) może pracować na składnikach swego obiektu klasy P,
d) może pracować na obiekcie klasy K pod warunkiem, że wie na którym.
- XIII** Klasa K deklaruje przyjaźń z klasą P. Czy funkcja składowa klasy P otrzymuje wskaźnik `this` pozwalający jej pracować na składnikach klasy K?
- XIV** Co to znaczy, że klasa K deklaruje przyjaźń z klasą P?
- XV** Co to znaczy, że przyjaźń nie jest przechodnia?
- XVI** Czy przyjaźń jest wzajemna?
- XVII** Gdzie należy umieszczać deklarację przyjaźni w klasie K. W jej części `public` czy `private`?
- XVIII** Funkcja składowa klasy P ma przydomek `static`. W klasie K chcemy zamieścić deklarację przyjaźni z tą funkcją. Gdzie umieszczamy ten przydomek?
a) po słowie `friend`, a przed typem rezultatu,
b) na samym końcu deklaracji.



Skorowidz

!

'l' operator sumy bitowej 130
 ll operator sumy logicznej 125
 # dyrektywa pusta 269
 > a przeładowanie 957
 ## sklejacz 274
 > a przeładowanie 957
 #define 269-270
 #elif 278
 #else 277
 #endif 276
 #error 279
 #if 276
 #ifdef 278
 #ifndef 279
 #include 280-281
 #line 280
 #pragma 282
 #undef 271
 % (operator modulo) 120
 & operator iloczynu bitowego 130
 & operator pobrania adresu 356
 && operator iloczynu logicznego 125
 \a' 69
 \b' 69
 \f' 69
 \n' 13, 69
 \r' 69
 \t' 69
 \v' 69
 , (przecinek) operator 148
 /* komentarze */ 14
 // komentarze 14
 :: operator zakresu 910
 > a zasłanianie 86
 __STDC__ 284
 __STDC_HOSTED__ 284
 __func__ 284
 __STDC__ 284
 __cplusplus 284
 __DATE__ 283
 __LINE__ 283
 __NAME__ 283

__TIME__ 283

A

abstrakcyjna klasa 1290-1296, 1479
 adjustfield maska, pole 1337
 adres
 > 0 zero (dawniej zwany NULL) 367
 > funkcji 453
 • a jej nazwa 447
 • przeładowanej 487-490
 > funkcji szablonowej 1518
 > nullptr (zerowy) 365
 > tablicy 293
 > zamiana go na liczbę całkowitą 362
 agregat 883, 892
 > klasa 581-582
 > tablica to agregat 581-582
 aktualny
 > argument funkcji 184
 > parametr szablonu 1511
 alarm (znak specjalny) 69
 algorytm
 > w postaci szabł. funkcji 1123
 > zliczający 1114
 alias 102, 463
 alignas 113-114, 139
 alignment 115
 alignof operator 139-140
 > a przeładowanie 957
 alokacja (rezerwacja) tablic 390
 alternatywa (operacja logiczna) 125
 alternatywna
 > dekl. szablonowej f-cji. składowej 1532
 > deklaracja funkcji 181-182, 1532
 ampersand 293
 analiza zachowań obiektów 1476
 anonimowa
 > przestrzeń nazw 223
 > unia 1086-1087
 ANSI C 5
 apostrof 69
 app, tryb otwarcia pliku 1392
 argc 441

argument

- › aktualny funkcji 184
- › będący obiektem 544-546
- › będący tablicą 384
- › będący wskaźnikiem do funkcji 454-457
- › domniemany 199-206
 - funkcji składowej 541
 - kolejne definiowane „na raty” 203
 - a zakres ważności 204
- › formalny funkcji 184
- › formalny, a aktualny 184
- › funkcji
 - będący tablicą 292-295
 - jego nazwa 176
 - referencją l-wartości 190-197
 - referencją r-wartości 192
 - referencją rwartości 195
 - typu initializer_list 808
- › identyczny czy różny dla przeładowania 479-486
- › nienazwany 207
- › przesyłanie przez referencję 185-187
- › przesyłanie przez wartość 184
- › wskaźnikiem do const 386
- › wywołania funkcji 184
- › wywołania programu 440-442, 1456

argumentowość operatora 958

argv 441

arytmetyczny

- › operator 119-123
- › typ 49

ASCII kod 69, 299

at - f. w kl. std::string 622-626

ate, tryb otwarcia pliku 1392

atof - fun. bibl. (Ascii To Float) 443, 633

auto 106-108, 179

- › a const 420-424
- › a gwiazdka wskaźnika 423
- › a wskaźnik 359
- › a const i volatile 253
- › definicja referencji 253-262
- › użyte wobec wyr. lambda 1144
- › we wskaźniku do klasy szabł. 1530
- › a wskaźnik do f. składowej 910
- › a wskaźnik do skł. klasy 900
- › z operatorem new 395

automatyczny obiekt 213

awans 494

B

o babci przypowieść 185

back - f. składowa string:: 627

backslash 69

backspace (znak specjalny) 69

bad - f. sprawdzająca stan strumienia 1399

bad_alloc - klasa wyjątku 409, 411, 1060

bad_cast - klasa wyjątku 1305

badbit (flaga stanu strumienia) 1397-1398

basefield, maska, pole 1364

begin - f. ustawiająca iterator std::stringu 670

begin - funkcja

- › w klasie pojemnika 1531
- › zastosowanie w szabł. 1532

bekslesz 70

beta rozpad 742

bezpośrednia kl. podstawowa 1183

białe znaki 9

biblioteczna funkcja 237-239

biblioteka

- › a przestrzeń nazw 80
- › iostream 1316
- › standardowa 608
- › standardowych strumieni we/wy 1316
- › stdio 1316

binarne wczytywanie 1378

binarny system liczenia 1595

binarny tryb

- › odczytu pliku 1418
- › zapisu pliku 1417

ios::binary, tryb otwarcia pliku 1392

bit 127, 1597

- › najbardziej znaczący 1597
- › najmniej znaczący 1597

bit po bicie 1092, 1193

bitowy operator 127-129

bliższe pokrewieństwo bez znaczenia 1214

blok

- › catch 710
- › funkcji 79
- › instrukcji 23
- › lokalny 78
- › try 709

bool() operator (w strumieniach) 1400

bool, stałe dosłowne tego typu 63

bool, typ 21, 48

boolalpha manipulator 1342

- › zastosowanie 1072

boolalpha, flaga 1337

break 32

„bryk” czyli spis funkcji kl. std::string 679-686

buforowanie strumienia 1320, 1344

błędy pracy strumienia 1397-1407

C

C klasyczny 5

- C-string 71-72, 607, 610
 - › argumentem funkcji 433
 - › bardzo długi 72
 - › długość, a rozmiar 301
 - › jego typ 72
 - › konkatencja 439
 - › w cudzysłowie - (jest jako static) 440
 - › wariacje na temat 433-439
 - › zapis w kilku liniach 72
 - › ze znaków wchar_t 73
- c_str - f. w kl. std::string 648-650
- callable object 1150
- capacity - f. w kl. std::string 617
- carriage return (znak specjalny) 69
- case 32
 - › a constexpr 34
- catch 410-411
 - › a lista inicjalizacyjna konstruktora 793
 - › blok 710
 - › dla listy inicj. konstruktora 792
 - › kolejność stawiania bloków 712
 - › wszystkożerny 727
- cctype (nagłówek) 659
- cecha i wykładnik 1340, 1370
- cechowanie aparatury 839
- CERN ośrodek badań fizyki cząstek 1279
- cerr 1319
- char16_t 47, 609
- char32_t 47, 609
- charakterystyka
 - › konstruktora 1221
 - › w specyfikacji wyjątków 1221
- Church Alonzo 1129
- chwilowy obiekt 546
- ciało
 - › funkcji 8, 175
 - › klasy 504
- cin 17, 1319
- ciąg znaków 71
- class, a typename (szablony) 1516
- clear - f. w kl. std::string 622
- clear - f. w kl. strumienia 1401
- clog 1319
- close, funkcja w kl. strumienia 1395
- compare - funkcja w kl. std::string 651
- const 87
 - › a konstruktor 759
 - › a przeładowanie 580
 - › funkcja składowa 576-579
 - › głębokie 419, 485
 - › obiekt, a wskaźniki 364
 - › obiekty 87
 - › przy wysyłaniu argumentu do funkcji 386
 - › wierzchnie 419, 484
 - › wskaźnik 417
 - › wskaźnik do takiego obiektu 418-419
- const_cast 142, 145, 427-430
 - › a przeładowanie 957
 - › a volatile 431
- constant expression 89
- constexpr 88-91
 - › a lista wyliczeniowa 834
 - › a pola bitowe 833
 - › a switch case 34
 - › funkcja tego typu 240-252
 - › funkcja, a inline 245
 - › konstruktor 825-834
 - › obiekty 88-91
 - › vs. #define 271
- continue 39
- copy - f. w kl. std::string 659
- copy elision 846
- copyfmt, funkcja 1370
- count_if - algorytm biblioteczny 1123
- cout 8, 1319
- covariant 1264
- cyfra 1593
- cykl życiowy obiektu 1482
- czas życia 392
 - › a zakres ważności 78-84
 - › obiektu 773
 - co to jest 78
 - globalnego 772
 - lokalnego 771
 - wybór tego czasu 212-217
 - wytworzonego przez new 772
- czteropak 1526
- czysto wirtualna funkcja 1293, 1295
- czytanie deklaracji 445
- częściowa specjalizacja 1575
 - › a ref. do l-wartości, r-wartości 1580
 - › zastosowanie 1577

D

- dana składowa 505
 - › publiczna - odwołanie się 524
- data() - f. w klasie std::string 647
- debugger 10, 210, 1344, 1409
- dec, flaga 1335-1337
- dec, manipulator 1343, 1352
- decltype 109-110
 - › w alternatywnej dekl. fun. 182
 - › w deklaracji funkcji 182
 - › operator - pełny opis 873-877
 - › typu wsk. f-cji 465-466

- › a wskaźnik do skł. klasy 905
- › a wskaźnik do f. składowej 910
- default
 - › etykieta (w switch) 33
 - › w deklaracji funkcji (=default) 781
- defaultfloat manipulator 1346
- #define
 - › vs. constexpr 271
- defined 276
- definicja 16, 44, 218, 221
 - › a deklaracja 16, 45
 - › funkcji 175
 - składowej szablonu kl. 1527
 - zaprzyjżnionej będąca w klasie 699
 - › klasy 504
 - › klasy zagnieżdżonej 734-740
 - › lokalna klasy 752-754
 - › obiektów klasy string 610-614
 - › po łacinie 45
 - › przeładowanego operatora 956
 - › składnika statycznego 557
 - › stałej za pomocą #define 271
 - › szablonu klas 1510-1511
 - › w warunku
 - instrukcji if 61
 - › w wyrażeniu
 - inicjalizującym pętli for 61
 - warunkowym instr. while 62
 - › wskaźnika 355
 - › „w biegu“ 60-61
- deklaracja 16, 218, 221
 - › a definicja 16, 45
 - › czytanie jej 445
 - › dostępu 1178, 1642
 - › dostępu using 1178
 - › funkcji 175
 - alternatywna 181-182
 - nieobowiązkowa 198
 - › po łacinie 45
 - › przyjaźni 695
 - › typedef 93-95
 - › using 84, 93-95, 1178
 - › zapowiadająca 698, 1209
- dekorator wnętr 758
- dekrementacji operator 121
- delegat 798
- delegowania relacja 1476
- delegowanie łańcuchowe 799
- delegujący konstruktor 794-800
- delete 391
 - › a adres zerowy (nullptr) 777
 - › a możliwy polimorfizm 1075
 - › niemożliwe virtual 1062
- › operator 398
- › przeładowanie 1057-1083
- › standardowe, jego wywołanie 1069
- › tablicy
 - przeładowanie 1075
- › w deklaracji funkcji (=delete) 783
- destrukcja obiektu złożonego 821
- destruktor 552-555, 758-837
 - › a const i volatile 777
 - › a dziedziczenie 1182
 - › a przeładowanie 777
 - › a rzucanie wyjątków 779
 - › a dziedziczenie 1182
 - › jawne wywołanie 778
 - › klasy szabł., jego wywołanie 1534
 - › kolejność ich pracy 821
 - › kolejność wywołania 1186-1191
 - › szablonu klas 1528
 - › wirtualny 1273-1274
 - › wywołanie z this 778
 - › zastosowania 555, 776
- deszyfrowanie słów 1103-1109
- detektor - stoper 743
- Deutsche Oper (ilustr. dziedziczenia prywatnego) 1227
- długość, a rozmiar C-stringu 301
- do... while... 27
- domeny zastosowania wskaźników 369-416
- dominacja klas wirtualnych 1245
- domniemane-
 - › argument 199-206
 - › argument „na raty” 203
 - › argument, a przeładowanie 474
 - › konstruktor 780
 - › parametr szablonu 1549-1551
 - › parametr w szablonie f-cji 1550
 - › parametr w szablonie klas 1549
 - › wartość będąca wyrażeniem lambda 1158-1161
- dopasowanie
 - › a konwersja 946-950
 - › z awansem 494
 - › etapy 492-497
 - › funkcji przeładowanych 491
 - › z promocją 494
- dopasowanie dokładne 492
- Dopplera zjawisko 911
- dorzecze, schemat konwersji 951
- dostęp
 - › a dwuznaczność 949
 - › do skł. odziedziczonych 1173-1180
 - › do składników klasy 508-510
 - › private 509

- › protected 509
- › public 510
- › wybiórczo 1178, 1642

dostępu

- › deklaracja 1178, 1642
- › deklaracja using 1178
- › specyfikator 1171

dosłowne stałe 62-75

dosłowny typ 584

double, typ 48

duża tablica, przykład 1428-1433

dwuargumentowy operator 119

dwuznaczność, a dostęp 949

dwójkowy

- › kod 1597
- › system liczenia 1594

dynamic_cast 142, 146

- › a przeładowanie 957
- › a typy polimorficzne 1303-1305

dynamiczna alokacja (rezerwacja) tablicy 390-412

dynastii koniec 1173

dyrektywa

- › preprocesora 269
- › pusta 269
- › using 83

dzieci z próbowki (in vitro) 1519

dziedziczenie 1170-1252

- › a inicjalizacja 1192-1193
- › a operatory we/wy 1330
- › a przypisanie 1192-1193
- › a wieloznaczność 1212
- › a zawieranie 1222-1223
- › czego się nie dziedziczy 1181-1182
- › graf dziedziczenia 1183
- › jednokrotne 1208
- › kilkupokoleniowe 1183
- › konstruktorów (sposób na) 1215-1221
- › obiektów - nie istnieje 1173
- › od kilku rodziców 1208-1214
- › operatorów 1026
- › prywatne 1227
 - kiedy 1177
 - np. Deutsche Oper 1227
- › prywatne, kiedy? 1178, 1184
- › wielodziedziczenie 1208-1214
- › wielokrotne 1208
- › wielopokoleniowe 1184
- › wielorakie 1208
- › wirtualne 1237-1245
- › zakresów zagnieżdżanie 1171

dzielenie z resztą 120

dziesiątkowy system liczenia 1593

E

early binding 1268

edytor tekstu programu 10

egzotyczne izotopy 741

Tekran_alfanumeryczny

- › klasa przykładowa 760

ekran, urządzenie wyjściowe 1317

eksplozja, schemat konwersji 951

else 22-25

empty() - f. skł. kl. std::string 617

encapsulation 507

end - funkcja

- › iteratora kl. std::string 673
- › w szablonie klasy 1532
 - zastosowanie 1532

endl, manipulator 18, 1344

ends, manipulator 1344

enkapsulacja 1177, 1184, 1481

enum 46, 96-105

- › a przeładowanie 480, 985
- › enum class 97
- › enum class konwersja na int 543
- › tablica takich elementów 296-297
- › w klasie 541
- › w operatorze wej/wyj 1329
- › zwykle, a enum class 103

EOF 1373, 1380

eof() - fun. sprawdzająca stan strumienia 1399

eofbit (flaga stanu strumienia) 1397-1398

erase - f. w kl. std::string 641

etykieta 37

- › (prawdziwa, czyli nie-case, nie-default) 32
- › case 32
- › default 33
- › dostępu 509
- › gdzie jest znana 79
- › private 508-510
- › protected 508-510, 1175
- › public 508-510
- › zakres ważności 79, 212

exception (klasa std::exception) 1420

exception handling 708-733

exceptions, funkcja 1420

EXCLUSIVE OR operator 130

explicit 770, 849

- › a konstr. konwertujący 927
- › konstruktor 770
- › przy operatorze konwersji 941

extensibility (cecha języka C++) 1263, 1471

extern 45, 218, 222

extract we/wyj operator 1320

F

- Fahrenheita skala temperatur 993
- fail() - fun. spr. stan strumienia 1399
- failbit (flaga stanu strumienia) 1397-1398
- failure, klasa ios_base::failure 1420
- false 48, 63
- fałsz 22, 48
- fałsz-prawda 20-21
- fill - f. w klasie strumienia 1350, 1368
- final - słowo kluczowe kontekstowe
 - › jako koniec dziedziczenia klasy 1173
 - › jako koniec ulepszeń f. wirtualnych 1277-1289
- find - f. skł w kl. std::string 635-637
- find_first_not_of 639
- find_first_of 639
- find_last_not_of 639
- find_last_of 639
- fixed, flaga 1336, 1339
- fixed, manipulator 1346
- flaga
 - › skąd określenie 1334
 - › stanu błędu strumienia 1397
 - › stanu formatowania 1334-1340
- flags(fmtflags), f. we/wy 1361, 1365
- float, typ 48
- flush, manipulator 1344
- fmtflags, typ 1362
- for
 - › instrukcja pętli (zwykła) 28-30
 - › zakresowe for 169-171, 627
- for_each - algorytm biblioteczny 1145
- form feed (znak specjalny) 69
- formalny
 - › argument funkcji 184
 - › parametr szablonu 1511
- format operacji we/wy 1334-1340
- formatowanie informacji 1318
- formatowanie wewnętrzne 1439-1464
- free store, (heap) 394, 412, 881
- free()
 - › fun. bibl. do zwalniania pamięci 1066
- front - f. składowa string:: 627
- frytki (jak pola bitowe) 1104
- fundamentalny typ 46, 76
- funkcja 174-268, 1468
 - › adres f. przetwarzanej 487-490
 - › alternatywna deklaracja 181-182
 - › argument aktualny 184
 - › argument będący wskaźnikiem 380-388
 - › argument formalny 184
 - › argument wskaźnikiem do const 386
 - › biblioteczna 237-239
 - › ciało 8
 - › constexpr 240-252
 - › constexpr, a inline 245
 - › czysto wirtualna 1293
 - › deklaracja 175
 - › domniemane przesyłanie obiektów 546
 - › dwa sposoby wysyłania jej tablicy 384
 - › inline 208-211
 - › inline, a szablony 1520
 - › jej blok 79
 - › jej definicja 175
 - › jej nazwa 175, 447
 - › jej sygnatura 1264
 - › jej szablon 1515-1518
 - › konwertująca 927, 935-941
 - › main 8
 - › nawiasy w wywołaniu 447
 - › operatorowa jako przyjaciel 964
 - › operatorowa jako składowa 961-963
 - › orzekająca 456, 1117
 - a obiekt funkcyjny 1126
 - › outline 210
 - › przekazywanie jej tablicy 292-295
 - › przeładowanie jej nazwy 469, 491, 1275-1276
 - › przesyłanie argumentu przez wartość 184
 - › przesyłanie obiektów przez referencję 546
 - › rekurencyjna 228-236
 - › rekurencyjna, a constexpr 248
 - › rezultat referencją l-wartości 223-227
 - › rozmieszczenie tychże w kilku plikach 218-222
 - › składowa 505-506, 513, 516-521
 - a arg. domniemany 541
 - const 576-579
 - const, a constexpr 588
 - constexpr 583-589
 - definiowanie jej 517
 - inline 519
 - specjalizowana 1581-1582
 - specjalna 1035, 1097
 - specjalne 1093
 - static 962
 - statyczna 565-575
 - szablonu klasy 1527
 - volatile 576-579
 - wskaźn. do niej 908-916
 - wywołanie dla obiektu 516
 - wywołanie dla referencji 517
 - wywołanie dla wskaźnika 517
 - wywołanie jej z listy inicjalizacyjnej 790
 - › szablonowa 1516
 - generacja jej (kiedy) 1518
 - › wirtualna 1253-1315, 1496
 - dostęp do niej 1267

- inline 1273
- a przyjaźń 1268
- a static 1268
- w klasie pochodnej 1267
- › wskaźnik do niej 445-468
- › wysłanie do niej elementu tablicy 296, 433, 544-546
- › z wielokropkiem, dopasowanie 498
- › zaciera ślad innej f. wirtualnej 1275
- › zaprzyjaźniona 694-707
 - a wirtualność 1268
 - zdefiniowana w klasie 699
- › zwracanie rezultatu 177-180
- funkcja we/wy
 - › bad() 1399
 - › clear(io_state) 1401
 - › eof() 1399
 - › fail() 1399
 - › fill(char) 1350, 1368
 - › flags(fmtflags) 1365
 - › gcount() 1381
 - › good() 1398
 - › ignore 1379
 - › open(char*, int, int) 1392
 - › peek() 1381
 - › precision(int) 1369
 - › put(char) 1383
 - › rdstate 1401
 - › read(char*, int) 1378
 - › setf 1353
 - › setf(fmtflags) 1360-1365
 - › tellg() 1425
 - › tellp() 1425
 - › unsetf 1353
 - › unsetf(fmtflags) 1360-1365
 - › width(int) 1347, 1367
 - › write(const char*, int) 1384
- funkcyjny
 - › obiekt 1119
 - a lambda 1115
 - a przeładowanie operatora() 1042
- funktor (czyli obiekt funkcyjny) 1042, 1120

G

- garbage collector 1058
- gcount, funkcja 1381
- generowany
 - › bo przypisek =default 781
 - › destruktor 778
 - › f-cji skł. nie będzie gdy przypisek =delete 783
 - › konstruktor 581
 - › konstruktor domniemany 780

- › konstruktor kl. pochodnej 1194
- › konstruktor kopiujący 839, 848
- › konstruktor przenoszący 868
- › mechanizm kopiowania 1035
- › operator przypisania 1001, 1012, 1026
- get from 1320
- getline
 - › a std::string 661-667
 - › pułapka 664
- gl-wartość 870-872
- głębokie const 485
- globalne
 - › nazwa 80
 - › obiekt 212
 - › obiekt, a klasa 528
 - › obiekt, jego inicjalizacja 217
 - › wskaźnik 365
 - › zmienna 1468
- good() - f. spr. stan strumienia 1398
- goodbit (flaga stanu strumienia) 1397
- goto 37-38, 79, 1468
- gotowiec - jak radzić sobie z błędami 1404
- graf
 - › dziedziczenia 1183
 - › współpracy klas 1480, 1494
- greg, gregis 581
- gwiazdolat - metafora wskaźnika 359
- głęboka kopia 855
- głębokie const 419

H

- heap 394
- heksadecymalny system liczenia 1599-1600
- hex, flaga 1335-1337
- hex, manipulator 1343, 1352
- hexfloat, manipulator 1346
- hierarchia 1185, 1228, 1476, 1479, 1493
- hybrydowość 1470

I

- identyfikacja
 - › obiektów (klas obiektów) 1477
 - › zachowań systemu 1476-1477
- if - instrukcja sterująca 22-25
- ignore, f. strumienia 667, 1379
- iloczyn logiczny 125
- implantacja jonu 750
- implantacja jądra atomowego 742
- in - tryb otwarcia pliku 1392
- include 280-281
- inicjalizacja 59, 1014
 - › a klasy podst. wirtualne 1241

- › agregatowa 581
- › definicja 88
- › konstruktorem 551
- › konwersja w niej 945
- › obiektu 548
- › przy dziedziczeniu 1192-1193
- › std::stringu 612
- › tablicy 291
- › tablicy obiektów jakiejś klasy 883-889
- › unii 1086
- › w klasie 513-515
- › w klasie, a konstruktor 551
- › w new
 - klamry { } i nawiasy () 396
- › wektora wielowymiarowego 326
- › zbiorcza 884
- inicjalizator
 - › klamrowy 581
 - › kopiujący 838
 - › przenoszący 863-869
- inicjalizatorów lista 614
- inicjalizujące wyrażenie 359
- initializer_list
 - › a vector 810
 - › argumentem funkcji 808
 - › co to jest 801-815
 - › w inicjalizacji wektora 892
 - › zastosow. w szabl. funkcji 1557
- inkrementacji operator 121
- inline 208-211
 - › a makrodefinicja, porównanie 272
 - › a wirtualność 1273
 - › funkcja składowa 519
 - › gdzie definiować 210
 - › takż przyjaciel 1329
 - › vs. makrodefinicja 272
- inne języki programowania - linkowanie 476
- insert - f. w kl. std::string 642-643
- instrukcja
 - › blok tychże 23
 - › kroku pętli 28
 - › składana 23
 - › sterująca 20-43
 - › throw 710
 - › warunkowa
 - a w nim definicja obiektu 61
- int, typ 46
- int16_t 55
- int32_t 55
- int64_t 55
- int8_t 55
- int_fast16_t 57
- int_fast32_t 57
- int_fast64_t 57
- int_fast8_t 57
- int_least16_t 57
- int_least32_t 57
- int_least64_t 57
- int_least8_t 57
- integer (ang.) 46
- interface (interfejs) 1100, 1598
- internal, flaga 1335-1336
- internal, manipulator 1347
- invalid_argument (wyjątek) 631, 724
- ios:: zamiast ios_base:: 1363
- ios::app 1392, 1394
- ios::ate 1392, 1394
- ios::badbit (flaga stanu strumienia) 1397
- ios::basefield, maska, pole 1364
- ios::beg 1425
- ios::binary 1392, 1394
- ios::cur 1425
- ios::dec 1335-1336
- ios::end 1425
- ios::eofbit (flaga stanu strumienia) 1397
- ios::failbit (flaga stanu strumienia) 1397
- ios::fixed 1335-1336
- ios::goodbit (flaga stanu strumienia) 1397
- ios::hex 1335-1336
- ios::in 1392-1393
- ios::internal 1335-1336
- ios::left 1335-1336
- ios::nocreate (przestarzałe) 1396
- ios::noreplace (przestarzałe) 1396
- ios::oct 1335-1336
- ios::out 1392-1393
- ios::right 1335-1336
- ios::scientific 1335-1336
- ios::seek_dir 1425
- ios::showbase 1335-1336
- ios::showpoint 1335-1336
- ios::showpos 1335-1336
- ios::skipws 1335-1336
- ios::stdio 1335-1336
- ios::trunc 1392, 1394
- ios::unitbuf 1335-1336
- ios::uppercase 1335-1336
- ios_base::failure 1420
- iostream biblioteka 1317
- isdigit, f. biblioteczna 1382
- istringstream 1446-1459
- iteracja 669
- iterator
 - › do obiektu stałego 672
 - › harcerska definicja 1531
 - › stringu 668-677

izotopy 741

J

jawna konwersja

- › jej zapis 945
- › typy 142

jawne wywołanie konstruktora 773

jednoargumentowy operator 121

jest rodzajem...

- › powód dziedziczenia 1222

justowanie 1336

język obiektowo orientowany 1467

K

kalibracja 840

kapsułowanie 507

karty modelujące 1477

kaskadowe przypisywanie 1021

kasowanie tablicy w zapasie pamięci 883

kdevelop 12, 540

kilkupokoleniowe dziedziczenie 1183

klamra { } pusta 111-112

klamrowa lista inicjalizatorów 1557

klamry

- › bloku instrukcji - jak stawiać 40
- › { } 111-112

klamry inicjalizacyjne

- › tablicy new 398

klasa 503-605

- › a obiekt - różnica 511-512
- › a typ 504
- › abstrakcyjna 1290-1296, 1479
- › agregat 581
- › ciało 504
- › definicja 504
- › dominuje 1245
- › enum w niej zdefiniowane 541
- › literalna 826, 834
- › lokalna 700, 752-754
- › najbardziej pochodna 1242
- › pochodna 1171
- › podstawowa 1171
 - bezpośrednia 1183-1184
 - pośrednia 1183-1184
 - wirtualna 1237-1245
- › polimorficzna 1261
- › prywatna 823
- › rozmieszczenie w plikach 529-543
- › składniki 505
- › składowa 734-740
- › std::string 156-160, 607-693
- › szablonowa 1509, 1511

- destruktora wywołanie 1534

- kiedy powstaje 1539

- klasa podstawową 1540

- synonim jej nazwy 1530

- › trywialna 1307

- › uniopodobna 1088-1089

- › uogólnienie jej 1510

- › uogólniona 1186

- › wskaźnik do jej obiektów 881

- › o zagnieżdżonej definicji 734-740

- › zaprzyjaźniona 703-704

klasyczny C 5

klauzura 1138

klauzurowy obiekt 1138

klawiatura 1317

kod ASCII 69

kod dwójkowy 1597

kod źródłowy programu 11

kod źródłowy przykładów z tej książki 6

kolejka 1186

- › jako szablon 1509

kolejność na liście inicjalizacyjnej 789

komentarze 14

komora drutowa 902

kompilacja warunkowa 275-278

- › w przykładzie 1031

- › zastosowanie 1533

kompilator 11

- › optymalizacja pracy 775

kompletna (zupełna) specjalizacja 1573

komputer steruje pomiarami 960

komórka pamięci adresowana wskaźnikiem 389

konflikt nazw 80

kongregacja 581

kongres 581

koniunkcja 125

konkatenacja stringów 439

konstrukcja obiektu z obiektami składowymi 816-822

konstruktor 547-551, 758-837

- › a const i volatile 759

- › a dziedziczenie 1181

- › a static 759

- › a virtual 759

- › a const i volatile 759

- › constexpr 825-834

- › delegujący 794-800

- › do konwersji 927-934

- › domniemany 780, 1243

- dla elementów tablicy 890

- prywatny 825

- › dwa etapy pracy 786

- › explicit 770

- › generowany automatycznie 868
- › i new 772
- › z argumentem typu initializer_list 811
- › jawne wywołanie 773
- › klasy pochodnej 1187, 1209
- › klasy std::string 612
- › kolejność ich pracy 822
- › kolejność wywołania 1186-1191
- › konwertujący 927-934
 - a explicit 927
 - szablonowy 1557
- › kopiujący 838, 1014
 - dla obiektów const 847
 - generowany automatycznie 848
 - klasy pochodnej 1194-1207
 - niejawne wywołanie 839
 - użyty niejawnie 839
- › na cudzej liście inicjalizacyjnej 887
- › niby-wirtualny 1297-1302
- › niepubliczny 823-824
- › prywatny 1520
- › przenoszący 863-869
 - generowany 868
- › przeładowanie nazwy 550
- › przeładowany 758
- › pułapka przy domniemanym 769
- › szablonu klasy 1527
- › wirtualności symulacja 1297-1302
- konstruowanie obiektu złożonego 822
- kontrakt 1481
- konwersja 123, 925-953
 - › a dopasowanie 946-950
 - › argumentu funkcji 1228
 - › argumentów operatora 1228
 - › arytmetyczna 497
 - › dwa warianty 942-943
 - › jawnie 945
 - › kaskadowo 947
 - › konstruktorem 927-934
 - › niejawna 927, 986
 - › niejawnie zachodzi gdy 944
 - › operator tejże 935-941
 - › przy inicjalizacji 1228
 - › referencji (przy dziedziczeniu) 497
 - › referencji do klasy pochodnej 1224-1236
 - › rezultatu funkcji 1228
 - › rzutowaniem 936, 946
 - › standardowa 143, 496
 - przy dziedziczeniu 1224-1236
 - wskaźnika do składnika klasy 1235
 - › trywialna 493
 - › typu całkowitego 496

- › typów zmiennoprzecinkowych 496
- › wskaźnika do klasy pochodnej 1224-1236
- › wskaźników 497
- › wywołanie funkcji 936, 945
- › zawiązująca 60
- konwertujący konstruktor 927-934
- kopia głęboka 855
- kopia płytka 854
- kopiujący
 - › konstruktor 838
 - › operator przypisania 1020
- kowariant 1264
- końcówkowy operator 981
- kryterium
 - › oceniania 1114-1127
 - › orzekające 1114
 - › w obiekcie funkcyjnym 1119
 - › z parametrem 1121
- król Midas (metafora kopiowania) 1024
- kwalifikator zakresu 541, 735, 1172
- kwalifikowana nazwa 735
 - › składnika 1179
 - › zależna 1532

L

- l-wartość 623, 1038
 - › co to jest 188-189
 - › referencja do niej 190-197
- łączność 151, 958
- lambda (wyrażenie)
 - › a noexcept 1139
 - › a rekurencja 1154-1157
 - › a słowo auto 1144
 - › a wskaźnik this 1139-1142
 - › bez nazwy (beziemienne) 1146
 - › ciało 1134
 - › jako domniemana wartość arg. 1158-1161
 - › lista argumentów 1134
 - › mające swą nazwę 1143-1150
 - › nazwane, mające nazwę 1146
 - › rzucające wyjątek 1162-1165
 - › typ rezultatu 1135
 - › w obiekcie 1146
- Lambda rachunek (Alonzo Church) 1129
- łańcuchowe łączenie delegowania 799
- late binding 1269
- least significant bit 1597
- left, flaga 1335-1336
- left, manipulator 1347
- leksykalny zakres 699, 740
- length - f. string:: 616
- lewostronnie łączny operator 990

liczba atomowa 742
 liczba przeciwna 121
 liczba w tekście stringu 628-629
 liczby zespolone 925, 954-955
 liczenia systemy 1593-1602
 likwidacja obiektu 552, 776
 liniowe (linearne) programowanie 1468
 linker 11, 218, 391, 476, 518
 linkowanie 11, 476

- › szablonów klas 1520
- › z innymi językami 476

 lista

- › inicjalizacyjna 783-793, 1186-1191
 - a lista inicjalizatorów - to co innego 884
 - kolejność wykonywania 789
 - wyw. z niej funkcji składowej 790
- › inicjalizatorów
 - a lista inicjalizacyjna - to co innego 884
 - a na niej wywołania konstruktorów 887
 - klamrowa 614
- › klamrowa (inicjalizatorów) 801-815
- › param. formalnych 1515
- › pochodzenia klasy 1171, 1178, 1208
 - a rodzaj dziedziczenia 1176

 lista wychwytywania

- › wyrażenia lambda 1136

 lista wyliczeniowa 97

- › a constexpr 834

 literalna klasa 834
 literalny typ 573, 584, 834
 literał klasy 826
 logiczny operator 124-126
 lokalne -

- › klasa 700
 - jej definicja 752-754
 - jej zakres leksykalny 753
- › nazwa typu (typedef) 755
- › obiekt
 - a wstępna inicjalizacja 217
 - automatyczny 217, 771
 - statyczny 217, 772
- › zakres ważności 78
- › zmienna 1468

 long double, typ 48
 long int, typ 46
 long long int, typ 46
 long long, typ 46
 long, typ 46
 LSB 1597
 lvalue 360

- › Zob. l-wartość

M

magiczne oko 590
 Magnetyczny Rezonans Jądrowy 646
 main 8

- › brak deklaracji 181, 199
- › brak return 180
- › brak wywołania 181
- › rezultat 180-181

 makrodefinicja 272-273

- › a przeładowanie 273
- › a rekurencja 273
- › a szablon funkcji 274
- › i nawiasy 274
- › rozwinięcie 272
- › vs. funkcja inline 272

 malloc - fun. bibl. alokująca pamięć 1064
 manipulator 1341-1352

- › bezargumentowy 1342
- › boolalpha, zastosowanie 1072
- › dec 1343
- › defaultfloat 1346
- › definiowanie przez użytkownika 1353-1359
- › endl 1344
- › ends 1344
- › flush 765, 1344
- › hex 1343
- › hexfloat 1346
- › oct 1343
- › precision(int) 1350
- › predefiniowany 1342
- › resetiosflags(ftmflags) 1353
- › setbase(int) 1352
- › setfill 1350
- › setiosflags(fmtflags) 1353
- › setw 612, 1347
 - › z argumentem 1347

 martwa referencja 225, 1151-1153
 maszynka do funkcji 1519
 max_size() - f. w kl. std::string 617
 mechanizm

- › kopiowania 838, 870, 1012-1025, 1035
- › obsługi syt. wyjątkowych 708
- › przenoszenia 775
 - w klasie pochodnej 1204
- › specjalizacji szablonu (przez użytkownika) 1570
- › zwrotu rezultatu 839

 member function 522
 memberwise copy 848
 memcpy - f. biblioteczna 1306
 memory allocation 1064

- memset - f. biblioteczna 1109
 - menażeria 481, 483-484, 486-487
 - metoda (inna nazwa f. składowej) 522
 - metoda „odtąd dotąd” 674
 - metoda „płytkiej kopii” 1024
 - Midas, król 1024
 - mile na kilometry - przeliczanie 242
 - mnożenie (szybkie) 133
 - model
 - › składanie 1482
 - › widoczne własności 1483
 - › zachowania 1483
 - modelowanie 1471, 1473
 - modulo operator 120
 - moduły programu 218-222, 529-543
 - › z szablonem klas 1520
 - modyfikator
 - › zob. też: przydomek, specyfikator
 - › const 87
 - › constexpr 88-91
 - › explicit 770
 - › mutable 590
 - › register 92
 - › static 216
 - › volatile 92
 - most significant bit 1597
 - move 678, 859-860
 - › w klasie pochodnej 1206
 - MRJ 646
 - MSB 1597
 - mutable
 - › w klasie 590
 - › w wyrażeniu lambda 1138-1139
- N
- na oślepie strzał 365-366
 - nagłówka strażnik 281, 533
 - nagłówkowy plik 219
 - najbardziej pochodna klasa 1242
 - namespace 80, 222
 - › zob. też: przestrzeń nazw
 - › a przeładowanie 479
 - › przykład tworzenia swojego 719
 - › zakres 80
 - napis czyli stała tekstowa 71
 - naukowa notacja 1338
 - nawias pusty w deklaracji funkcji 176
 - nawiasy ostre 142, 1513
 - nazwa
 - › argumentu funkcji 176
 - › funkcji 175, 447, 453
 - › kl. szablonowej 1540
 - › konflikt 80
 - › nie jest obiektem 377
 - › obiektu 392
 - › statyczna globalna 222
 - › szablonu kl. unikalna 1521
 - › tablicy 293, 882
 - › tablicy, a wskaźnik 375
 - › typu, lokalna 755
 - › w C++ 15
 - › w funkcji, zakres ważności 212
 - › w klasie, zakres ważności 506
 - › zasłanianie 85-86, 525-528
 - nazwane obiekty lambda 1146
 - negacji operator ! 126
 - new 391
 - › a tablica wielowymiarowa 398
 - › auto (razem z) 395
 - › bad_alloc 411
 - › i konstruktor 772
 - › a inicjalizacja 395
 - › nadanie obiektowi wartości w momencie stworzenia 395
 - › niemożliwe virtual 1062
 - › nothrow 409
 - a adres nullptr 1076
 - › przeładowanie 1057-1083
 - › standardowe, jego wywołanie 1069
 - › tworzenie obiektu stałego 396
 - › umiejscawiający (operator) 401, 1072
 - › wstępna zawartość tak stworzonego obiektu 392
 - › wyjątku rzucenie 409
 - new line (znak specjalny) 13, 69
 - niebuforowany strumień 1320
 - nieformatowane operacje we/wy 1370-1371
 - niejawna konwersja 986
 - niekompletny typ 739
 - nienazwany argument 207
 - nieprostokątny wektor 2D 335-336
 - nieprostopadłościenny wektor 3D 347-350
 - nieruchomy wskaźnik 417, 419
 - nieskończona pętla 29
 - noboolalpha, manipulator 1342
 - norecreate, dawny tryb otwarcia 1396
 - noexcept 137
 - › a przeładowanie 957
 - › operator 729-731
 - › specyfikator 729-731
 - › w operatorze delete 1062
 - noreplace, dawny tryb otwarcia 1396
 - noshowbase, manipulator 1345
 - noshowpoint, manipulator 1345
 - noshowpos, manipulator 1345
 - noskipws, manipulator 1345

notacja

- › dziesiętna, flaga 1339
- › naukowa (wykładnicza) 1323, 1338
- › wykładnicza, „naukowa“, flaga 1339
- › wykładnicza, „naukowa“, wczytywanie 1323

nothrow 409

- › obiekt typu nothrow_t 1062
- › użyte w new 409

nothrow_t typ obiektu 1062

nounitbuf, manipulator 1345

nouppercase, manipulator 1346

nowa linia 13

NULL adres 68, 367

- › a nullptr 367
- › dawna makrodefinicja 299, 366

null, czyli znak ASCII o kodzie zero 70-71, 299

nullptr 67, 365

- › a NULL 367

nullptr_t 67

numeracja elementów tablicy 289

numeric_limits nazwa typu 51

O

obiekt

- › a klasa - różnica 511-512
- › automatyczny 213
- › automatyczny, a destruktor 777
- › chwilowy 546, 928
- › const 87
- › const, a wskaźniki 364
- › funkcyjny 1115
 - a funkcja orzekająca 1126
 - a przeładowanie operatora() 1042
- › globalny 212, 772
 - jego konstruktor 772
- › inicjalizacja 548
- › klasyfikacja wg. dziedz. lub zawierania 1476
- › konstruowany new 772
- › lambda 1146
- › likwidacja 776
- › lokalny
 - automatyczny 771
 - konstruowanie go 771
 - statyczny 772
- › new, a destruktor 777
- › przesyłany przez wartość 1230
- › składnikiem klasy 816-822
- › składowy bez konstruktora 821
- › static 214
- › statyczny
 - a destruktor 777
 - globalny 222

- lokalny 214

› stały 87

- jego konstruktor 783

› volatile 92

› w środku innego obiektu 816-822

› wytworzony przez new 392

› wywoływalny 1129

obiekt funkcyjny 1119

obiekt-kłauzura 1138

obiektywne programowanie 1469

obiektywo orientowane programowanie 1469-1472

obsługa syt. wyjątkowych 708-733

› rola kompilatora 779

› rola programisty 779

oct, flaga 1335-1337

oct, manipulator 1343, 1352

odczyt pliku

› w binarnym trybie 1415

› w tekstowym trybie 1415

odejmowanie dwóch wskaźników 377

odniesienie się do obiektu 356

„odtąd dotąd” (metoda) 674

off_type, typ w strumieniach we/wy 1425

on flight - definiowanie obiektów 60

open(char*, int, int) 1392

operacja rzutowania 361

operacje wejścia/wyjścia 9, 1316-1389

› błędy 1397-1407

› na plikach 1390-1396

› nieformatowane 1370-1371

operand 128, 954

operandowość tzw. 958, 965

operator 119-155

› ! negacji 126

› "->" 505

› % modulo 120

› % reszta z dzielenia 120

› %= 132

› & (adres) 356

› & | ~ ^ 130

› && oraz || 125

› &= 132

› (), przeładowanie 1040-1045

› * (odniesienie się do obiektu) 356

› *= 132

› += 132

› , (przecinek) 148

› -= 132

› . (kropka) 505

› /= 132

› indeksowania tablicy, przeładowanie 1036-1039

- › ^= 132
- › _Pragma 282
- › alignof 139-140
- › argumentowość tegoż 965
- › arytmetyczny 119-123
- › bitowego iloczynu 130
- › bitowej negacji 130
- › bitowej sumy 130
- › bitowy 127-129
 - a logiczny - porównanie 130-131
- › bool()
 - w strumieniach 1400
- › decltype 873-877
- › dekrementacji 121
- › delete 391
 - a dwukrotnie kasowanie 406
 - odwołanie rezerwacji 398
- › dodawania 954
- › dwuargumentowy 119
- › dwuoperandowy
 - przeładowanie 968-970
- › exclusive or (XOR) 130
- › indeksowania tablicy (w kl. std::string) 622-626
- › inkrementacji 121
- › jako f. składowa, czy globalna 985
- › jako funkcja globalna 967
- › jednoargumentowy (kiedy jest) 121
- › jednoargumentowy +, - 121
- › jego priorytet 148-150
- › jego symbol 954
- › jego łączność 958
- › konwersja w jego obecności 944
- › konwersji 935-941
- › końcówkowy 981
- › logiczny 124-126
 - a bitowy - porównanie 130-131
- › new 391
 - przeładowanie go 1057
- › noexcept 729-731
- › noexcept 137
- › operandowość tegoż 958
- › postdekrementacji 123
- › postinkrementacji 123
- › predefiniowany 964
- › predekrementacji 123
- › preinkrementacji 123
- › priorytet 958
- › przedrostkowy 966
- › przemienność przy przeładowaniu 970
- › przesunięcia bitów w lewo 128
- › przesunięcia bitów w prawo 129
- › przeładowany, definicja 956
 - › przeładowany, lista 956
 - › przypisania 123, 1014, 1026
 - dziedziczenie tegoż 1182
 - klasy pochodnej 1194-1207
 - kopiujący 1012-1025
 - private 1026
 - prywatny 1520
 - przenoszący 1026-1034
 - przenoszący (move) 1026-1034
 - › relacji 124
 - › rzutowania 141-147
 - › różnicy symetrycznej 130
 - › sizeof 135-136
 - › tablicy indeksowania 1429
 - › tworzący typ pochodny 77
 - › typu końcówkowego (postfix) 968
 - › typu postfix 981
 - › wkładania do strumienia 1320, 1325-1333
 - › wyjmowania ze strumienia 1320, 1325-1333
 - › wypisywania - jego przeładowanie 986-991
 - › wywołania funkcji 448
 - › zakresu 86, 910
 - › != 132
 - › łączność operatorów 151
- operatory
 - › których nie przeładujemy 957
- optymalizacja pracy kompilatora 775
- orzekająca funkcja 456, 1117
- ostre nawiasy 142, 280, 1147, 1509, 1511, 1513, 1515, 1522
- ostream 1439-1445
- out, tryb otwarcia pliku 1392
- out_of_range - wyjątek 626, 724
- „outline” funkcja 210
- overloading 469
- override 1275
 - › słowo kluczowe kontekstowe 1277-1289

P

- pamięć komputera 114, 139, 390, 401, 405, 512, 618, 718, 1596
- paradygmat 870
- parametr
 - › aktualny funkcji 184
 - › aktualny szablonu klasy 1511
 - będący referencją 1514
 - › formalny funkcji 184
 - › formalny szablonu 1511
 - › szablonu
 - a constexpr 1541
 - będący adresem fun. globalnej 1546
 - będący adresem obiektu 1546

- będący adresem skł. statycznego 1547
- będący innym szablonem 1547
- będący referencją 1546
- będący stałą całkowitą 1545
- będący wsk. do skł. klasy 1547
- co może być 1540-1548
- domniemany 1549-1551
- szablonem bibliotecznym 1548
- trzy rodzaje 1522
- Pasja C++ 731, 752
- peek, funkcja 1381
- pętla nieskończona 29
- pętla programowa 26
- plain enum 103
- Plain Old Data - Pospolite Stare Dane 1306
- plik
 - › dyskowy 1317
 - › dyskowy, operacje we/wy 1390-1396
 - › nagłówkowy 219
 - › programu
 - z szablonem klas 1520
 - › rozmieszczenie w nich funkcji 218-222
 - › rozmieszczenie w nich klas 529-543
 - › tryby otwarcia do oper. we/wy 1392
- płytką kopia 854
- płytkiej kopii metoda 1024
- pochodzenia klasy lista 1171
- POD - Pospolite Stare dane 1306-1309
- podciąg znaków 633
- podprogram 174
- podstawa systemu liczenia 1338
- podsystemy 1481
- podwalina typu wyliczeniowego 99
- pojemnik na sześć jajek 1526
- pokaż kropkę dziesiątą, flaga 1339
- poła bitowe 356, 1099-1102
 - › a deszyfrowanie słów 1103-1109
 - › szerokość zadana jako constexpr 833
- pole justowania 1337
- polimorficzna klasa 1261
- polimorficzny typ 1304
- polimorfizm 1260-1262, 1469, 1471
 - › anulowany 1213
 - › przy operatorze delete 1075
- pomijanie kopiowania 846
- pomocnik stringowy 725
- pop_back (string::) 641
- porównywanie
 - › stringów (alfabetyczne) 656
 - › wskaźników 379
- pos_type, typ 1425
- postfix operator 968
- postmortem dump 365
- pozycyjny system liczenia 1594
- pośrednia kl. podstawowa 1183
- pr-wartość 870-872
- pragma 282
- prawda - stan logiczny 48
- prawda-fałsz 20-21
- prawostronnie łączny operator 958
- precision, funkcja strumienia 1361, 1369
- predefiniowany
 - › manipulator 1342
 - › strumień 1319
- preprocesor 269-287
 - › dyrektywa 269
- priorytet
 - › operatora 148-150, 958
 - › operatorów we/wy 1324
- private 508-510
- procedura 1468
- proceduralne programowanie 1468
- proces - opisany przez klasę 1042
- program
 - › OO, projektowanie 1467-1508
 - › składający się z kilku plików 218-222
 - › wywołanie go z argumentami 440-442, 1456
 - › zawieszanie się 377
- programowanie
 - › liniowe (linearne) 1468
 - › obiektowe 1469
 - › obiektowo orientowane 1469-1472
 - › proceduralne 1468
 - › uogólnione 1509-1590
 - › z ukrywaniem danych 1468
- projektowanie programu OO 1467-1508
- promocja argumentu 494
- protected 508-510, 1175-1176
- protony 741
- prywatna klasa 823
- przechwytywanie wyjątków 625
- przecinek (operator) 148
- przecinki (dwa obok siebie) 201
- przeładowanie
 - › a enum 985
 - › a przestrzeń nazw 479
 - › a const 580
 - › a przyjaźń 700
 - › a volatile 580
 - › a wirtualność 1275-1276
 - › a zakres ważności nazwy funkcji 477-478
 - › czwórki operatorów 1012-1056
 - › delete 1057-1083
 - › funkcji 469-502
 - › i zasłonięcie równocześnie 529
 - › konstruktora 550
 - › nazw funkcji, a technika OO 475

- › nazw klas jest niemożliwe 1522
- › nazwy funkcji 469-502
 - a argumenty domniemane 474
- › new (operatora) 1057-1083
- › operatora 954-1011
 - () - wywołania funkcji 1040-1045
 - dwuoperandowego 968-970
 - indeksowania tablicy 1036-1039
 - indeksowania tablicy - przykład 1429
 - jednooperandowego 965-967
 - lista możliwych 956
 - new 1057
 - odniesienia się wskaźnikiem 1046-1054
 - ogólna definicja 956
 - postdekrementacji 981-982
 - postinkrementacji 981-982
 - przypisania (kopiującego) 1012-1025
 - przypisania (przenoszącego) 1026-1034
 - wypisywania 986-991
- › operatora wypisywania 986-991
- › zaoszczędzone dzięki konwersjom 928
- przenoszenia mechanizm 775
 - › w klasie pochodnej 1204
- przenoszenie a klasa string 678
- przenoszący
 - › konstruktor 863-869
 - › operator przypisania 1026-1034
- przepis na placek ze... 1517
- przepis-szablon 1510
- przeźrzeń nazw 222
 - › a przeładowanie 479
 - › a zagnieżdżanie klas 736
 - › anonimowa 223
 - › dyrektywa using 83
 - › std:: 239, 609, 1390
 - › stopniowe zaludnianie 82, 719
 - › zakres 80
- przesunięcie bitów
 - › w lewo 128
 - › w prawo 129
- przez wartość przesłanie obiektu 1230
- przydomek
 - › const 87
 - › constexpr 88-91
 - › explicit 770
 - › zob. też: modyfikator, specyfikator
 - › mutable 590
 - › noexcept 729-731
 - › register 92
 - › static 216
 - › volatile 92
- przyjaźń
 - › jej deklaracja 695

- › w świecie szablonów 1563-1569
- przylegające C-stringi 72
- przypisanie 59, 1014
 - › „kaskadowe” 1021
 - › definicja 88
 - › operator 123
 - › przy dziedziczeniu 1192-1193
 - › „bit po bicie” 1193
 - › „składnik po składniku” 1193
- pseudokod 22, 41
- ptrdiff_t 378
- public 508-510
- push_back - f. skł. std::vector 168
- pusta klamra { } 111-112
- put(char) - f. w kl. strumienia 1383
- putback - f. w kl. strumienia 1382

Q

QtCreator 12

R

r-wartość 188-189

- › jej referencja 190-197

rachunek Lambda (Alonzo Church) 1129

radiany 250

raw (surowy) string 74

rdstate, f. sprawdzająca stan strumienia 1401-1402

read(char*, int) 1378

recykling, a konstruktory 856-858

referencja 77, 356

- › a przeładowanie 483
- › argumentem funkcji 185-187
- › definicja jej za pom. auto 253-262
- › do klasy pochodnej 1224-1236
- › jako par. aktualny sz. klas 1514
- › l-wartości 190-197
- › martwa 225, 1151-1153
- › r-wartości 190-197, 856-858

regex 74

register 92

regular expressions 74

reguła SFINAE 1535-1538

reinterpret_cast 142, 147

- › a przeładowanie 957
- › a wskaźniki 360-362
- › przy ustaw. wskaźnika 389

rekurencja 228-236

- › bezpośrednia 229
- › pośrednia 229
- › w funkcji constexpr 249
- › warunek zatrzymujący 233
- › zatrzymanie jej 228

relacja delegowania 1476
 relacji operator 124
 replace - f. w kl. `std::string` 644-646
 reserve - f. w kl. `std::string` 619
`resetiosflags`, manipulator 1353
`resize` - f. w kl. `std::string` 620
 reszta z dzielenia 120
`return` 175, 177-180
 > mechanizm 382
 reusability 1230, 1470, 1479
 rezerwacja obszarów pamięci 390-412
 rezultat zwracany przez funkcję 177-180
`rfind` - f. w kl. `std::string` 638
`right`, flaga 1335-1336
`right`, manipulator 1347
 robot przemysłowy 1598
 rozbudawalność C++ 1471
 rozmiar tablicy (wymagania) 288
 rozmiar unii 1084
 rozmiar, a długość C-stringu 301
 rozmieszczenie klas w plikach 530
 rozmieszczenie szablonów klas w plikach 1520
 rozpad beta 750
 rozpad promieniotwórczy 742
 rozszerzalność C++ 1263, 1471
 rozszerzona dokładność 48
`runtime_error` (klasa wyjątku) 1420
`rvalue` 188-189
 > zob. też: zob. r-wartość
 RVO technika 845
 rzucenie wyjątku przez operator `new` 409
 rzutowanie 141-147
 > `const_cast` 145, 427-430
 > `dynamic_cast` 146, 1303-1305
 > nowymi operatorami 142
 > `reinterpret_cast` 147
 > `static_cast` 143
 > `std::move` 859-860

S

środowisko programowania 539
 scientific
 > flaga 1335-1336, 1339
 > manipulator 1346
 > notation (notacja) 1323
`seek_dir`, typ 1425-1426
`seekg`, funkcja 1425
`seekp`, funkcja 1425
 segment violation 365
 sekwencja działań obiektu 1482
 semantyka przenoszenia tzw. 870
`set_new_handler` 411-412
`setbase`, manipulator 1352

`setf`, funkcja 1353, 1360-1365
`setfill`, manipulator 1350
`setiosflags`, manipulator 1353
`setprecision`, manipulator 1350
`setstate`, funkcja 1401-1402
`setw`, manipulator 1347
 SFINAE reguła 1535-1538
`shared_ptr` - sprytny wskaźnik 1054
`short` 46
`short int`, typ 46
`show positive` 1338
`showbase`, flaga 1335-1336, 1338
`showbase`, manipulator 1345
`showpoint`, flaga 1335-1336, 1339
`showpoint`, manipulator 1345
`showpos`, flaga 1335-1336, 1338
`showpos`, manipulator 1345
`shrink_to_fit` 620
`signed` 46
`silnia` 249
 sinus - rozwinięcie w szereg Taylora 250
`size()` 616
`size_t` 1060
`size_type`, typ w klasie `std::string` 616, 635
`sizeof` operator 135-136
 > a przeladowanie 957
`skip white space` 1336
`skipws`, flaga 1335-1336
`skipws`, manipulator 1336, 1345
 „składnik po składniku” 1193
`sklejacz ##` 274
 składa się z...
 > czyli ma obiekt składowy 1222
 składana instrukcja 23
 składnik
 > będący klasą 734-740
 > będący obiektem innej klasy 506
 > będący typem 734-740
 > `const` w klasie 1026
 > dana 505
 > funkcja 506
 > klasy 505
 > klasy, dostęp 508-510
 > odziedziczony 1171
 > odziedziczony, dostęp 1173-1180
 > referencja w klasie 1026
 > statyczny 556-564
 • definicja tegoż 557
 • deklaracja, a definicja 557
 • inicjalizacja w klasie 570
 • kiedy się przydaje 565
 • trzy sposoby odniesienia się 558
 • w szablonie klas 1528
 • wskaźnik doń 922

- › wariantywny 1089
- › zasłonięty 1172
- składnik po składniku 854, 1193
 - › metoda kopiowania 1013
- słowa kluczowe C++ 16
- słowo 1596
 - › jednostka informacji 127
 - › pamięci 114
- spacje 24, 1322, 1324
- spaghetti kod à la 1468, 1481
- specjalizacja
 - › częściowa fun. skł szabl. klas
 - a ref. l-wartości, r-wartości 1580
 - › częściowa szablonu klas 1575
 - zastosowanie 1577
 - › funkcji składowej szablonu 1581-1582
 - › kompletna (zupełna) 1573
 - › szablonu funkcji
 - zrobiona przez użytkownika 1583-1584
 - › szablonu klas
 - zrobiona przez użytkownika 1570-1580
- specjalizowana klasa szablonowa 1570-1580
- specjalne funkcje składowe 1035
- specjalny znak 69
- specyfikacja wyjątków 731
- specyfikator
 - › zob. też: przydomek, modyfikator
 - › a deklaracja przyjaźni 705
 - › const 87
 - › constexpr 88-91
 - › dostępu 1171
 - › explicit 770
 - a konstruktor konwertujący 927
 - › mutable 590
 - › noexcept 729-731
 - › przestrzeni nazw 84
 - › register 92
 - › signed, unsigned 46
 - › static 216
 - › volatile 92
- spis relacji klas 1480, 1494
- sposób dziedziczenia 1176
- sprytny wskaźnik 1048
 - › shared_ptr 1054
 - › unique_ptr 1054
- środowisko programowania 12
- środowisko programowania 530
- stan niski i stan wysoki 1595
- stan wewnętrzny obj. funkcyjnego 1122
- standardowa biblioteka 237
- standardowa biblioteka strumieni we/wy 1316
- static
 - › a funkcja wirtualna 1268
 - › funkcja 565-575
 - › funkcja składowa 962
 - › modyfikator 216
 - › obiekt 214, 217
 - › składnik w klasie 556-564
 - › specyfikator 216
 - › string 440
- static_assert 137-138
- static_cast 142
 - › a przeładowanie 957
 - › jako funkcja szablonowa 1517
- statyczne-
 - › f. składowa szablonu klasy 1529
 - › funkcja składowa 565-575
 - › obiekt globalny 222
 - › obiekt lokalny 214
 - › składnik
 - inicjalizacja w klasie 570
 - zastosowanie 565
 - › są f. składowe new i delete 1060
- stała
 - › dosłowna 62-75
 - C-string 71
 - całkowita 63
 - definiowana przez użytkownika 992-1007
 - klasy 826
 - typu long long 64
 - zmiennoprzecinkowa 66
 - znakowa 68
 - znakowa typu wchar_t 70
 - › tekstowa 71
 - › za pomocą #define 271
- stały
 - › obiekt 87
 - › wskaźnik 417, 882
- std::bad_alloc 792, 1060
- std::bad_alloc, typ wyjątku 409
- std::count_if - algorytm biblioteczny 1123
- std::exception 792, 1420
- std::flush - manipulator 765
- std::for_each - algorytm biblioteczny 1145
- std::initializer_list 801-815
- std::memcopy - f. biblioteczna
 - › f. biblioteczna 1306
- std::move
 - › funkcja, a przenoszenie 859-860
 - › w klasie pochodnej 1206
- std::nullptr_t 67
- std::string 156-160, 607-693
 - › out_of_range, typ wyjątku 625
 - › alfabetyczne porównywanie 656
 - › at 622-626
 - › begin - f. iteratora 673

- › bryk, czyli zestawienie funkcji 679-686
- › c_str() 648-650
- › capacity 617
- › clear 622
- › compare 652
- › copy 659
- › data() 647
- › definiowanie obj. tej klasy 610-614
- › długość tegoż 616-621
- › dopisywanie do końca 615
- › empty 617
- › end - f. iteratora 673
- › erase 641
- › find 635-637
- › find_first_not_of 639
- › find_first_of 639
- › find_last_not_of 639
- › find_last_of 639
- › fragment tegoż 633
- › inicjalizacja 612
- › insert 642-643
- › iteratory 668-677
- › konstruktory 612
- › length 616
- › liczby wczytanie z niego 630-632
- › liczby wpisanie do niego 628-629
- › liter małych na wielkie zamiana 657-658
- › litery wielkie i małe - a porównywanie 651
- › max_size() 617
- › npos, wartość (string::npos) 635
- › operator indeksowania tablicy 623
- › operatory =,+,+= 615
- › operatory porównywania 656
- › pierwsza wzmianka 156-160
- › pojemność 616-621
- › porównywanie tychże (alfabetyczne) 651-656
- › replace 644-646
- › reserve 619
- › resize 620
- › rfind 638
- › rozmiar 616-621
- › shrink_to_fit 619
- › size 616
- › size_type, typ w klasie string 635
- › substr 634
- › swap 660
- › zamiana na C-string 648-650
- std::transform - algorytm biblioteczny 1163
- stdexcept - plik nagłówkowy wyjątków 626, 633
- stdio
 - › biblioteka 1316
 - › flaga ios:: 1336
- sterow. formatem oper. we/wy 1334
- sterowanie formatem operacji we/wy 1334-1340, 1360-1365
- stod 630
- stof 630
- stoi 630
- stol 630
- stold 630
- stoll 630
- stopnie na radiany 250
- stos 183
- stoul 630
- stoull 630
- stowarzyszony typ 1531
- strażnik nagłówka 281, 533
 - › a #pragma once 282
- strcpy 301, 304, 437
- streamsize, typ 1366
- string
 - › dopisywanie do końca 159
 - › klasa biblioteczna 156-160, 607-693
 - › porównywanie ich 158
 - › a przenoszenie 678
 - › surowy 73
 - › to_string 628
- string::iterator 669
- string::npos 635
- stringstream
 - › opis 1460-1464
 - › przykład użycia 1461
- Stroustrup Bjarne 1512-1513, 1516, 1575
- struktura 580, 1178, 1209
- strumienie, a wyjątki 1420-1423
- strumień 1317-1318
 - › a operator bool() 1400
 - › buforowany 1320
 - › jego ujście 1320
 - › niebuforowany 1320
 - › otwieranie 1392
 - › predefiniowany 1319
 - › predefiniowany, domniemania 1321-1323
 - › tie() - funkcja do powiązania 1434-1435
 - › tryb pracy 1392
- strzał na oślep 365-366
- substr - f. w kl. std::string 634
- substring 633
- suma logiczna 125
- surowe stałe tekstowe 74
- surowy string (stała tekstowa) 73
 - › przykładowy program 725
- swap - f. w kl. std::string 660
- switch 31-32
 - › a constexpr 34

- Sydney Opera 349
 sygnatura funkcji 1264
 symboliczna nazwa typu 1513
 symetria operacji we/wy 1329
 synonim nazwy typu 93
 > klasy szablonowej 1530
 system (część zagadnienia) 1475
 system liczenia 1593-1602
 > dwójkowy 1594
 > dziesiętkowy 1593
 > pozycyjny 1594
 > szesnastkowy 1599-1600
 > wagowy 1594
 sytuacja wyjątkowa obsługa 708-733
 szablon funkcji
 > a funkcje virtual 1556
 > lista parametrów 1515
 > o nazwie static_cast 1517
 > po co? 1515-1518
 > przepis kucharski 1517
 > specjalizacja użytkownika 1583-1584
 > zagnieżdżanie 1552
 szablon klas
 > a specjalizacja użytkownika 1570-1580
 > bardziej specjalizowany 1577
 > definicja 1510-1511
 > deklaracje przyjaźni 1563-1569
 > destruktor 1528
 > funkcja składowa 1527
 > kiedy generuje klasę szablonową 1539
 > konstruktor 1527
 > linkowanie 1520
 > parametr aktualny 1511
 > parametr formalny 1511
 > składniki statyczne 1528
 > specjalizacja częściowa 1575
 > specjalizacja funkcji składowej 1581-1582
 > statyczna funkcja składowa 1529
 > unikalna nazwa 1521
 > using - wykorzystanie 1531
 > zagnieżdżanie 1552
 szablonowa klasa 1509
 szablon 1509-1590
 szeroki znak 47
 szesnastkowy system liczenia
 > system liczenia 1599-1600
 sześciopak 1526
 szybkie mnożenie 133
- T**
- tablica 76, 288-310
 > bardzo duża, przykład 1428-1433
 > dwa sposoby wysyłania jej do funkcji 384
 > dynamiczna alokacja 390, 397
 > element jest l-wartością 190
 > inicjalizacja 291
 > jej adres 293
 > jej elementy 289-290
 > jej nazwa 293, 882
 > new, kasowanie jej 398, 883
 > numeracja elementów 289
 > obiektów
 • definiowana new 881-882
 • inicjalizacja 883-889
 • jakiejś klasy 880-895
 • stałych, jej inicjalizacja 292
 > określanie rozmiaru w definicji 288
 > przekazywanie do funkcji 292-295
 > rezerwowana dynamicznie 390-412
 > rozmiar 296
 > tablic 316
 > tablic, a new 399
 > wielowymiarowa 311-320, 481, 1042
 • a przeladowanie 481
 • a new 398
 • jej typ 315-316
 • wysyłanie do funkcji 317-318
 > wskaźników 431-432
 • do danych składowych 917
 • do funkcji 458-462
 • do funkcji składowych 918-921
 > wymaganie konstr. domniemanego 890
 > wysłanie do funkcji jednego jej elementu 296
 > z czego można tworzyć 289
 > zerowanie klamrami {} 398
 > znakowa 298-305
 tabliczka czekolady (detektor) 741
 tabulator (znak specjalny) 69
 Taylora szereg 250
 technika RVA 845
 technika obiektowo orientowana 1467
 > a przeladowanie nazw funkcji 475
 technika proceduralna 1483
 teksańska masakra 195
 tekst źródłowy programu 11
 tekstowa stała 71
 tekstowy tryb
 > odczytu pliku 1415
 > zapisu pliku 1414
 telegraf maszynowy 99
 tellg() 1425
 tellp() 1425
 template słowo kluczowe 1511
 terminate funkcja biblioteczna 716
 this 522-523

- › dla destruktora 778
- › przykładowe zastosowanie 524
- › typ tego wskaźnika 523, 579

throw 410, 710

tie() - powiązanie strumieni 1434-1435

to_string - funkcja pomocnicza (std::) 159, 628

tolower 659

transform - algorytm biblioteczny 1163

true 48, 63

trunc, tryb otwarcia pliku 1392

try 410-411

- › a lista inicjalizacyjna konstruktora 793
- › blok 709
- › dla listy inicj. konstruktora 792

tryb otwarcia pliku 1392

tryb pracy pliku

- › binarny odczyt 1418
- › binarny zapis 1417
- › tekstowy odczyt 1415
- › tekstowy zapis 1414

trywialna klasa 1307

trywialna konwersja 493

trójwymiarowe wektory 337-339

typ 44-118

- › C-stringu 72
- › arytmetyczny 49
- › definiowany przez użytkownika 503-504
- › dosłowny 584
- › fundamentalny 46-54, 76
- › a klasa 504
- › literalny 573, 834
- › niekompletny 739
- › pochodny 505
- › polimorficzny 1304
- › size_t 1060
- › składowy 734-740
- › stowarzyszony (z szablonem) 1531
- › systematyka 45
- › void 77
- › void* 77
- › wbudowany 46
- › wskaźnika do funkcji 453
- › wyliczeniowy enum 96-105
 - jako indeks tablicy 542
- › zależny
 - w szabl. klasy 1532
- › zdefiniowany przez użytkownika 46
- › złożony 46, 76

typedef 93-95, 946

- › a przeładowanie 480
- › deklaracja 93-95
- › lokalne 755

- › w operatorze wej/wyj 1329

typeid operator

- › a przeładowanie 957

typename

- › a słowo 'class' 1516
- › dodane do typu zależnego 1531
- › słowo kluczowe 1511

U

u16string 609

u32string 609

udostępnianie wybiórcze 1178

uint16_t 56

uint32_t 56

uint64_t 56

uint8_t 56

uint_fast16_t 57

uint_fast32_t 57

uint_fast64_t 57

uint_fast8_t 57

uint_least16_t 57

uint_least32_t 57

uint_least64_t 57

uint_least8_t 57

ujście strumienia 1317, 1320

ukrywanie danych, programowanie 1468

ukrywanie informacji 508-510

układ sprzęgający 1100, 1598

umiejscawiający operator new 401, 1072, 1097

unset, funkcja 1383

unia 1084-1085

- › a deszyfrowanie słów 1103-1109
- › anonimowa 1086-1087
- › inicjalizacja 1086
- › jej wyróżnik 1090
- › z metryczką 1090
- › rozmiar 1084

UNICODE 47

uniopodobna klasa 1088-1089

unique_ptr 792

- › sprytny wskaźnik 1054

unitbuf, flaga 1335-1336, 1341

unitbuf, manipulator 1345

unsetf, funkcja 1353, 1360-1365

unsigned 46

uogólnione

- › klasa 1186, 1510
- › programowanie 1509-1590

uppercase, flaga 1335-1336, 1338

uppercase, manipulator 1346

using 946

- › deklaracja 84, 93-95
- › deklaracja dostępu 1178

- › do typu wskaźnika 463
 - › dyrektywa 83
 - › składnikiem kl. szablonowej 1531
 - › a wskaźnik do skł. klasy 906
- ustawianie wskaźnika 356-358
 UTF-16 47
 UTF-32 47

V

vector

- › a initializer list 810
 - › jego budowa 163
 - › pierwsza wzmianka 161-168
- vertical tabulator (znak specjalny) 69
 virtual
- › a konstruktor 759
 - › funkcja 1253-1315
 - › funkcja, a klasa - różne znaczenie 1258
 - › klasa podstawowa 1237-1245

void 77

void* wskaźnik 363-364

volatile 92

- › a const_cast 431
- › a konstruktor 759
- › a przeładowanie 580
- › funkcja składowa 576-579

VXI (standard urządzeń) 1104

W

- w klasie inicjalizacja 513-515
 wagowy system liczenia 1594
 wahadło (na ekranie symbolicznie) 461
 wariantywny składnik 1089
 warunek zatrzymujący rekurencję 228, 233
 warunkowa kompilacja 275-278
 warunkowe wyrażenie 134
 wbudowany typ 46
 wchar_t 47, 609
 wchar_t - stałe znakowe tego typu 70
 wczytywanie formatowane 1381
 wczytywanie nieformatowane 1381
 wczytywanie z klawiatury długiego stringu 661-667
 wdowa po prenumeratorku (metafora) 1234
 wejścia/wyjścia operacje 1316-1389
 wektor
- › 2D 322
 - nieprostokątny 335-336
 - › 3D 337-339
 - nieprostokątnościenny 347-350
 - › agregatów 892
 - › dwuwymiarowy, definiowanie 322

- › dwuwymiarowy, inicjalizacja 326
 - › inicjalizacja go 894
 - › jako agregat 892
 - › nieagregatów 894
 - › obiektów 890-894
 - › trójwymiarowy 337-339
- what - funkcja składowa wyjątku 728, 792, 1423
 while 26
 wiatraczek - na ekranie (symbolicznie) 461
 wide character 47
 widmo promieniowania 961
 widoczne własności 1483
 width, funkcja 1347, 1361, 1367
 wielka szóstka funkcji składowych 1035
 wielka tablica 957
 wielodziedziczenie 1208
 wielokropek 498
 wielokrotne dziedziczenie 1208
 wielorakie dziedziczenie 1208
 wielowariantowy wybór 25, 31, 33
 wielowymiarowa
- › tablica 311-320, 1042
 - › tablica new 398
 - › tablica, a przeładowanie 481
 - › wektor 321-352
- wielowątkowy program 91
 wieloznaczność
- › a dziedziczenie wirtualne 1240
 - › przy dziedziczeniu 1212
 - › przy konwersji 942, 949, 951
- wierzchni const 419, 484
 wirtualne-
- › czysto (funkcja) 1293
 - › destruktor 1273-1274
 - › dziedziczenie 1237-1245
 - › funkcja 1253-1315, 1496
 - › klasa podstawowa 1237-1245
- wirtualność
- › a operatory we/wy 1331
 - › a przeładowanie 1275-1276
 - › a wczesne wiązanie 1270
- wiązanie
- › późne 1268-1269
 - › wczesne 1268-1269
 - › wczesne, a wirtualność 1270
- wolny format zapisu 9
 write(const char*, int) 1384
 ws, manipulator 1345
 wskaźnik 76, 353-368, 417-444
- › a auto (ostrzeżenie) 359
 - › a nazwa tablicy 375
 - › a reinterpret_cast 360-362
 - › argumentem funkcji 380-388

- › arytmetyka 377
- › definiowanie 355
- › do const 418-419
- › do funkcji 445-468
 - argumentem innej funkcji 454-457
 - definiowanie i odczytywanie deklaracji 448-453
 - orzekającej 1115
 - składowej
 - Zob. niżej: 'wsk. do fun. składowej klasy'*
 - tablica takich wsk. 458-462
 - typ tegoż 453
- › do klasy pochodnej 1224-1236
- › do obiektów klasy 881
- › do przebiegania (iterator) 1531
- › do składnika-danej 897-907
- › do składników klasy 896-924
 - tablica/wektor tychże 917
- › do składników statycznych 922
- › do stałej 418-419
- › dodawanie i odejmowanie liczby całkowitej 377
- › domeny zastosowania 369-416
- › dostęp do komórek pamięci 389
- › globalny 365
- › konwersja 497
- › odejmowanie dwóch od siebie 377
- › pisania i czytania 1424-1427
- › poruszanie nim 369
- › porównywanie 379
- › sposoby ustawiania 425-426
- › sprytny 1048
- › stały 417, 882
 - inicjalizacja go 418
- › tablica tychże 431-432
- › this 522-523
 - jego typ 523, 579
- › typu void* 363-364
- › ustawianie 356-358
- › ustawianie za pom. reinterpret_cast 389
- › w zastosowaniu do tablic 369-379
- › zwykły, we wnętrzu obiektu 897

wskaźnik czytania get 1424

wskaźnik do funkcji składowej klasy 908-916

- › a auto 910
- › a decltype 910
- › tablica/vector tychże 918-921

wskaźnik pisania put 1425

wstring 609

wszystkożerny catch 727

wybiórcze udostępnianie 1178, 1642

wybór wielowariantowy 25, 31, 33

wychwytywania lista 1136

wyciek pamięci 1094

wygaśnięcie dynastii 1173

wyjatek

- › a strumieniu 1420-1423
- › nowy sposób powiadamiania 409
- › przechwytywanie go 625
- › rzucający z konstruktorowej listy inic. 790
- › specyfikacja 731
- › z destruktora? 779
- › z funkcji string::at 625
- › z wyrażenia lambda 1162-1165

wyjatków specyfikacja

- › w wyrażeniu lambda 1139

wykładnicza notacja 66, 1323, 1339

wykładnik i cecha 1340

wyliczeniowy typ enum 96-105

- › jako indeks tablicy 542

wyrażenie

- › inicjalizujące 359
- › lambda 1114-1169
 - cztery formy zapisu 1133-1138
- › logiczne 20
- › regularne (regex) 74
- › warunkowe 134
 - a w nim definicja obiektu 62
 - w funkcji constexpr 246

wyrównanie adresu 115

wyróżnik unii 1090

wywołania funkcji - operator 448

wywoływalny obiekt 1129, 1150

X

x-wartość 870-872

Z

zacieranie funkcji 1275

zagnieżdżanie

- › a szablony 1552-1562
- › klasy w szablonie 1552-1562
- › komentarzy 14
- › zakresów 1171-1172

zagnieżdżona

- › deklaracja przyjaźni 1552
- › klasa 734-740
- › klasa, a przyjaciele 740

zagnieżdżony

- › szabł. funkcji składowych 1553
- › szablon funkcji 1552
- › szablon klas 1552

zakres

- › blok funkcji 79
- › klasy 1171

- pochodnej 1171
- podstawowej 1171
- › kwalifikator tegoż 1172
- › leksykalny 699, 740, 1329
 - szablonu kl. 1534
- › obszar pliku 80
- › przestrzeń nazw 80
- › ważności
 - Zob. niżej: 'zakres ważności'
- › zagnieżdżanie ich 1172
- zakres instrukcji 79
- zakres ważności
 - › definicja 78
 - › etykiety 212
 - › funkcja 79
 - › klasa 80, 507
 - › lokalny 78
 - › namespace 80
 - › nazw w funkcji 212
 - › nazw w klasie 506
 - › nazwy funkcji, a przeładowanie 477-478
 - › nazwy obiektu 212-217, 773
 - › obiektów globalnych 772
 - › obiektów lokalnych 771
 - › obiektów new 392, 772
 - › pliku 80
 - › zagnieżdżanie 1171
- zakresowe enum 103
- zakresowe for 169-171, 627
 - › a funkcje begin, end 1533
 - › a wektor wielowymiarowy 325
- zakresu operator 910
 - › a zasłanianie 86
- zależny typ w szablonie klasy 1532
- zapas pamięci 394, 881, 1057
 - › wyczerpanie go 408
- zapis binarny liczby 127, 1595
 - › jak dokonać zmiany 235
- zapowiadająca deklaracja 698
- zaprzyjaźniona funkcja 695
- zaprzyjaźniona klasa 703-704
- zasada trzech 856, 1020
- zasłanianie nazw 85-86, 525-528
- zasłanianie składnika 1172
- zasłonięcie i przeładowanie 529
- zasłonięta nazwa globalna - dostęp 527
- zatrzymanie rekurencji 228
- zawieranie
 - › obiektu 1476, 1479, 1493
 - › obiektów, a dziedziczenie klas - różnica 1222-1223
- zawieszający się program 377
- zawężająca konwersja 60
- zbieracz śmieci (garbage collector) 1058
- zdarzenie (w fizyce jądrowej) 1108
- zerowanie nowej tablicy 398
- zerowy adres 67
- zespolone liczby 925, 954-955
- zgrupowanie danych 883
- zjawisko Dopplera 911
- zmiana formatu operacji we/wy 1360-1365
- zmienna 15
 - › automatyczna 213
 - › statyczna globalna 222
 - › statyczna lokalna 214
- znak
 - › specjalny 69
 - › szeroki 47
 - › wypełniający 1350
- znakowe stałe dosłowne 68
- źródłowy kod programu 11
- źródło strumienia 1317
- złożony typ 46, 76

PROGRAM PARTNERSKI

— GRUPY HELION —

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

OPUS MAGNUM C++11

Programowanie w języku C++

Dzięki tej książce poznasz:

- Proste i złożone typy danych
- Instrukcje sterujące
- Funkcje i operatory
- Wskaźniki
- Klasy i dziedziczenie
- Obsługę wyjątków
- Wyrażenia lambda
- Operacje wejścia-wyjścia
- Projektowanie zorientowane obiektowo
- Szablony

Jeżeli chcesz nauczyć się tego języka w łatwy, pogodny, przyjazny sposób, ta książka jest właśnie dla Ciebie.

Jedno C i same plusy!

Dawno, dawno temu, w głębokich latach osiemdziesiątych ubiegłego wieku, pewien duński informatyk, zainspirowany językiem C, opracował jeden z najważniejszych, najbardziej elastycznych i do dziś niezastąpionych języków programowania – C++. Dziś ten język ten wykorzystywany do tworzenia gier komputerowych, obliczeń naukowych, technicznych, w medycynie, przemyśle i bankowości. NASA posługuje się nim w naziemnej kontroli lotów. Duża część oprogramowania Międzynarodowej Stacji Kosmicznej została napisana w tym języku. Nawet w marsjańskim łaziku Curiosity pracuje program w C++, który analizuje obraz z kamer i planuje dalszą trasę.

Autor tej książki – wybitny specjalista pracujący nad wieloma znaczącymi projektami we francuskich, niemieckich i włoskich instytucjach fizyki jądrowej, znany czytelnikom m.in. z genialnej *Symfonii C++* – postawił sobie za cel napisanie nowej, przekrojowej publikacji o tym języku, która w prostym, obrazowym stylu wprowadza czytelnika w fascynujący świat programowania zorientowanego obiektowo. Zobacz, jak potężny jest C++ 11.

Helion 

 helion.pl

 **HELION SA**
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

Sprawdź nasze szkolenia!

SZKOLENIA



AKADEMIA IT & BUSINESS

HELIONSZKOLENIA.PL

KOD KORZYŚCI
Sięgnij po więcej! ▶



ISBN 978-83-283-6965-8

