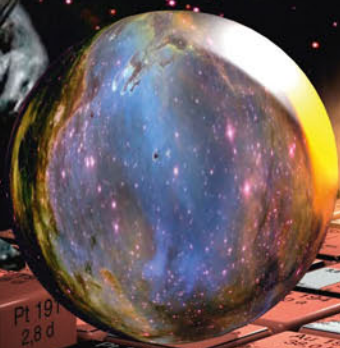


Jerzy Grębosz

OPUS MAGNUM C++11

Programowanie w języku C++

ŁATWY PODRĘCZNIK



Helion

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 Wydawnictwo HELION 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 Wydawnictwo HELION 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

Przygotowanie okładki: Jan Paluch

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

Wydanie pierwsze

ISBN: 978-83-283-3882-1

Copyright © Jerzy Grębosz 2018

Printed in Poland

Wydawnictwo HELION

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/ocpp11>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:

<ftp://ftp.helion.pl/przyklady/ocpp11.zip>

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

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

Spis treści trzech tomów

Tom 1

0	Proszę tego nie czytać!.....	1
0.1	Zaprzyężnijmy 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	bool – 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 instrukcji.....	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	Zasłanianie 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 wyliczeniowe <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	Cwiczenia	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	Cwiczenia	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	Cwiczenia	172

6 Funkcje.....174

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: lwartość i rwartość.....	188
6.10	Referencje do lwartości i referencje do rwartości 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 lwartości	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 Preprocesor270

7.1	Dyrektywa pusta <i>#</i>	270
7.2	Dyrektywa <i>#define</i>	270
7.3	Dyrektywa <i>#undef</i>	272
7.4	Makrodefinicje.....	273
7.5	Sklejacz nazw argumentów, czyli operator <i>##</i>	275
7.6	Parametr aktualny makrodefinicji – w postaci tekstu	276
7.7	Dyrektywy kompilacji warunkowej.....	276
7.8	Dyrektywa <i>#error</i>	280
7.9	Dyrektywa <i>#line</i>	281
7.10	Wstawianie treści innych plików do tekstu kompilowanego właśnie pliku.....	281
7.11	Dyrektywy zależne od implementacji	283
7.12	Nazwy predefiniowane	283
7.13	Ćwiczenia	286

8 Tablice289

8.1	Co to jest tablica	289
8.2	Elementy tablicy	290
8.3	Inicjalizacja tablic	292
8.4	Przekazywanie tablicy do funkcji	293
8.5	Przykład z tablicą elementów typu <i>enum</i>	297
8.6	Tablice znakowe	299
8.7	Ćwiczenia	307

9 Tablice wielowymiarowe312

9.1	Tablica tablic	312
9.2	Przykład programu pracującego z tablicą dwuwymiarową.....	314
9.3	Gdzie w pamięci jest dany element tablicy.....	316
9.4	Typ wyrażeń związanych z tablicą wielowymiarową.....	316
9.5	Przesyłanie tablic wielowymiarowych do funkcji	318
9.6	Ćwiczenia	320

10 Wektory wielowymiarowe.....322

10.1	Najpierw przypomnienie istotnych tu cech klasy <i>vector</i>	322
10.2	Jak za pomocą klasy <i>vector</i> budować tablice wielowymiarowe	323
10.3	Funkcja pokazująca zawartość wektora dwuwymiarowego	324
10.4	Definicja dwuwymiarowego wektora – pustego.....	326
10.5	Definicja wektora dwuwymiarowego z listą inicjalizatorów	327
10.6	Wektor dwuwymiarowy o żądanych rozmiarach, choć bez inicjalizacji	328
10.7	Zmiana rozmiarów wektora dwuwymiarowego funkcją <i>resize</i>	329
10.8	Zmiany rozmiaru wektora 2D funkcjami <i>push_back</i> , <i>pop_back</i>	330
10.9	Zmniejszanie rozmiaru wektora dwuwymiarowego funkcją <i>pop_back</i>	333
10.10	Funkcje mogące modyfikować treść wektora 2D.....	333
10.11	Wysłanie rzędu wektora 2D do funkcji pracującej z wektorem 1D.....	335
10.12	Całość przykładu definiującego wektory dwuwymiarowe	336
10.13	Po co są dwuwymiarowe wektory nieprostokątne.....	336

10.14	Wektory trójwymiarowe.....	338
10.15	Sposoby definicji wektora 3D o ustalonych rozmiarach.....	341
10.16	Nadawanie pustemu wektorowi 3D wymaganych rozmiarów.....	345
10.16.1	Zmiana rozmiarów wektora 3D funkcjami <i>resize</i>	345
10.16.2	Zmiana rozmiarów wektora 3D funkcjami <i>push_back</i>	347
10.17	Trójwymiarowe wektory 3D – nieprostokątne.....	348
10.18	Ćwiczenia.....	352

11 Wskaźniki – wiadomości wstępne.....354

11.1	Wskaźniki mogą bardzo ułatwić życie.....	354
11.2	Definiowanie wskaźników.....	356
11.3	Praca ze wskaźnikiem.....	357
11.4	Definiowanie wskaźnika z użyciem <i>auto</i>	360
11.5	Wyrażenie <i>*wskaźnik</i> jest wartością.....	361
11.6	Operator rzutowania <i>reinterpret_cast</i> a wskaźniki.....	361
11.7	Wskaźniki typu <i>void*</i>	364
11.8	Strzał na oślep – wskaźnik zawsze na coś wskazuje.....	366
11.8.1	Wskaźnik wolno porównać z adresem zero – <i>nullptr</i>	368
11.9	Ćwiczenia.....	368

12 Cztery domeny zastosowania wskaźników.....370

12.1	Zastosowanie wskaźników wobec tablic.....	370
12.1.1	Ćwiczenia z mechaniki ruchu wskaźnika.....	370
12.1.2	Użycie wskaźnika w pracy z tablicą.....	374
12.1.3	Arytmetyka wskaźników.....	378
12.1.4	Porównywanie wskaźników.....	380
12.2	Zastosowanie wskaźników w argumentach funkcji.....	381
12.2.1	Jeszcze raz o przesyłaniu tablic do funkcji.....	385
12.2.2	Odbieranie tablicy jako wskaźnik.....	385
12.2.3	Argument formalny będący wskaźnikiem do obiektu <i>const</i>	387
12.3	Zastosowanie wskaźników przy dostępie do konkretnych komórek pamięci.....	390
12.4	Rezerwacja obszarów pamięci.....	391
12.4.1	Operatory <i>new</i> i <i>delete</i> albo Oratorium Stworzenie Świata.....	392
12.4.2	Operator <i>new</i> a słowo kluczowe <i>auto</i>	396
12.4.3	Inicjalizacja obiektu tworzonych operatorem <i>new</i>	396
12.4.4	Operatorem <i>new</i> możemy także tworzyć obiekty stałe.....	397
12.4.5	Dynamiczna alokacja tablicy.....	398
12.4.6	Tablice wielowymiarowe tworzone operatorem <i>new</i>	399
12.4.7	Umiejscawiający operator <i>new</i>	402
12.4.8	„Przychodzimy, odchodzimy – cichuteńko, na...”.....	407
12.4.9	Zapas pamięci to nie studnia bez dna.....	409
12.4.10	Nowy sposób powiadomienia: rzucenie wyjątku <i>std::bad_alloc</i>	410
12.4.11	Funkcja <i>set_new_handler</i>	412
12.5	Ćwiczenia.....	414

13 Wskaźniki – runda trzecia.....418

13.1	Stałe wskaźniki.....	418
13.2	Stałe wskaźniki a wskaźniki do stałych.....	419
13.2.1	Wierzch i głębia.....	420
13.3	Definiowanie wskaźnika z użyciem <i>auto</i>	421
13.3.1	Symbol zastępczy <i>auto</i> a opuszczanie gwiazdki przy definiowaniu wskaźnika.....	424
13.4	Sposoby ustawiania wskaźników.....	426
13.5	Parada kłamaczów, czyli o rzutowaniu <i>const_cast</i>	428

13.6	Tablice wskaźników	432
13.7	Wariacje na temat C-stringów	434
13.8	Argumenty z linii wywołania programu	441
13.9	Ćwiczenia	444

14 Wskaźniki do funkcji446

14.1	Wskaźnik, który może wskazywać na funkcję	446
14.2	Ćwiczenia z definiowania wskaźników do funkcji	449
14.3	Wskaźnik do funkcji jako argument innej funkcji	455
14.4	Tablica wskaźników do funkcji	459
14.5	Użycie deklaracji <i>using</i> i <i>typedef</i> w świecie wskaźników	464
14.5.1	Alias przydatny w argumencie funkcji.....	464
14.5.2	Alias przydatny w definicji tablicy wskaźników do funkcji	465
14.6	Użycie <i>auto</i> lub <i>decltype</i> do automatycznego rozpoznania potrzebnego typu	466
14.7	Ćwiczenia	468

15 Przeładowanie nazwy funkcji.....470

15.1	Co oznacza przeładowanie.....	470
15.2	Przeładowanie od kuchni.....	473
15.3	Jak możemy przeładowywać, a jak się nie da?	473
15.4	Czy przeładowanie nazw funkcji jest techniką orientowaną obiektowo?	476
15.5	Linkowanie z modułami z innych języków	477
15.6	Przeładowanie a zakres ważności deklaracji funkcji	478
15.7	Rozważania o identyczności lub odmienności typów argumentów.....	480
15.7.1	Przeładowanie a typy tworzone z <i>using</i> lub <i>typedef</i> oraz typy <i>enum</i>	481
15.7.2	Tablica a wskaźnik	481
15.7.3	Pewne szczegóły o tablicach wielowymiarowych.....	482
15.7.4	Przeładowanie a referencja.....	484
15.7.5	Identyczność typów: <i>T</i> , <i>const T</i> , <i>volatile T</i>	485
15.7.6	Przeładowanie a typy: <i>T*</i> , <i>volatile T*</i> , <i>const T*</i>	486
15.7.7	Przeładowanie a typy: <i>T&</i> , <i>volatile T&</i> , <i>const T&</i>	487
15.8	Adres funkcji przeładowanej	488
15.8.1	Zwrot rezultatu będącego adresem funkcji przeładowanej.....	490
15.9	Kulisy dopasowywania argumentów do funkcji przeładowanych.....	492
15.10	Etapy dopasowania	493
15.10.1	Etap 1. Dopasowanie dokładne	493
15.10.2	Etap 1a. Dopasowanie dokładne, ale z tzw. trywialną konwersją.....	494
15.10.3	Etap 2. Dopasowanie z awansem (z promocją).....	495
15.10.4	Etap 3. Próba dopasowania za pomocą konwersji standardowych	497
15.10.5	Etap 4. Dopasowanie z użyciem konwersji zdefiniowanych przez użytkownika	499
15.10.6	Etap 5. Dopasowanie do funkcji z wielokropkiem	499
15.11	Wskaźników nie dopasowuje się inaczej niż dosłownie.....	499
15.12	Dopasowywanie wywołań z kilkoma argumentami	500
15.13	Ćwiczenia	501

16 Klasy504

16.1	Typy definiowane przez użytkownika.....	504
16.2	Składniki klasy	506
16.3	Składnik będący obiektem	507
16.4	Kapsułowanie	508
16.5	Ukrywanie informacji.....	509
16.6	Klasa a obiekt	512
16.7	Wartości wstępne w składnikach nowych obiektów. Inicjalizacja „w klasie”	514
16.8	Funkcje składowe	517

16.8.1	Posługiwanie się funkcjami składowymi	517
16.8.2	Definiowanie funkcji składowych	518
16.9	Jak to właściwie jest? (<i>this</i>)	523
16.10	Odwołanie się do publicznych danych składowych obiektu	525
16.11	Zasłanianie nazw	526
16.11.1	Nie sięgaj z klasy do obiektów globalnych	529
16.12	Przeładowanie i zasłonięcie równocześnie	530
16.13	Nowa klasa? Osobny plik!	530
16.13.1	Poznajmy praktyczną realizację wieloplikowego programu	533
16.13.2	Zasada umieszczania dyrektywy <i>using namespace</i> w plikach	545
16.14	Przesyłanie do funkcji argumentów będących obiektami	545
16.14.1	Przesyłanie obiektu przez wartość	545
16.14.2	Przesyłanie przez referencję	547
16.15	Konstruktor – pierwsza wzmianka	548
16.16	Destruktor – pierwsza wzmianka	553
16.17	Składnik statyczny	557
16.17.1	Do czego może się przydać składnik statyczny w klasie?	566
16.18	Styczna funkcja składowa	566
16.18.1	Deklaracja składnika statycznego mająca inicjalizację „w klasie”	571
16.19	Funkcje składowe typu <i>const</i> oraz <i>volatile</i>	577
16.19.1	Przeładowanie a funkcje składowe <i>const</i> i <i>volatile</i>	581
16.20	Struktura	582
16.21	Klasa będąca agregatem. Klasa bez konstruktora	582
16.22	Funkcje składowe z przydomkiem <i>constexpr</i>	585
16.23	Specyfikator <i>mutable</i>	591
16.24	Bardziej rozbudowany przykład zastosowania klasy	593
16.25	Ćwiczenia	603

Tom 2

17 Biblioteczna klasa *std::string*609

17.1	Rozwiązanie przechowywania tekstów musiało się znaleźć	609
17.2	Klasa <i>std::string</i> to przecież nasz stary znajomy	611
17.3	Definiowanie obiektów klasy <i>string</i>	612
17.4	Użycie operatorów =, +, += w pracy ze stringami	617
17.5	Pojemność, rozmiar i długość stringu	618
17.5.1	Bliźniacze funkcje <i>size()</i> i <i>length()</i>	618
17.5.2	Funkcja składowa <i>empty</i>	619
17.5.3	Funkcja składowa <i>max_size</i>	619
17.5.4	Funkcja składowa <i>capacity</i>	619
17.5.5	Funkcje składowe <i>reserve</i> i <i>shrink_to_fit</i>	621
17.5.6	<i>resize</i> – zmiana długości stringu „na siłę”	622
17.5.7	Funkcja składowa <i>clear</i>	624
17.6	Użycie operatora <i>[]</i> oraz funkcji <i>at</i>	624
17.6.1	Działanie operatora <i>[]</i>	625
17.6.2	Działanie funkcji składowej <i>at</i>	626
17.6.3	Przebieganie po wszystkich literach stringu zakresowym <i>for</i>	629
17.7	Funkcje składowe <i>front</i> i <i>back</i>	629
17.8	Jak umieścić w tekście liczbę?	630
17.9	Jak wczytać liczbę ze stringu?	632

17.10	Praca z fragmentem stringu, czyli z substringiem	635
17.11	Funkcja składowa <i>substr</i>	636
17.12	Szukanie zadanego substringu w obiekcie klasy <i>string</i> – funkcje <i>find</i>	637
17.13	Szukanie rozpoczynane od końca stringu	640
17.14	Szukanie w stringu jednego ze znaków z zadanego zestawu	641
17.15	Usuwanie znaków ze stringu – <i>erase</i> i <i>pop_back</i>	643
17.16	Wstawianie znaków do istniejącego stringu – funkcje <i>insert</i>	644
17.17	Zamiana części znaków na inne znaki – <i>replace</i>	646
17.18	Zagłębienie do wnętrza obiektu klasy <i>string</i> funkcją <i>data</i>	649
17.19	Zawartość obiektu klasy <i>string</i> a C-string	650
17.20	W porządku alfabetycznym, czyli porównywanie stringów	653
17.20.1	Porównywanie stringów za pomocą funkcji <i>compare</i>	654
17.20.2	Porównywanie stringów przy użyciu operatorów <i>==</i> , <i>!=</i> , <i><</i> , <i><=</i> , <i>>=</i>	658
17.21	Zamiana treści stringu na małe lub wielkie litery	659
17.22	Kopiowanie treści obiektu klasy <i>string</i> do tablicy znakowej – funkcja <i>copy</i>	662
17.23	Wzajemna zamiana treści dwóch obiektów klasy <i>string</i> – funkcja <i>swap</i>	662
17.24	Wczytywanie z klawiatury stringu o nieznanym wcześniej długości – <i>getline</i>	663
17.24.1	Pułapka, czyli jak <i>getline</i> może Cię zaskoczyć	666
17.25	Iteratory stringu	670
17.25.1	Iterator do obiektu stałego	674
17.25.2	Funkcje składowe klasy <i>string</i> pracujące z iteratorami	675
17.26	Klasa <i>string</i> korzysta z techniki przenoszenia	680
17.27	Bryk, czyli „pamięć zewnętrzna” programisty	681
17.28	Ćwiczenia	689

18 Deklaracje przyjaźni696

18.1	Przyjaciele w życiu i w C++	696
18.2	Przykład: dwie klasy deklarują przyjaźń z tą samą funkcją	698
18.3	W przyjaźni trzeba pamiętać o kilku sprawach	700
18.4	Obdarzenie przyjaźnią funkcji składowej innej klasy	703
18.5	Klasy zaprzyjaźnione	705
18.6	Konwencja umieszczania deklaracji przyjaźni w klasie	707
18.7	Kilka otrzeźwiających słów na zakończenie	707
18.8	Ćwiczenia	708

19 Obsługa sytuacji wyjątkowych710

19.1	Jak dać znać, że coś się nie udało?	710
19.2	Pierwszy prosty przykład	712
19.3	Kolejność bloków <i>catch</i> ma znaczenie	714
19.4	Który blok <i>catch</i> nadaje się do złapania lecącego wyjątku?	715
19.5	Bloki <i>try</i> mogą być zagnieżdżane	718
19.6	Obsługa wyjątków w praktycznym programie	721
19.7	Specyfikator <i>noexcept</i> i operator <i>noexcept</i>	731
19.8	Ćwiczenia	734

20 Klasa-składnik oraz klasa lokalna737

20.1	Klasa-składnik, czyli gdy w klasie jest zagnieżdżona definicja innej klasy	737
20.2	Prawdziwy przykład zagnieżdżenia definicji klasy	744
20.3	Lokalna definicja klasy	755
20.4	Lokalne nazwy typów	758
20.5	Ćwiczenia	759

21 Konstruktory i destruktory761

21.1	Konstruktor	761
21.1.1	Przykład programu zawierającego klasę z konstruktorami	762
21.2	Specyfikator (przydomek) <i>explicit</i>	773
21.3	Kiedy i jak wywołwany jest konstruktor	774
21.3.1	Konstruowanie obiektów lokalnych	774
21.3.2	Konstruowanie obiektów globalnych	775
21.3.3	Konstrukcja obiektów tworzonych operatorem <i>new</i>	775
21.3.4	Jawne wywołanie konstruktora	776
21.3.5	Dalsze sytuacje, gdy pracuje konstruktor	779
21.4	Destruktor	779
21.4.1	Jawne wywołanie destruktora (ogromnie rzadka sytuacja)	781
21.5	Nie rzucacie wyjątków z destruktorów	781
21.6	Konstruktor domniemany	783
21.7	Funkcje składowe z przypiskami = <i>default</i> i = <i>delete</i>	784
21.8	Konstruktorowa lista inicjalizacyjna składników klasy	786
21.8.1	Dla wtajemniczonych: wyjątki rzucone z konstruktorowej listy inicjalizacyjnej	793
21.9	Konstruktor delegujący	797
21.10	Pomocnicza klasa <i>std::initializer_list</i> – lista inicjalizatorów	804
21.10.1	Zastosowania niekonstruktorowe	804
21.10.2	Konfuzja: lista inicjalizatorów a lista inicjalizacyjna	813
21.10.3	Konstruktor z argumentem będącym klamrową listą inicjalizatorów	814
21.11	Konstrukcja obiektu, którego składnikiem jest obiekt innej klasy	819
21.12	Konstruktory niepubliczne?	826
21.13	Konstruktory <i>constexpr</i> mogą wytwarzać obiekty <i>constexpr</i>	828
21.14	Cwiczenia	838

22 Konstruktory: kopiujący i przenoszący841

22.1	Konstruktor kopiujący (albo inicjalizator kopiujący)	841
22.2	Przykład klasy z konstruktorem kopiującym	842
22.3	Kompilatorowi wolno pominąć niepotrzebne kopiowanie	847
22.4	Dlaczego przez referencję?	849
22.5	Konstruktor kopiujący gwarantujący nietykalność	850
22.6	Współodpowiedzialność	851
22.7	Konstruktor kopiujący generowany automatycznie	851
22.8	Kiedy powinniśmy sami zdefiniować konstruktor kopiujący?	852
22.9	Referencja do wartości daje zezwolenie na recykling	859
22.10	Funkcja <i>std::move</i> , która nie przenosi, a tylko rzutuje	862
22.11	Odebrana wartość staje się w ciele funkcji lwartością	864
22.12	Konstruktor przenoszący (inicjalizator przenoszący)	866
22.12.1	Konstruktor przenoszący generowany przez kompilator	871
22.12.2	Inne konstruktory generowane automatycznie	871
22.12.3	Zwrot obiektu lokalnego przez wartość? Nie używamy przenoszenia!	872
22.13	Tak zwana „semantyka przenoszenia”	873
22.14	Nowe pojęcia dla ambitnych: glwartość, xwartość i prwartość	873
22.15	<i>decltype</i> – operator rozpoznawania typu bardzo wyszukanych wyrażeń	876
22.16	Cwiczenia	881

23 Tablice obiektów883

23.1	Definiowanie tablic obiektów i praca z nimi	883
23.2	Tablica obiektów definiowana operatorem <i>new</i>	884
23.3	Inicjalizacja tablic obiektów	886
23.3.1	Inicjalizacja tablicy, której obiekty są agregatami	886

23.3.2	Inicjalizacja tablic, których elementy nie są agregatami	889
23.4	Wektory obiektów	893
23.4.1	Wektor, którego elementami są obiekty klasy będącej agregatem	895
23.4.2	Wektor, którego elementami są obiekty klasy niebędącej agregatem	897
23.5	Ćwiczenia	898

24 Wskaźnik do składników klasy899

24.1	Wskaźniki zwykłe – repetytorium	899
24.2	Wskaźnik do pokazywania na składnik-daną	900
24.2.1	Przykład zastosowania wskaźników do składników klasy	904
24.3	Wskaźnik do funkcji składowej	911
24.3.1	Przykład zastosowania wskaźników do funkcji składowych	913
24.4	Tablica wskaźników do danych składowych klasy	920
24.5	Tablica wskaźników do funkcji składowych klasy	921
24.5.1	Przykład tablicy/wektora wskaźników do funkcji składowych	922
24.6	Wskaźniki do składników statycznych są zwykłe	925
24.7	Ćwiczenia	926

25 Konwersje definiowane przez użytkownika928

25.1	Sformułowanie problemu	928
25.2	Konstruktory konwertujące	930
25.2.1	Kiedy jawnie, kiedy niejawnie	931
25.2.2	Przykład konwersji konstruktorem	936
25.3	Funkcja konwertująca – operator konwersji	938
25.3.1	Na co funkcja konwertująca zamieniać nie może	944
25.4	Który wariant konwersji wybrać?	945
25.5	Sytuacje, w których zachodzi konwersja	947
25.6	Zapis jawnego wywołania konwersji typów	948
25.6.1	Advocatus zapisu przypominającego: „wywołanie funkcji”	948
25.6.2	Advocatus zapisu: „rzutowanie”	949
25.7	Nie całkiem pasujące argumenty, czyli konwersje kompilatora przy dopasowaniu	949
25.8	Kilka rad dotyczących konwersji	954
25.9	Ćwiczenia	955

26 Przeładowanie operatorów957

26.1	Co to znaczy przeładować operator?	957
26.2	Przeładowanie operatorów – definicja i trochę teorii	959
26.3	Moje zabawki	963
26.4	Funkcja operatorowa jako funkcja składowa	964
26.5	Funkcja operatorowa nie musi być przyjacielem klasy	967
26.6	Operatory predefiniowane	967
26.7	Ile operandów ma mieć ten operator?	968
26.8	Operatory jednooperandowe	968
26.9	Operatory dwuoperandowe	971
26.9.1	Przykład na przeładowanie operatora dwuoperandowego	971
26.9.2	Przemienność	973
26.9.3	Choć operatory inne, to nazwę mają tę samą	974
26.10	Przykład zupełnie niematematyczny	974
26.11	Operatory postinkrementacji i postdekrementacji – koniec z niesprawiedliwością	984
26.12	Praktyczne rady dotyczące przeładowania	986
26.13	Pojedynek: operator jako funkcja składowa czy globalna?	988
26.14	Zasłona spada, czyli tajemnica operatora <<	989
26.15	Stałe dosłowne definiowane przez użytkownika	995
26.15.1	Przykład: stałe dosłowne użytkownika odbierane jako gotowane	999

26.15.2	Przykład: stałe dosłowne użytkownika odbierane na surowo	1008
26.16	Ćwiczenia	1011

27 Przeladowanie: =, [], (), ->.....1015

27.1	Cztery operatory, które muszą być niestacycznymi funkcjami składowymi.....	1015
27.2	Operator przypisania = (wersja kopiująca)	1015
27.2.1	Przykład na przeladowanie (kopiującego) operatora przypisania	1017
27.2.2	Przypisanie „kaskadowe”.....	1024
27.2.3	Po co i jak zabezpieczamy się przed przypisaniem $a = a$	1026
27.2.4	Jak opowiedzieć potocznie o konieczności istnienia operatora przypisania?.....	1027
27.2.5	Kiedy kopiujący operator przypisania nie jest generowany automatycznie	1029
27.3	Przenoszący operator przypisania =	1029
27.4	Specjalne funkcje składowe i nierealna prosta zasada	1038
27.5	Operator [].....	1039
27.6	Operator ().....	1043
27.7	Operator ->	1049
27.7.1	„Spirytny wskaźnik” wykorzystuje przeladowanie właśnie tego operatora	1051
27.8	Ćwiczenia	1058

Tom 3

28 Przeladowanie operatorów *new* i *delete* na użytek klasy1061

28.1	Po co przeladowujemy operatory <i>new</i> i <i>new[]</i>	1061
28.2	Funkcja <i>operator new</i> i <i>operator new[]</i> w klasie K	1062
28.3	Jak się deklaruje operatory <i>new</i> i <i>delete</i> w klasie?.....	1065
28.4	Przykładowy program z przeladowanymi <i>new</i> i <i>delete</i>	1067
28.4.1	Gdy dopuszczamy rzucanie wyjątku <i>std::bad_alloc</i>	1068
28.4.2	Po staremu nadal można.....	1073
28.4.3	Rezerwacja tablicy obiektów naszej klasy <i>Twektorek</i>	1073
28.4.4	Nasze własne argumenty wysłane do operatora <i>new</i>	1075
28.4.5	🐞 Operatory <i>new</i> i <i>delete</i> odziedziczone do klasy pochodnej.....	1077
28.4.6	A jednak polimorfizm jest możliwy	1079
28.4.7	Tworzenie i likwidowanie tablicy obiektów klasy pochodnej	1079
28.4.8	Operatory <i>new</i> , które nie rzucają wyjątku <i>std::bad_alloc</i>	1080
28.5	Rzut oka wstecz na przeladowanie operatorów	1085
28.6	Ćwiczenia	1086

29 Unie i pola bitowe1088

29.1	Unia	1088
29.2	Unia anonimowa.....	1090
29.3	Klasa uniopodobna (unia z metryczką)	1092
29.4	Gdy składnik unii jest obiektem jakiejś klasy.....	1094
29.5	Unia o składnikach mających swe konstruktory, destruktory itp.	1096
29.6	Pola bitowe	1103
29.7	Unia i pola bitowe upraszczają deszyfrowanie słów danych	1107
29.8	Ćwiczenia	1114

30 Wyrażenia lambda i wysłanie kodu do innych funkcji 1118

30.1	Preludium: dwa sposoby przesłania kryterium oceniania.....	1118
30.1.1	Sposób I. Kryterium przekazane wskaźnikiem do funkcji (orzekającej)	1121
30.1.2	Sposób II. Kryterium umieszczone w obiekcie funkcyjnym.....	1123
30.1.3	Kryterium oceny z parametrem (czyli o wyższości funktorów).....	1125
30.1.4	Funkcja-algorytm biblioteczny <i>std::count_if</i>	1127
30.1.5	Co lepsze: funkcja orzekająca czy orzekający obiekt funkcyjny?.....	1130
30.2	Wyrażenie lambda	1132
30.3	Formy wyrażenia lambda	1137
30.3.1	Lista argumentów (formalnych).....	1138
30.3.2	Ciało wyrażenia lambda.....	1138
30.3.3	Typ rezultatu	1139
30.3.4	Lista wychwytywania.....	1140
30.3.5	Słowo kluczowe <i>mutable</i> w wyrażeniu lambda.....	1142
30.3.6	Specyfikacja dotycząca wyjątków rzucanych z wyrażenia lambda.....	1143
30.4	Wyrażenie lambda zastosowane w funkcji składowej.....	1143
30.5	Tworzenie (nazwanych) obiektów lambda słowem <i>auto</i>	1147
30.5.1	Tworzenie obiektów na lambdy słowem kluczowym <i>auto</i>	1148
30.5.2	Tworzenie (nazwanych) obiektów lambda szablonem <i>std::function</i>	1150
30.6	Stowarzyszenie martwych referencji	1155
30.7	Rekurencja przy użyciu wyrażenia lambda	1158
30.8	Wyrażenie lambda jako domniemana wartość argumentu.....	1162
30.9	Rzucanie wyjątków z wyrażenia lambda.....	1166
30.10	Vivat lambda!	1170
30.11	Cwiczenia	1171

31 Dziedziczenie klas 1174

31.1	Istota dziedziczenia.....	1174
31.2	Dostęp do składników	1177
31.2.1	Prywatne składniki klasy podstawowej.....	1177
31.2.2	Nieprywatne składniki klasy podstawowej.....	1179
31.2.3	Klasa pochodna też decyduje	1180
31.2.4	Deklaracja dostępu <i>using</i> , czyli udostępnianie wybiórcze	1182
31.3	Czego się nie dziedziczy.....	1185
31.3.1	„Niedziedziczenie” konstruktorów	1185
31.3.2	„Niedziedziczenie” operatora przypisania.....	1186
31.3.3	„Niedziedziczenie” destruktora	1186
31.4	Drzewo genealogiczne.....	1187
31.5	Dziedziczenie – doskonałe narzędzie programowania.....	1188
31.6	Kolejność wywoływania konstruktorów	1190
31.7	Przypisanie i inicjalizacja obiektów w warunkach dziedziczenia.....	1196
31.7.1	Klasa pochodna nie definiuje swojego kopiującego operatora przypisania	1196
31.7.2	Klasa pochodna nie definiuje swojego konstruktora kopiującego	1197
31.7.3	Inicjalizacja i przypisywanie według obiektu będącego <i>const</i>	1198
31.8	Przykład: konstruktor kopiujący i operator przypisania dla klasy pochodnej	1198
31.8.1	Jak zainstalować mechanizm kopiowania w klasie pochodnej	1204
31.8.2	Jak w klasie pochodnej zainstalować mechanizm przenoszenia	1208
31.9	Dziedziczenie od kilku „rodziców” (wielodziedziczenie)	1212
31.9.1	Konstruktor klasy pochodnej przy wielodziedziczeniu.....	1213
31.9.2	Ryzyko wieloznaczności przy wielodziedziczeniu	1216
31.9.3	Czy bliższe pokrewieństwo usuwa wieloznaczność?	1218
31.9.4	Poszlaki	1218
31.10	Sposób na „odziedziczenie” konstruktorów	1219
31.11	Pojedynek: dziedziczenie klasy contra zawieranie obiektów składowych.....	1226

31.12	Wspaniałe konwersje standardowe przy dziedziczeniu	1228
31.12.1	Panorama korzyści	1232
31.12.2	Czego się nie opłaca robić.....	1234
31.12.3	Tuzin samochodów nie jest rodzajem tuzina pojazdów	1235
31.12.4	Konwersje standardowe wskaźnika do składnika klasy	1239
31.13	Wirtualne klasy podstawowe	1241
31.13.1	Publiczne i prywatne dziedziczenie tej samej klasy wirtualnej	1245
31.13.2	Uwagi o konstrukcji i inicjalizacji w przypadku klas wirtualnych	1245
31.13.3	Dominacja klas wirtualnych.....	1249
31.14	Ćwiczenia	1250

32 Wirtualne funkcje składowe1257

32.1	Wirtualny znaczy: (teoretycznie) możliwy	1257
32.2	Polimorfizm	1264
32.3	Typy rezultatów różnych realizacji funkcji wirtualnej	1267
32.3.1	Zamiast „odpowiedni typ rezultatu” kompilator powie „kowariant”	1268
32.4	Dalsze cechy funkcji wirtualnej.....	1270
32.5	Wczesne i późne wiązanie	1272
32.6	Kiedy dla wywołań funkcji wirtualnych zachodzi jednak wczesne wiązanie?.....	1274
32.7	Kulisy białej magii, czyli: jak to jest zrobione?.....	1275
32.8	Funkcja wirtualna, a mimo to <i>inline</i>	1277
32.9	Destruktor? Najlepiej wirtualny!	1277
32.10	Pojedynek – funkcje przeładowane, zasłaniające się i wirtualne (zacierające się).....	1279
32.11	Kontekstowe słowa kluczowe <i>override</i> i <i>final</i>	1281
32.11.1	Przykład użycia <i>override</i> i <i>final</i> , a także wirtualnych destruktorów	1282
32.12	Klasy abstrakcyjne.....	1294
32.13	Wprawdzie konstruktor nie może być wirtualny, ale... ..	1301
32.14	Rzutowanie <i>dynamic cast</i> jest dla typów polimorficznych.....	1307
32.15	POD, czyli Pospolite Stare Dane	1310
32.16	Wszystko, co najważniejsze	1313
32.17	Finis coronat opus.....	1316
32.18	Ćwiczenia	1316

33 Operacje wejścia/wyjścia – podstawy1320

33.1	Biblioteka <i>iostream</i>	1321
33.2	Strumień	1321
33.3	Strumienie zdefiniowane standardowo	1323
33.4	Operatory <i>>></i> i <i><<</i>	1324
33.5	Domniemania w pracy strumieni zdefiniowanych standardowo	1325
33.6	Uwaga na priorytet	1328
33.7	Operatory <i><<</i> oraz <i>>></i> definiowane przez użytkownika	1329
33.7.1	Operatorów wstawiania i wyjmowania ze strumienia nie dziedziczy się	1334
33.7.2	Operatory wstawiania i wyjmowania nie mogą być wirtualne. Niestety.....	1335
33.8	Sterowanie formatem.....	1338
33.9	Flagi stanu formatowania	1338
33.9.1	Znaczenie poszczególnych flag sterowania formatem	1340
33.10	Sposoby zmiany trybu (reguł) formatowania	1345
33.11	Manipulatory	1345
33.11.1	Manipulatory bezargumentowe	1346
33.11.2	Manipulatory mające argumenty	1351
33.11.3	Manipulator <i>setw(int)</i>	1351
33.11.4	Manipulator <i>setfill</i>	1354
33.11.5	Manipulator <i>setprecision(int)</i>	1354
33.11.6	Manipulator <i>std::setbase(int)</i>	1356
33.11.7	Manipulatory <i>setiosflags</i> , <i>resetiosflags</i>	1357

33.11.8	Tabele z zestawieniem manipulatorów	1357
33.12	Definiowanie swoich manipulatorów	1359
33.12.1	Manipulator jako funkcja	1359
33.12.2	Definiowanie manipulatora z argumentem	1361
33.13	Zmiana sposobu formatowania funkcjami <i>setf</i> , <i>unsetf</i>	1364
33.14	Dodatkowe funkcje do zmiany parametrów formatowania	1370
33.14.1	Funkcja <i>width</i>	1371
33.14.2	Funkcja składowa <i>fill</i>	1372
33.14.3	Funkcja <i>precision</i>	1373
33.14.4	Funkcja <i>copyfmt</i>	1374
33.15	Nieformatowane operacje wejścia/wyjścia	1374
33.16	Omówienie funkcji wyjmujących ze strumienia.....	1376
33.16.1	Funkcje do pracy ze znakami i napisami.....	1376
33.16.2	Wczytywanie binarne – funkcja <i>read</i>	1382
33.16.3	Funkcja <i>ignore</i>	1383
33.16.4	Pożyteczne funkcje pomocnicze	1385
33.16.5	Funkcje wstawiające do strumienia.....	1387
33.17	Ćwiczenia	1389

34 Operacje we/wy na plikach.....1394

34.1	Strumienie płynące do lub od plików	1394
34.1.1	Otwieranie i zamykanie strumienia.....	1396
34.2	Błędy w trakcie pracy strumienia	1401
34.2.1	Flagi stanu błędu strumienia	1401
34.2.2	Funkcje do pracy na flagach błędu.....	1402
34.2.3	Kilka udogodnień dla sprawdzania poprawności	1403
34.2.4	Ustawianie i kasowanie flag błędu strumienia	1404
34.2.5	Trzy plagi, czyli „gotowiec”, jak radzić sobie z błędami	1408
34.3	Przykład programu pracującego na plikach	1412
34.4	Przykład programu zapisującego dane tekstowo i binarnie	1414
34.4.1	Zapis w trybie tekstowym	1418
34.4.2	Odczyt z pliku tekstowego	1419
34.4.3	Zapis danych w plikach binarnych.....	1421
34.4.4	Odczyt danych z pliku binarnego.....	1422
34.5	Strumienie a technika rzucania wyjątków	1424
34.6	Wybór miejsca czytania lub pisania w pliku	1428
34.6.1	Funkcje składowe informujące o pozycji wskaźników	1429
34.6.2	Wybrane funkcje składowe do pozycjonowania wskaźników	1429
34.7	Pozycjonowanie w przykładzie większego programu	1432
34.8	Tie – harmonijna praca dwóch strumieni.....	1438
34.9	Ćwiczenia	1440

35 Operacje we/wy na stringach.....1443

35.1	Strumień zapisujący do obiektu klasy <i>string</i>	1443
35.1.1	Przykłady ilustrujące użycie klasy <i>ostream</i>	1447
35.2	Strumień czytający z obiektu klasy <i>string</i>	1450
35.2.1	Prosty przykład użycia strumienia <i>istream</i>	1452
35.2.2	Strumień <i>istream</i> a wczytywanie parametrów-danych	1455
35.2.3	Wczytywanie argumentów wywołania programu	1460
35.3	Ożenek: strumień <i>stringstream</i> czytający i zapisujący do stringu	1464
35.3.1	Przykładowy program posługujący się klasą <i>stringstream</i>	1465
35.4	Ćwiczenia	1469

36 Projektowanie programów orientowanych obiektowo1471

36.1	Przegląd kilku technik programowania	1471
36.1.1	Programowanie liniowe (linearne)	1472
36.1.2	Programowanie proceduralne (czyli „orientowane funkcyjnie”)	1472
36.1.3	Programowanie z ukrywaniem (zgrupowaniem) danych	1472
36.1.4	Programowanie obiektowe – programowanie bazujące na obiektach	1473
36.1.5	Programowanie obiektowo orientowane (OO).....	1473
36.2	O wyższości programowania OO nad Świętami Wielkiej Nocy	1474
36.3	Obiektowo orientowane: projektowanie	1477
36.4	Praktyczne wskazówki dotyczące projektowania programu techniką OO.....	1478
36.4.1	Rekonesans, czyli rozpoznanie zagadnienia.....	1479
36.4.2	Faza projektowania	1479
36.4.3	Etap 1. Identyfikacja zachowań systemu.....	1481
36.4.4	Etap 2. Identyfikacja obiektów (klas obiektów).....	1481
36.4.5	Etap 3. Usystematyzowanie klas obiektów	1483
36.4.6	Etap 4. Określenie wzajemnych zależności klas	1484
36.4.7	Etap 5. Składanie modelu. Sekwencje działań obiektów i cykle życiowe	1486
36.5	Faza implementacji.....	1487
36.6	Przykład projektowania	1487
36.7	Rozpoznanie naszego zagadnienia.....	1488
36.8	Projektowanie	1492
36.8.1	Etap 1. Identyfikacja zachowań naszego systemu.....	1492
36.8.2	Etap 2. Identyfikacja klas obiektów, z którymi mamy do czynienia.....	1493
36.8.3	Etap 3. Usystematyzowanie klas obiektów z naszego systemu.....	1496
36.8.4	Etap 4. Określamy wzajemne zależności klas	1498
36.8.5	Etap 5. Składamy model naszego systemu.....	1500
36.9	Implementacja modelu naszego systemu	1505

37 Szablony – programowanie uogólnione1513

37.1	Definiowanie szablonu klas.....	1514
37.2	Prosty program z szablonem klas	1516
37.2.1	Ostrożnie z referencją jako parametrem aktualnym	1518
37.3	Szablon do produkcji funkcji.....	1519
37.4	Cudów nie ma. Sorry.....	1523
37.5	Jak rozmieszczać w plikach szablony klas?.....	1524
37.6	Tylko dla orłów	1525
37.7	Szablony klas, drugie starcie	1525
37.8	Co może być parametrem szablonu – zwiastun	1526
37.9	Rozbudowany przykład z szablonem klas	1526
37.9.1	Definiowanie funkcji składowych szablonu klas	1531
37.9.2	Składniki statyczne w szablonie klasy	1532
37.9.3	Obiekt klasy szablonowej tworzony operatorem <i>new</i>	1534
37.9.4	Dyrektywa <i>using</i> składnikiem szablonu klas	1535
37.9.5	Przeładowany operator << w szablonie klas	1537
37.9.6	Jawne wywołanie destruktoru klasy szablonowej.....	1538
37.10	Reguła SFINAE.....	1539
37.11	Kiedy kompilator sięga po nasz szablon klas?.....	1543
37.12	Co może być parametrem szablonu? Szczegóły	1544
37.13	Parametry domniemane	1553
37.13.1	Szablon klas z domniemanymi parametrami.....	1553
37.13.2	Domniemane parametry w szablonie funkcji	1554
37.14	Zagnieżdżenie a szablony	1556
37.14.1	Szablon funkcji składowych zagnieżdżony w szablonie klasy	1557
37.14.2	Szablon klasy zagnieżdżony w zwykłej klasie.....	1563

37.14.3	Szablon klasy z zagnieżdżoną definicją klasy	1565
37.15	Poradnik: jak pisać deklaracje przyjaźni w świecie szablonów	1567
37.15.1	Szablon obdarza przyjaźnią swój parametr	1573
37.16	Użytkownik sam może specjalizować szablon klas	1574
37.16.1	Kompletna (zupełna) specjalizacja szablonu klasy	1577
37.16.2	Częściowa specjalizacja szablonu klasy	1579
37.16.3	Częściowa specjalizacja pozwala wybrać parametry będące wskaźnikami	1581
37.17	Specjalizacja funkcji składowej szablonu klas	1585
37.18	Specjalizacja użytkownika szablonu funkcji	1587
37.19	Ćwiczenia	1589

38 Posłowie1595

38.1	Per C++ ad astra	1595
------	------------------------	------

A Dodatek: Systemy liczenia1597

A.1	Dlaczego komputer nie liczy tak jak my?	1597
A.2	System szesnastkowy (heksadecymalny)	1603
A.3	Ćwiczenia	1605

B Skorowidz1607



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 krzychałby 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 x, y;
    // 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 prostokąta 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 zewnatrz " << 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 zewnatrz 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:

- a) skorzystać z właśnie obowiązujących (wewnątrz deklaracji tej klasy) instrukcji using lub typedef,
- b) skorzystać ze zdefiniowanych w tej klasie typów wyczerpujących enum.

Nie zawsze można takim chwytem się posłużyć. Aby to było możliwe:

- ❖ Klasa, w której ma się to zdarzyć, nie może być tak zwaną *klasą lokalną*.
Dotychczasowe nasze klasy były nielocalne (czyli zwykłe), tutaj więc nie ma przeszkód. (O klasach lokalnych porozmawiamy na str. 737).
- ❖ Tak definiowana funkcja zaprzyjaźniona nie może mieć nazwy z kwalifikatorem zakresu twierdzącej, że jest ona jakoby funkcją składową jakiejś innej klasy, podczas gdy naprawdę takiej funkcji składowej w tamtej klasie nie ma.
To by były wręcz podchody! Wyobraź sobie: ktoś zdefiniował klasę i jest z niej dumny, a my – za pomocą deklaracji przyjaźni w naszej klasie – cichaczem dodefiniujemy tej jego klasie piętnaście dodatkowych funkcji składowych!

Jeśli skorzystamy z tej możliwości, aby w klasie K, przy okazji deklaracji przyjaźni ze zwykłą funkcją f, od razu umieścić także definicję tej funkcji zaprzyjaźnionej f, to pamiętajmy, że wtedy ta funkcja nie jest widoczna na zewnątrz klasy K. Aby tak było, trzeba ją wcześniej deklarować:



```
void rezydent(); ❶
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////
class K
{
    friend void rezydent() { cout << „Jestem rezydent funkcja” << endl; }
};
///////////////////////////////////////////////////////////////////////////////////////////////////////////////////
int main()
{
    rezydent(); ❷
}
```

Zatem bez deklaracji **❶** kompilator w miejscu **❷** wykryje błąd, mówiąc, że nazwa rezydent nie jest mu znana. Wniosek: w każdej sytuacji najlepiej, żeby funkcja, którą klasa ma obdarzyć przyjaźnią, była już wcześniej zadeklarowana.



Przy przeładowaniu

- ❖ W przypadku funkcji przeładowanych przyjaciелеm klasy K jest tylko ta wersja funkcji, która odpowiada liście argumentów widocznej w deklaracji przyjaźni w definicji danej klasy K.

```
class K
{
    //...
    friend void alarm(K obj, int k);
};

void alarm(float*, K obiekt);
void alarm(void);
void alarm(K obiekt, int i); // to jest przyjaciel klasy K
```

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 `sedzia` 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 sedzia (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::sedzia(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 przyjazni. 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 przyjazni.
- 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 rzeczywiście 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. 957).

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 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. 701).

|| 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;

Przechodność 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 szafu 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. 957).

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) 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 `mutable`. W klasie K chcemy zamieścić deklarację 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 270
 > a przeładowanie 960
 ## sklejacz 275
 > a przeładowanie 960
 #define 270-271
 #elif 279
 #else 278
 #endif 277
 #error 280
 #if 277
 #ifdef 279
 #ifndef 280
 #include 281-282
 #line 281
 #pragma 283
 #undef 272
 % (operator modulo) 120
 & operator iloczynu bitowego 130
 & operator pobrania adresu 357
 && 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 913
 > a zasłanianie 86
 __STDC__ 285
 __STDC_HOSTED__ 285
 __func__ 285
 __STDC__ 285
 __cplusplus 285

__DATE__ 284
 __LINE__ 284
 __NAME__ 284
 __TIME__ 284

A

abstrakcyjna klasa 1294-1300, 1483
 adjustfield maska, pole 1341
 adres
 > 0 zero (dawniej zwany NULL) 368
 > funkcji 454
 • a jej nazwa 448
 • przeładowanej 488-491
 > funkcji szablonowej 1522
 > nullptr (zerowy) 366
 > tablicy 294
 > zamiana go na liczbę całkowitą 363
 agregat 886, 895
 > klasa 582-584
 > tablica to agregat 582-584
 aktualny
 > argument funkcji 184
 > parametr szablonu 1515
 alarm (znak specjalny) 69
 algorytm
 > w postaci szabł. funkcji 1127
 > zliczający 1118
 alias 102, 464
 alignas 113-114, 139
 alignment 115
 alignof operator 139-140
 > a przeładowanie 960
 alokacja (rezerwacja) tablic 391
 alternatywa (operacja logiczna) 125
 alternatywna
 > dekl. szablonowej f-cji. składowej 1536
 > deklaracja funkcji 181-182, 1536
 ampersand 294
 analiza zachowań obiektów 1480
 anonimowa

- › przestrzeń nazw 223
- › unia 1090-1091
- ANSI C 5
- apostrof 69
- app, tryb otwarcia pliku 1396
- argc 442
- argument
 - › aktualny funkcji 184
 - › będący obiektem 545-547
 - › będący tablicą 385
 - › będący wskaźnikiem do funkcji 455-458
 - › domniemany 199-206
 - funkcji składowej 542
 - kolejne definiowane „na raty” 203
 - a zakres ważności 204
 - › formalny funkcji 184
 - › formalny, a aktualny 184
 - › funkcji
 - będący tablicą 293-296
 - jego nazwa 176
 - referencją lwartości 190-197
 - referencją rwartości 192, 195
 - typu initializer_list 811
 - › identyczny czy różny dla przeladowania 480-487
 - › nienazwany 207
 - › przesyłanie przez referencję 185-187
 - › przesyłanie przez wartość 184
 - › wskaźnikiem do const 387
 - › wywołania funkcji 184
 - › wywołania programu 441-443, 1460
- argumentowość operatora 961
- argv 442
- arytmetyczny
 - › operator 119-123
 - › typ 49
- ASCII kod 69, 300
- at - f. w kl. std::string 624-628
- ate, tryb otwarcia pliku 1396
- atof - fun. bibl. (Ascii To Float) 444, 635
- auto 106-108, 179
 - › a const 421-425
 - › a gwiazdka wskaźnika 424
 - › a wskaźnik 360
 - › a const i volatile 253
 - › definicja referencji 253-262
 - › użyte wobec wyr. lambda 1148
 - › we wskaźniku do klasy szabł. 1534
 - › a wskaźnik do f. składowej 913
 - › a wskaźnik do skł. klasy 903
 - › z operatorem new 396
- automatyczny obiekt 213
- awans 495

B

- o babci przypowieść 185
- back - f. składowa string:: 629
- backslash 69
- backspace (znak specjalny) 69
- bad - f. sprawdzająca stan strumienia 1403
- bad_alloc - klasa wyjątku 410, 412, 1064
- bad_cast - klasa wyjątku 1309
- badbit (flaga stanu strumienia) 1401-1402
- basefield, maska, pole 1368
- begin - f. ustawiająca iterator std::stringu 672
- begin - funkcja
 - › w klasie pojemnika 1535
 - › zastosowanie w szabł. 1536
- bekslesz 70
- beta rozpad 745
- bezpośrednia kl. podstawowa 1187
- białe znaki 9
- biblioteczna funkcja 237-239
- biblioteka
 - › a przestrzeń nazw 80
 - › iostream 1320
 - › standardowa 610
 - › standardowych strumieni we/wy 1320
 - › stdio 1320
- binarne wczytywanie 1382
- binarny system liczenia 1599
- binarny tryb
 - › odczytu pliku 1422
 - › zapisu pliku 1421
- ios::binary, tryb otwarcia pliku 1396
- bit 127, 1601
 - › najbardziej znaczący 1601
 - › najmniej znaczący 1601
- bit po bicie 1096, 1197
- bitowy operator 127-129
- bliższe pokrewieństwo bez znaczenia 1218
- blok
 - › catch 712
 - › funkcji 79
 - › instrukcji 23
 - › lokalny 78
 - › try 711
- bool() operator (w strumieniach) 1404
- bool, stałe dosłowne tego typu 63
- bool, typ 21, 48
- boolalpha manipulator 1346
 - › zastosowanie 1076
- boolalpha, flaga 1341
- break 32
- „bryk” czyli spis funkcji kl. std::string 681-688
- buforowanie strumienia 1324, 1348

błędy pracy strumienia 1401-1411

C

C klasyczny 5

C-string 71-72, 609, 612

- › argumentem funkcji 434
- › bardzo długi 72
- › długość, a rozmiar 302
- › jego typ 72
- › konkatencja 440
- › w cudzysłowie - (jest jako static) 441
- › wariacje na temat 434-440
- › zapis w kilku liniach 72
- › ze znaków wchar_t 73

c_str - f. w kl. std::string 650-652

callable object 1154

capacity - f. w kl. std::string 619

carriage return (znak specjalny) 69

case 32

- › a constexpr 34

catch 411-412

- › a lista inicjalizacyjna konstruktora 796
- › blok 712
- › dla listy inicj. konstruktora 795
- › kolejność stawiania bloków 714
- › wszystkożerny 729

cctype (nagłówek) 661

cecha i wykładnik 1344, 1374

cechowanie aparatury 842

CERN ośrodek badań fizyki cząstek 1282

cerr 1323

char16_t 47, 611

char32_t 47, 611

charakterystyka

- › konstruktora 1225
- › w specyfikacji wyjątków 1225

Church Alonzo 1133

chwilowy obiekt 547

ciało

- › funkcji 8, 175
- › klasy 505

cin 17, 1323

ciąg znaków 71

class, a typename (szablony) 1520

clear - f. w kl. std::string 624

clear - f. w kl. strumienia 1405

clog 1323

close, funkcja w kl. strumienia 1399

compare - funkcja w kl. std::string 653

const 87

- › a konstruktor 762
- › a przeładowanie 581
- › funkcja składowa 577-581

› głębokie 420, 486

› obiekt, a wskaźniki 365

› obiekty 87

› przy wysłaniu argumentu do funkcji 387

› wierzchnie 420, 485

› wskaźnik 418

› wskaźnik do takiego obiektu 419-420

const_cast 142, 145, 428-431

› a przeładowanie 960

› a volatile 432

constant expression 89

constexpr 88-91

› a lista wyliczeniowa 837

› a pola bitowe 836

› a switch case 34

› funkcja tego typu 240-252

› funkcja, a inline 245

› konstruktor 828-837

› obiekty 88-91

› vs. #define 272

continue 39

copy - f. w kl. std::string 662

copy elision 849

copyfmt, funkcja 1374

count_if - algorytm biblioteczny 1127

cout 8, 1323

covariant 1268

cyfra 1597

cykl życiowy obiektu 1486

czas życia 393

› a zakres ważności 78-84

› obiektu 776

• co to jest 78

• globalnego 775

• lokalnego 774

• wybór tego czasu 212-217

• wytworzonego przez new 775

czteropak 1530

czysto wirtualna funkcja 1297, 1299

czytanie deklaracji 446

częściowa specjalizacja 1579

› a ref. do lwartości, rwartości 1584

› zastosowanie 1581

D

dana składowa 506

› publiczna - odwołanie się 525

data() - f. w klasie std::string 649

debugger 10, 210, 1348, 1413

dec, flaga 1339-1341

dec, manipulator 1347, 1356

decltype 109-110

- › w alternatywnej dekl. fun. 182
- › w deklaracji funkcji 182
- › operator - pełny opis 876-880
- › typu wsk. f-cji 466-467
- › a wskaźnik do skł. klasy 908
- › a wskaźnik do f. składowej 913
- default
 - › etykieta (w switch) 33
 - › w deklaracji funkcji (=default) 784
- defaultfloat manipulator 1350
- #define
 - › vs. constexpr 272
- defined 277
- definicja 16, 44, 218, 221
 - › a deklaracja 16, 45
 - › funkcji 175
 - składowej szablonu kl. 1531
 - zaprzyjżnionej będąca w klasie 701
 - › klasy 505
 - › klasy zagnieżdżonej 737-743
 - › lokalna klasy 755-757
 - › obiektów klasy string 612-616
 - › po łacinie 45
 - › przeładowanego operatora 959
 - › składnika statycznego 558
 - › stałej za pomocą #define 272
 - › szablonu klas 1514-1515
 - › w wyrażeniu
 - inicjalizującym pętli for 61
 - warunkowym instr. if 61
 - warunkowym instr. while 62
 - › wskaźnika 356
 - › „w biegu“ 60-61
- deklaracja 16, 218, 221
 - › a definicja 16, 45
 - › czytanie jej 446
 - › dostępu 1182, 66
 - › dostępu using 1182
 - › funkcji 175
 - alternatywna 181-182
 - nieobowiązkowa 198
 - › po łacinie 45
 - › przyjaźni 697
 - › typedef 93-95
 - › using 84, 93-95, 1182
 - › zapowiadająca 700, 1213
- dekorator wewnątrz 761
- dekrementacji operator 121
- delegat 801
- delegowania relacja 1480
- delegowanie łańcuchowe 802
- delegujący konstruktor 797-803
- delete 392
 - › a adres zerowy (nullptr) 780
 - › a możliwy polimorfizm 1079
 - › niemożliwe virtual 1066
 - › operator 399
 - › przeładowanie 1061-1087
 - › standardowe, jego wywołanie 1073
 - › tablicy
 - przeładowanie 1079
 - › w deklaracji funkcji (=delete) 786
- destrukcja obiektu złożonego 824
- destruktor 553-556, 761-840
 - › a const i volatile 780
 - › a dziedziczenie 1186
 - › a przeładowanie 780
 - › a rzucanie wyjątków 782
 - › a dziedziczenie 1186
 - › jawne wywołanie 781
 - › klasy szabł., jego wywołanie 1538
 - › kolejność ich pracy 824
 - › kolejność wywołania 1190-1195
 - › szablonu klas 1532
 - › wirtualny 1277-1278
 - › wywołanie z this 781
 - › zastosowania 556, 779
- deszyfrowanie słów 1107-1113
- detektor - stoper 746
- Deutsche Oper (ilustr. dziedziczenia prywatnego) 1231
- długość, a rozmiar C-stringu 302
- do... while... 27
- domeny zastosowania wskaźników 370-417
- dominacja klas wirtualnych 1249
- domniemane-
 - › argument 199-206
 - › argument „na raty” 203
 - › argument, a przeładowanie 475
 - › konstruktor 783
 - › parametr szablonu 1553-1555
 - › parametr w szablonie f-cji 1554
 - › parametr w szablonie klas 1553
 - › wartość będąca wyrażeniem lambda 1162-1165
- dopasowanie
 - › a konwersja 949-953
 - › z awansem 495
 - › etapy 493-498
 - › funkcji przeładowanych 492
 - › z promocją 495
- dopasowanie dokładne 493
- Dopplera zjawisko 914
- dorzecze, schemat konwersji 954
- dostęp
 - › a dwuznaczność 952

- › do skł. odziedziczonych 1177-1184
- › do składników klasy 509-511
- › private 510
- › protected 510
- › public 511
- › wybiórczo 1182, 66

dostępu

- › deklaracja 1182, 66
- › deklaracja using 1182
- › specyfikator 1175

dosłowne stałe 62-75

dosłowny typ 585

double, typ 48

duża tablica, przykład 1432-1437

dwuargumentowy operator 119

dwuznaczność, a dostęp 952

dwójkowy

- › kod 1601
- › system liczenia 1598

dynamic_cast 142, 146

- › a przeładowanie 960
- › a typy polimorficzne 1307-1309

dynamiczna alokacja (rezerwacja) tablicy 391-413

dynastii koniec 1177

dyrektywa

- › preprocesora 270
- › pusta 270
- › using 83

dzieci z próbowki (in vitro) 1523

dziedziczenie 1174-1256

- › a inicjalizacja 1196-1197
- › a operatory we/wy 1334
- › a przypisanie 1196-1197
- › a wieloznaczność 1216
- › a zawieranie 1226-1227
- › czego się nie dziedziczy 1185-1186
- › graf dziedziczenia 1187
- › jednokrotne 1212
- › kilkupokoleniowe 1187
- › konstruktorów (sposób na) 1219-1225
- › obiektów - nie istnieje 1177
- › od kilku rodziców 1212-1218
- › operatorów 1029
- › prywatne 1231
 - kiedy 1181
 - np. Deutsche Oper 1231
- › prywatne, kiedy? 1182, 1188
- › wielodziedziczenie 1212-1218
- › wielokrotne 1212
- › wielopokoleniowe 1188
- › wielorakie 1212
- › wirtualne 1241-1249

- › zakresów zagnieżdżanie 1175

dzielenie z resztą 120

dziesiątkowy system liczenia 1597

E

early binding 1272

edytor tekstu programu 10

egzotyczne izotopy 744

Tekran_alfanumeryczny

- › klasa przykładowa 763

ekran, urządzenie wyjściowe 1321

eksplozja, schemat konwersji 954

else 22-25

empty() - f. skł. kl. std::string 619

encapsulation 508

end - funkcja

- › iteratora kl. std::string 675
- › w szablonie klasy 1536
 - zastosowanie 1536

endl, manipulator 18, 1348

ends, manipulator 1348

enkapsulacja 1181, 1188, 1485

enum 46, 96-105

- › a przeładowanie 481, 988
- › enum class 97
- › enum class konwersja na int 544
- › tablica takich elementów 297-298
- › w klasie 542
- › w operatorze wej/wyj 1333
- › zwykłe, a enum class 103

EOF 1377, 1384

eof() - fun. sprawdzająca stan strumienia 1403

eofbit (flaga stanu strumienia) 1401-1402

erase - f. w kl. std::string 643

etykieta 37

- › (prawdziwa, czyli nie-case, nie-default) 32
- › case 32
- › default 33
- › dostępu 510
- › gdzie jest znana 79
- › private 509-511
- › protected 509-511, 1179
- › public 509-511
- › zakres ważności 79, 212

exception (klasa std::exception) 1424

exception handling 710-736

exceptions, funkcja 1424

EXCLUSIVE OR operator 130

explicit 773, 852

- › a konstr. konwertujący 930
- › konstruktor 773
- › przy operatorze konwersji 944

extensibility (cecha języka C++) 1267, 1475
 extern 45, 218, 222
 extract we/wyj operator 1324

F

Fahrenheita skala temperatur 996
 fail() - fun. spr. stan strumienia 1403
 failbit (flaga stanu strumienia) 1401-1402
 failure, klasa ios_base::failure 1424
 false 48, 63
 fałsz 22, 48
 fałsz-prawda 20-21
 fill - f. w klasie strumienia 1354, 1372
 final - słowo kluczowe kontekstowe

- › jako koniec dziedziczenia klasy 1177
- › jako koniec ulepszeń f. wirtualnych 1281-1293

 find - f. skł w kl. std::string 637-639
 find_first_not_of 641
 find_first_of 641
 find_last_not_of 641
 find_last_of 641
 fixed, flaga 1340, 1343
 fixed, manipulator 1350
 flaga

- › skąd określenie 1338
- › stanu błędu strumienia 1401
- › stanu formatowania 1338-1344

 flags(fmtflags), f. we/wy 1365, 1369
 float, typ 48
 flush, manipulator 1348
 fmtflags, typ 1366
 for

- › instrukcja pętli (zwykła) 28-29
- › zakresowe for 169-171, 629

 for_each - algorytm biblioteczny 1149
 form feed (znak specjalny) 69
 formalny

- › argument funkcji 184
- › parametr szablonu 1515

 format operacji we/wy 1338-1344
 formatowanie informacji 1322
 formatowanie wewnętrzne 1443-1468
 free store, (heap) 395, 413, 884
 free()

- › fun. bibl. do zwalniania pamięci 1070

 front - f. składowa string:: 629
 frytki (jak pola bitowe) 1108
 fundamentalny typ 46, 76
 funkcja 174-269, 1472

- › adres f. przeladowanej 488-491
- › alternatywna deklaracja 181-182
- › argument aktualny 184
- › argument będący wskaźnikiem 381-389
- › argument formalny 184
- › argument wskaźnikiem do const 387
- › biblioteczna 237-239
- › ciało 8
- › constexpr 240-252
- › constexpr, a inline 245
- › czysto wirtualna 1297
- › deklaracja 175
- › domniemane przesyłanie obiektów 547
- › dwa sposoby wysyłania jej tablicy 385
- › inline 208-211
- › inline, a szablony 1524
- › jej blok 79
- › jej definicja 175
- › jej nazwa 175, 448
- › jej sygnatura 1268
- › jej szablon 1519-1522
- › konwertująca 930, 938-944
- › main 8
- › nawiasy w wywołaniu 448
- › operatorowa jako przyjaciel 967
- › operatorowa jako składowa 964-966
- › orzekająca 457, 1121
 - a obiekt funkcyjny 1130
- › outline 210
- › przekazywanie jej tablicy 293-296
- › przeladowanie jej nazwy 470, 492, 1279-1280
- › przesyłanie argumentu przez wartość 184
- › przesyłanie obiektów przez referencję 547
- › rekurencyjna 228-236
- › rekurencyjna, a constexpr 248
- › rezultat referencją lwartości 223-227
- › rozmieszczenie tychże w kilku plikach 218-222
- › składowa 506-507, 514, 517-522
 - a arg. domniemany 542
 - const 577-581
 - const, a constexpr 590
 - constexpr 585-590
 - definiowanie jej 518
 - inline 520
 - specjalizowana 1585-1586
 - specjalna 1038, 1101
 - specjalne 1097
 - static 965
 - statyczna 566-576
 - szablonu klasy 1531
 - volatile 577-581
 - wskaźn. do niej 911-919
 - wywołanie dla obiektu 517
 - wywołanie dla referencji 518
 - wywołanie dla wskaźnika 518
 - wywołanie jej z listy inicjalizacyjnej 793

- › szablonowa 1520
 - generacja jej (kiedy) 1522
 - › wirtualna 1257-1319, 1500
 - dostęp do niej 1271
 - inline 1277
 - a przyjaźń 1272
 - a static 1272
 - w klasie pochodnej 1271
 - › wskaźnik do f. 446-469
 - › wysłanie do niej elementu tablicy 297, 434, 545-547
 - › z wielokropkiem, dopasowanie 499
 - › zaciera ślad innej f. wirtualnej 1279
 - › zaprzyjaźniona 696-709
 - a wirtualność 1272
 - zdefiniowana w klasie 701
 - › zwracanie rezultatu 177-180
- funkcja we/wy
- › bad() 1403
 - › clear(io_state) 1405
 - › eof() 1403
 - › fail() 1403
 - › fill(char) 1354, 1372
 - › flags(fmtflags) 1369
 - › gcount() 1385
 - › good() 1402
 - › ignore 1383
 - › open(char*, int, int) 1396
 - › peek() 1385
 - › precision(int) 1373
 - › put(char) 1387
 - › rdstate 1405
 - › read(char*, int) 1382
 - › setf 1357
 - › setf(fmtflags) 1364-1369
 - › tellg() 1429
 - › tellp() 1429
 - › unsetf 1357
 - › unsetf(fmtflags) 1364-1369
 - › width(int) 1351, 1371
 - › write(const char*, int) 1388
- funkcyjny
- › obiekt 1123
 - a lambda 1119
 - a przeladowanie operatora() 1045
- funktor (czyli obiekt funkcyjny) 1045, 1124

G

- garbage collector 1062
 gcount, funkcja 1385
 generowany
- › bo przypisek =default 784

- › destruktor 781
 - › f-cji skł. nie będzie gdy przypisek =delete 786
 - › konstruktor 582
 - › konstruktor domniemany 783
 - › konstruktor kl. pochodnej 1198
 - › konstruktor kopiujący 842, 851
 - › konstruktor przenoszący 871
 - › mechanizm kopiowania 1038
 - › operator przypisania 1004, 1015, 1029
- get from 1324
- getline
- › a std::string 663-669
 - › pułapka 666
- głębokie const 486
- globalne
- › nazwa 80
 - › obiekt 212
 - › obiekt, a klasa 529
 - › obiekt, jego inicjalizacja 217
 - › wskaźnik 366
 - › zmienna 1472
- glwartość 873-875
- good() - f. spr. stan strumienia 1402
- goodbit (flaga stanu strumienia) 1401
- goto 37-38, 79, 1472
- gotowiec - jak radzić sobie z błędami 1408
- graf
- › dziedziczenia 1187
 - › współpracy klas 1484, 1498
- greg, gregis 582
- gwiazdolot - metafora wskaźnika 360
- głęboka kopia 858
- głębokie const 420

H

- heap 395
- heksadecymalny system liczenia 1603-1604
- hex, flaga 1339-1341
- hex, manipulator 1347, 1356
- hexfloat, manipulator 1350
- hierarchia 1189, 1232, 1480, 1483, 1497
- hybrydowość 1474

I

- identyfikacja
- › obiektów (klas obiektów) 1481
 - › zachowań systemu 1480-1481
- if - instrukcja sterująca 22-25
- ignore, f. strumienia 669, 1383
- iloczyn logiczny 125
- implantacja jonu 753
- implantacja jądra atomowego 745

- in - tryb otwarcia pliku 1396
- include 281-282
- inicjalizacja 59, 1017
 - > a klasy podst. wirtualne 1245
 - > agregatowa 583
 - > definicja 88
 - > konstruktorem 553
 - > konwersja w niej 948
 - > obiektu 549
 - > przy dziedziczeniu 1196-1197
 - > std::stringu 614
 - > tablicy 292
 - > tablicy obiektów jakiejś klasy 886-892
 - > unii 1090
 - > w klasie 514-516
 - > w klasie, a konstruktor 553
 - > w new
 - klamry { } i nawiasy () 397
 - > wektora wielowymiarowego 327
 - > zbiorcza 887
- inicjalizator
 - > klamrowy 583
 - > kopiujący 841
 - > przenoszący 866-872
- inicjalizatorów lista 616
- inicjalizujące wyrażenie 360
- initializer_list
 - > a vector 813
 - > argumentem funkcji 811
 - > co to jest 804-818
 - > w inicjalizacji wektora 895
 - > zastosow. w szabl. funkcji 1561
- inkrementacji operator 121
- inline 208-211
 - > a makrodefinicja, porównanie 274
 - > a wirtualność 1277
 - > funkcja składowa 520
 - > gdzie definiować 210
 - > takż przyjaciel 1333
 - > vs. makrodefinicja 274
- inne języki programowania - linkowanie 477
- insert - f. w kl. std::string 644-645
- instrukcja
 - > blok tychże 23
 - > kroku pętli 28
 - > składana 23
 - > sterująca 20-43
 - > throw 712
- 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) 1104, 1602
- internal, flaga 1339-1340
- internal, manipulator 1351
- invalid_argument (wyjątek) 633, 727
- ios:: zamiast ios_base:: 1367
- ios::app 1396, 1398
- ios::ate 1396, 1398
- ios::badbit (flaga stanu strumienia) 1401
- ios::basefield, maska, pole 1368
- ios::beg 1429
- ios::binary 1396, 1398
- ios::cur 1429
- ios::dec 1339-1340
- ios::end 1429
- ios::eofbit (flaga stanu strumienia) 1401
- ios::failbit (flaga stanu strumienia) 1401
- ios::fixed 1339-1340
- ios::goodbit (flaga stanu strumienia) 1401
- ios::hex 1339-1340
- ios::in 1396-1397
- ios::internal 1339-1340
- ios::left 1339-1340
- ios::nocreate (przestarzałe) 1400
- ios::noreplace (przestarzałe) 1400
- ios::oct 1339-1340
- ios::out 1396-1397
- ios::right 1339-1340
- ios::scientific 1339-1340
- ios::seek_dir 1429
- ios::showbase 1339-1340
- ios::showpoint 1339-1340
- ios::showpos 1339-1340
- ios::skipws 1339-1340
- ios::stdio 1339-1340
- ios::trunc 1396, 1398
- ios::unitbuf 1339-1340
- ios::uppercase 1339-1340
- ios_base::failure 1424
- iostream biblioteka 1321
- isdigit, f. biblioteczna 1386
- istringstream 1450-1463
- iteracja 672
- iterator
 - > do obiektu stałego 674

- › harcerska definicja 1535
 - › stringu 670-679
- izotopy 744

J

- jawna konwersja
- › jej zapis 948
 - › typy 142
- jawne wywołanie konstruktora 776
- jednoargumentowy operator 121
- jest rodzajem...
- › powód dziedziczenia 1226
- justowanie 1340
- język obiektowo orientowany 1471

K

- kalibracja 843
- kapsułowanie 508
- karty modelujące 1481
- kaskadowe przypisywanie 1024
- kasowanie tablicy w zapasie pamięci 886
- kdevelop 12, 541
- kilkupokoleniowe dziedziczenie 1187
- klamra { } pusta 111-112
- klamrowa lista inicjalizatorów 1561
- klamry
- › bloku instrukcji - jak stawiać 40
 - › { } 111-112
- klamry inicjalizacyjne
- › tablicy new 399
- klasa 504-608
- › a obiekt - różnica 512-513
 - › a typ 505
 - › abstrakcyjna 1294-1300, 1483
 - › agregat 582
 - › ciało 505
 - › definicja 505
 - › dominuje 1249
 - › enum w niej zdefiniowane 542
 - › literalna 829, 837
 - › lokalna 702, 755-757
 - › najbardziej pochodna 1246
 - › pochodna 1175
 - › podstawowa 1175
 - bezpośrednia 1187-1188
 - pośrednia 1187-1188
 - wirtualna 1241-1249
 - › polimorficzna 1265
 - › prywatna 826
 - › rozmieszczenie w plikach 530-544
 - › składniki 506
 - › składowa 737-743

- › std::string 156-160, 609-695
 - › szablonowa 1513, 1515
 - destruktora wywołanie 1538
 - kiedy powstaje 1543
 - klasą podstawową 1544
 - synonim jej nazwy 1534
 - › trywialna 1311
 - › uniopodobna 1092-1093
 - › uogólnienie jej 1514
 - › uogólniona 1190
 - › wskaźnik do jej obiektów 884
 - › o zagnieżdżonej definicji 737-743
 - › zaprzyjaźniona 705-706
- klasyfikacja C 5
- klauzura 1142
- klauzurowy obiekt 1142
- klawiatura 1321
- kod ASCII 69
- kod dwójkowy 1601
- kod źródłowy programu 11
- kod źródłowy przykładów z tej książki 7
- kolejka 1190
- › jako szablon 1513
- kolejność na liście inicjalizacyjnej 792
- komentarze 14
- komora drutowa 905
- kompilacja warunkowa 276-279
- › w przykładzie 1034
 - › zastosowanie 1537
- kompilator 11
- › optymalizacja pracy 778
- kompletna (zupełna) specjalizacja 1577
- komputer steruje pomiarami 963
- komórka pamięci adresowana wskaźnikiem 390
- konflikt nazw 80
- kongregacja 582
- kongres 582
- koniunkcja 125
- konkatenacja stringów 440
- konstrukcja obiektu z obiektami składowymi 819-825
- konstruktor 548-552, 761-840
- › a const i volatile 762
 - › a dziedziczenie 1185
 - › a static 762
 - › a virtual 762
 - › a const i volatile 762
 - › constexpr 828-837
 - › delegujący 797-803
 - › do konwersji 930-937
 - › domniemany 783, 1247
 - dla elementów tablicy 893
 - prywatny 828

- › dwa etapy pracy 789
- › explicit 773
- › generowany automatycznie 871
- › i new 775
- › z argumentem typu initializer_list 814
- › jawne wywołanie 776
- › klasy pochodnej 1191, 1213
- › klasy std::string 614
- › kolejność ich pracy 825
- › kolejność wywołania 1190-1195
- › konwertujący 930-937
 - a explicit 930
 - szablonowy 1561
- › kopiujący 841, 1017
 - dla obiektów const 850
 - generowany automatycznie 851
 - klasy pochodnej 1198-1211
 - niejawne wywołanie 842
 - użyty niejawnie 842
- › na cudzej liście inicjalizacyjnej 890
- › niby-wirtualny 1301-1306
- › niepubliczny 826-827
- › prywatny 1524
- › przenoszący 866-872
 - generowany 871
- › przeładowanie nazwy 551
- › przeładowany 761
- › pułapka przy domniemany 772
- › szablonu klasy 1531
- › wirtualności symulacja 1301-1306

konstruowanie obiektu złożonego 825

kontrakt 1485

konwersja 123, 928-956

- › a dopasowanie 949-953
- › argumentu funkcji 1232
- › argumentów operatora 1232
- › arytmetyczna 498
- › dwa warianty 945-946
- › jawnie 948
- › kaskadowo 950
- › konstruktorem 930-937
- › niejawna 930, 989
- › niejawnie zachodzi gdy 947
- › operator teźże 938-944
- › przy inicjalizacji 1232
- › referencji (przy dziedziczeniu) 498
- › referencji do klasy pochodnej 1228-1240
- › rezultatu funkcji 1232
- › rzutowaniem 939, 949
- › standardowa 143, 497
 - przy dziedziczeniu 1228-1240
 - wskaźnika do składnika klasy 1239

- › trywialna 494
- › typu całkowitego 497
- › typów zmiennoprzecinkowych 497
- › wskaźnika do klasy pochodnej 1228-1240
- › wskaźników 498
- › wywołanie funkcji 939, 948
 - › zawiązująca 60

konwertujący konstruktor 930-937

kopia głęboka 858

kopia płytka 857

kopiujący

- › konstruktor 841
- › operator przypisania 1023

kowariant 1268

końcówkowy operator 984

kryterium

- › oceniania 1118-1131
- › orzekające 1118
- › w obiekcie funkcyjnym 1123
- › z parametrem 1125

król Midas (metafora kopiowania) 1027

kwifikator zakresu 542, 738, 1176

kwifikowana nazwa 738

- › składnika 1183
- › zależna 1536

L

łączność 151, 961

lambda (wyrażenie)

- › a noexcept 1143
- › a rekurencja 1158-1161
- › a słowo auto 1148
- › a wskaźnik this 1143-1146
- › bez nazwy (beziemienne) 1150
- › ciało 1138
- › jako domniemana wartość arg. 1162-1165
- › lista argumentów 1138
- › mające swą nazwę 1147-1154
- › nazwane, mające nazwę 1150
- › rzucające wyjątek 1166-1169
- › typ rezultatu 1139
- › w obiekcie 1150

Lambda rachunek (Alonzo Church) 1133

łańcuchowe łączenie delegowania 802

late binding 1273

least significant bit 1601

left, flaga 1339-1340

left, manipulator 1351

leksykalny zakres 701, 743

length - f. string:: 618

lewostronnie łączny operator 993

liczba atomowa 745

liczba przeciwna 121
 liczba w tekście stringu 630-631
 liczby zespolone 928, 957-958
 liczenia systemy 1597-1606
 likwidacja obiektu 553, 779
 liniowe (linearne) programowanie 1472
 linker 11, 218, 392, 477, 519
 linkowanie 11, 477

- › szablonów klas 1524
- › z innymi językami 477

 lista

- › inicjalizacyjna 786-796, 1190-1195
 - a lista inicjalizatorów - to co innego 887
 - kolejność wykonywania 792
 - wyw. z niej funkcji składowej 793
- › inicjalizatorów
 - a lista inicjalizacyjna - to co innego 887
 - a na niej wywołania konstruktorów 890
 - klamrowa 616
- › klamrowa (inicjalizatorów) 804-818
- › param. formalnych 1519
- › pochodzenia klasy 1175, 1182, 1212
 - a rodzaj dziedziczenia 1180

 lista wychwytywania

- › wyrażenia lambda 1140

 lista wyczeniowa 97

- › a constexpr 837

 literalna klasa 837
 literalny typ 574, 585, 837
 literał klasy 828
 logiczny operator 124-126
 lokalne -

- › klasa 702
 - jej definicja 755-757
 - jej zakres leksykalny 756
- › nazwa typu (typedef) 758
- › obiekt
 - a wstępna inicjalizacja 217
 - automatyczny 217, 774
 - statyczny 217, 775
- › zakres ważności 78
- › zmienna 1472

 long double, typ 48
 long int, typ 46
 long long int, typ 46
 long long, typ 46
 long, typ 46
 LSB 1601
 lvalue 361

- › Zob. lwartość

 lwartość 625, 1041

- › co to jest 188-189
- › referencja do niej 190-197

M

magiczne oko 592
 Magnetyczny Rezonans Jądrowy 648
 main 8

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

 makrodefinicja 273-274

- › a przeładowanie 274
- › a rekurencja 274
- › a szablon funkcji 275
- › i nawiasy 275
- › rozwinięcie 273
- › vs. funkcja inline 274

 malloc - fun. bibl. alokująca pamięć 1068
 manipulator 1345-1356

- › bezargumentowy 1346
- › boolalpha, zastosowanie 1076
- › dec 1347
- › defaultfloat 1350
- › definiowanie przez użytkownika 1357-1363
- › endl 1348
- › ends 1348
- › flush 768, 1348
- › hex 1347
- › hexfloat 1350
- › oct 1347
- › precision(int) 1354
- › predefiniowany 1346
- › resetiosflags(ftmflags) 1357
- › setbase(int) 1356
- › setfill 1354
- › setiosflags(fmtflags) 1357
- › setw 614, 1351
 - › z argumentem 1351

 martwa referencja 225, 1155-1157
 maszynka do funkcji 1523
 max_size() - f. w kl. std::string 619
 mechanizm

- › kopiowania 841, 873, 1015-1028, 1038
- › obsługi syt. wyjątkowych 710
- › przenoszenia 778
 - w klasie pochodnej 1208
- › specjalizacji szablonu (przez użytkownika) 1574
- › zwrotu rezultatu 842

 member function 523
 memberwise copy 851
 memcpy - f. biblioteczna 1310
 memory allocation 1068

- memset - f. biblioteczna 1113
 - menażeria 482, 484-485, 487-488
 - metoda (inna nazwa f. składowej) 523
 - metoda „odtąd dotąd” 676
 - metoda „płytkiej kopii” 1027
 - Midas, król 1027
 - mile na kilometry - przeliczanie 242
 - mnożenie (szybkie) 133
 - model
 - › składanie 1486
 - › widoczne własności 1487
 - › zachowania 1487
 - modelowanie 1475, 1477
 - modulo operator 120
 - moduły programu 218-222, 530-544
 - › z szablonem klas 1524
 - modyfikator
 - › zob. też: przydomek, specyfikator
 - › const 87
 - › constexpr 88-91
 - › explicit 773
 - › mutable 591-592
 - › register 92
 - › static 216
 - › volatile 92
 - most significant bit 1601
 - move 680, 862-863
 - › w klasie pochodnej 1210
 - MRJ 648
 - MSB 1601
 - mutable
 - › w klasie 591-592
 - › w wyrażeniu lambda 1142-1143
- N
- na oślepie strzał 366-367
 - nagłówka strażnik 282, 534
 - nagłówkowy plik 219
 - najbardziej pochodna klasa 1246
 - namespace 80, 222
 - › zob. też: przestrzeń nazw
 - › a przeładowanie 480
 - › przykład tworzenia swojego 722
 - › zakres 80
 - napis czyli stała tekstowa 71
 - naukowa notacja 1342
 - nawias pusty w deklaracji funkcji 176
 - nawiasy ostre 142, 1517
 - nazwa
 - › argumentu funkcji 176
 - › funkcji 175, 448, 454
 - › kl. szablonowej 1544
 - › konflikt 80
 - › nie jest obiektem 378
 - › obiektu 393
 - › statyczna globalna 222
 - › szablonu kl. unikalna 1525
 - › tablicy 294, 885
 - › tablicy, a wskaźnik 376
 - › typu, lokalna 758
 - › w C++ 15
 - › w funkcji, zakres ważności 212
 - › w klasie, zakres ważności 507
 - › zasłanianie 85-86, 526-529
 - nazwane obiekty lambda 1150
 - negacji operator ! 126
 - new 392
 - › a tablica wielowymiarowa 399
 - › auto (razem z) 396
 - › bad_alloc 412
 - › i konstruktor 775
 - › a inicjalizacja 396
 - › nadanie obiektowi wartości w momencie stworzenia 396
 - › niemożliwe virtual 1066
 - › nothrow 410
 - a adres nullptr 1080
 - › przeładowanie 1061-1087
 - › standardowe, jego wywołanie 1073
 - › tworzenie obiektu stałego 397
 - › umiejscawiający (operator) 402, 1076
 - › wstępna zawartość tak stworzonego obiektu 393
 - › wyjątku rzucenie 410
 - new line (znak specjalny) 13, 69
 - niebuforowany strumień 1324
 - nieformatowane operacje we/wy 1374-1375
 - niejawna konwersja 989
 - niekompletny typ 742
 - nienazwany argument 207
 - nieprostokątny wektor 2D 336-337
 - nieprostopadłościenny wektor 3D 348-351
 - nieruchomy wskaźnik 418, 420
 - nieskończona pętla 29
 - noboolalpha, manipulator 1346
 - nocreate, dawny tryb otwarcia 1400
 - noexcept 137
 - › a przeładowanie 960
 - › operator 731-733
 - › specyfikator 731-733
 - › w operatorze delete 1066
 - noreplace, dawny tryb otwarcia 1400
 - noshowbase, manipulator 1349
 - noshowpoint, manipulator 1349
 - noshowpos, manipulator 1349
 - noskipws, manipulator 1349

notacja

- › dziesiąta, flaga 1343
- › naukowa (wykładnicza) 1327, 1342
- › wykładnicza, „naukowa“, flaga 1343
- › wykładnicza, „naukowa“, wczytywanie 1327

nothrow 410

- › obiekt typu nothrow_t 1066
- › użyte w new 410

nothrow_t typ obiektu 1066

nounitbuf, manipulator 1349

nouppercase, manipulator 1350

nowa linia 13

NULL adres 68, 368

- › a nullptr 368
- › dawna makrodefinicja 300, 367

null, czyli znak ASCII o kodzie zero 70-71, 300

nullptr 67, 366

- › a NULL 368

nullptr_t 67

numeracja elementów tablicy 290

numeric_limits nazwa typu 51

O

obiekt

- › a klasa - różnica 512-513
- › automatyczny 213
- › automatyczny, a destruktor 780
- › chwilowy 547, 931
- › const 87
- › const, a wskaźniki 365
- › funkcyjny 1119
 - a funkcja orzekająca 1130
 - a przeładowanie operatora() 1045
- › globalny 212, 775
 - jego konstruktor 775
- › inicjalizacja 549
- › klasyfikacja wg. dziedz. lub zawierania 1480
- › konstruowany new 775
- › lambda 1150
- › likwidacja 779
- › lokalny
 - automatyczny 774
 - konstruowanie go 774
 - statyczny 775
- › new, a destruktor 780
- › przesyłany przez wartość 1234
- › składnikiem klasy 819-825
- › składowy bez konstruktora 824
- › static 214
- › statyczny
 - a destruktor 780
 - globalny 222

- lokalny 214

› stały 87

- jego konstruktor 786

› volatile 92

› w środku innego obiektu 819-825

› wytworzony przez new 393

› wywoływalny 1133

obiekt funkcyjny 1123

obiekt-klauzura 1142

obiektove programowanie 1473

obiektove orientowane programowanie 1473-1476

obsługa syt. wyjątkowych 710-736

› rola kompilatora 782

› rola programisty 782

oct, flaga 1339-1341

oct, manipulator 1347, 1356

odczyt pliku

› w binarnym trybie 1419

› w tekstowym trybie 1419

odejmowanie dwóch wskaźników 378

odniesienie się do obiektu 357

„odtąd dotąd” (metoda) 676

off_type, typ w strumieniach we/wy 1429

on flight - definiowanie obiektów 60

open(char*, int, int) 1396

operacja rzutowania 362

operacje wejścia/wyjścia 9, 1320-1393

› błędy 1401-1411

› na plikach 1394-1400

› nieformatowane 1374-1375

operand 128, 957

operandowość tzw. 961, 968

operator 119-155

› ! negacji 126

› "->" 506

› % modulo 120

› % reszta z dzielenia 120

› %= 132

› & (adres) 357

› & | ~ ^ 130

› && oraz || 125

› &= 132

› (), przeładowanie 1043-1048

› * (odniesienie się do obiektu) 357

› *= 132

› += 132

› , (przecinek) 148

› -= 132

› . (kropka) 506

› /= 132

› indeksowania tablicy, przeładowanie 1039-1042

- › ^= 132
- › _Pragma 283
- › alignof 139-140
- › argumentowość tegoż 968
- › arytmetyczny 119-123
- › bitowego iloczynu 130
- › bitowej negacji 130
- › bitowej sumy 130
- › bitowy 127-129
 - a logiczny - porównanie 130-131
- › bool()
 - w strumieniach 1404
- › decltype 876-880
- › dekrementacji 121
- › delete 392
 - a dwukrotnie kasowanie 407
 - odwołanie rezerwacji 399
- › dodawania 957
- › dwuargumentowy 119
- › dwuoperandowy
 - przeładowanie 971-973
- › exclusive or (XOR) 130
- › indeksowania tablicy (w kl. std::string) 624-628
- › inkrementacji 121
- › jako f. składowa, czy globalna 988
- › jako funkcja globalna 970
- › jednoargumentowy (kiedy jest) 121
- › jednoargumentowy +, - 121
- › jego priorytet 148-150
- › jego symbol 957
- › jego łączność 961
- › konwersja w jego obecności 947
- › konwersji 938-944
- › końcówkowy 984
- › logiczny 124-126
 - a bitowy - porównanie 130-131
- › new 392
 - przeładowanie go 1061
- › noexcept 731-733
- › noexcept 137
- › operandowość tegoż 961
- › postdekrementacji 123
- › postinkrementacji 123
- › predefiniowany 967
- › predekrementacji 123
- › preinkrementacji 123
- › priorytet 961
- › przedrostkowy 969
- › przemienność przy przeładowaniu 973
- › przesunięcia bitów w lewo 128
- › przesunięcia bitów w prawo 129
- › przeładowany, definicja 959
 - › przeładowany, lista 959
 - › przypisania 123, 1017, 1029
 - dziedziczenie tegoż 1186
 - klasy pochodnej 1198-1211
 - kopiujący 1015-1028
 - private 1029
 - prywatny 1524
 - przenoszący 1029-1037
 - przenoszący (move) 1029-1037
 - › relacji 124
 - › rzutowania 141-147
 - › różnicy symetrycznej 130
 - › sizeof 135-136
 - › tablicy indeksowania 1433
 - › tworzący typ pochodny 77
 - › typu końcówkowego (postfix) 971
 - › typu postfix 984
 - › wkładania do strumienia 1324, 1329-1337
 - › wyjmowania ze strumienia 1324, 1329-1337
 - › wypisywania - jego przeładowanie 989-994
 - › wywołania funkcji 449
 - › zakresu 86, 913
 - › != 132
 - › łączność operatorów 151
- operatory
 - › których nie przeładujemy 960
- optymalizacja pracy kompilatora 778
- orzekająca funkcja 457, 1121
- ostre nawiasy 142, 282, 1151, 1513, 1515, 1517, 1519, 1526
- ostream 1443-1449
- out, tryb otwarcia pliku 1396
- out_of_range - wyjątek 628, 727
- „outline” funkcja 210
- overloading 470
- override 1279
 - › słowo kluczowe kontekstowe 1281-1293

P

- pamięć komputera 114, 139, 391, 402, 406, 513, 620, 720, 1600
- paradygmat 873
- parametr
 - › aktualny funkcji 184
 - › aktualny szablonu klasy 1515
 - będący referencją 1518
 - › formalny funkcji 184
 - › formalny szablonu 1515
 - › szablonu
 - a constexpr 1545
 - będący adresem fun. globalnej 1550
 - będący adresem obiektu 1550

- będący adresem skł. statycznego 1551
 - będący innym szablonem 1551
 - będący referencją 1550
 - będący stałą całkowitą 1549
 - będący wsk. do skł. klasy 1551
 - co może być 1544-1552
 - domniemany 1553-1555
 - szablonem bibliotecznym 1552
 - trzy rodzaje 1526
- Pasja C++ 734, 755
 peek, funkcja 1385
 pętla nieskończona 29
 pętla programowa 26
 plain enum 103
 Plain Old Data - Pospolite Stare Dane 1310
 plik
 - › dyskowy 1321
 - › dyskowy, operacje we/wy 1394-1400
 - › nagłówekowy 219
 - › programu
 - z szablonem klas 1524
 - › rozmieszczenie w nich funkcji 218-222
 - › rozmieszczenie w nich klas 530-544
 - › tryby otwarcia do oper. we/wy 1396
 płytka kopia 857
 płytkiej kopii metoda 1027
 pochodzenia klasy lista 1175
 POD - Pospolite Stare dane 1310-1312
 podciąg znaków 635
 podprogram 174
 podstawa systemu liczenia 1342
 podsystemy 1485
 podwalina typu wyliczeniowego 99
 pojemnik na sześć jajek 1530
 pokaż kropkę dziesiątą, flaga 1343
 pola bitowe 357, 1103-1106
 - › a deszyfrowanie słów 1107-1113
 - › szerokość zadana jako constexpr 836
 pole justowania 1341
 polimorficzna klasa 1265
 polimorficzny typ 1307
 polimorfizm 1264-1266, 1473, 1475
 - › anulowany 1217
 - › przy operatorze delete 1079
 pomijanie kopiowania 849
 pomocnik stringowy 728
 pop_back (string::) 643
 porównywanie
 - › stringów (alfabetyczne) 658
 - › wskaźników 380
 pos_type, typ 1429
 postfix operator 971
 postmortem dump 366
 pozycyjny system liczenia 1598
- pośrednia kl. podstawowa 1187
 pragma 283
 prawda - stan logiczny 48
 prawda-fałsz 20-21
 prawostronnie łączny operator 961
 precision, funkcja strumienia 1365, 1373
 predefiniowany
 - › manipulator 1346
 - › strumień 1323
 preprocesor 270-288
 - › dyrektywa 270
 priorytet
 - › operatora 148-150, 961
 - › operatorów we/wy 1328
 private 509-511
 procedura 1472
 proceduralne programowanie 1472
 proces - opisany przez klasę 1045
 program
 - › OO, projektowanie 1471-1512
 - › składający się z kilku plików 218-222
 - › wywołanie go z argumentami 441-443, 1460
 - › zawsze się 378
 programowanie
 - › liniowe (linearne) 1472
 - › obiektowe 1473
 - › obiektowo orientowane 1473-1476
 - › proceduralne 1472
 - › uogólnione 1513-1594
 - › z ukrywaniem danych 1472
 projektowanie programu OO 1471-1512
 promocja argumentu 495
 protected 509-511, 1179-1180
 protony 744
 prwartość 873-875
 prywatna klasa 826
 przechwytywanie wyjątków 627
 przecinek (operator) 148
 przecinki (dwa obok siebie) 201
 przefadowanie
 - › a enum 988
 - › a przestrzeń nazw 480
 - › a const 581
 - › a przyjaźń 702
 - › a volatile 581
 - › a wirtualność 1279-1280
 - › a zakres ważności nazwy funkcji 478-479
 - › czwórki operatorów 1015-1060
 - › delete 1061-1087
 - › funkcji 470-503
 - › i zastąpienie równocześnie 530
 - › konstruktora 551
 - › nazw funkcji, a technika OO 476

- › nazw klas jest niemożliwe 1526
- › nazwy funkcji 470-503
 - a argumenty domniemane 475
- › new (operatora) 1061-1087
- › operatora 957-1014
 - () - wywołania funkcji 1043-1048
 - dwuoperandowego 971-973
 - indeksowania tablicy 1039-1042
 - indeksowania tablicy - przykład 1433
 - jednooperandowego 968-970
 - lista możliwych 959
 - new 1061
 - odniesienia się wskaźnikiem 1049-1057
 - ogólna definicja 959
 - postdekrementacji 984-985
 - postinkrementacji 984-985
 - przypisania (kopiującego) 1015-1028
 - przypisania (przenoszącego) 1029-1037
 - wypisywania 989-994
- › operatora wypisywania 989-994
- › zaoszczędzone dzięki konwersjom 931
- przenoszenia mechanizm 778
 - › w klasie pochodnej 1208
- przenoszenie a klasa string 680
- przenoszący
 - › konstruktor 866-872
 - › operator przypisania 1029-1037
- przepis na placek ze... 1521
- przepis-szablon 1514
- przeźrzeń nazw 222
 - › a przeładowanie 480
 - › a zagnieżdżanie klas 739
 - › anonimowa 223
 - › dyrektywa using 83
 - › std::: 239, 611, 1394
 - › stopniowe zaludnianie 82, 722
 - › zakres 80
- przesunięcie bitów
 - › w lewo 128
 - › w prawo 129
- przez wartość przesłanie obiektu 1234
- przydomek
 - › const 87
 - › constexpr 88-91
 - › explicit 773
 - › zob. też: modyfikator, specyfikator
 - › mutable 591-592
 - › noexcept 731-733
 - › register 92
 - › static 216
 - › volatile 92
- przyjaźń
 - › jej deklaracja 697

- › w świecie szablonów 1567-1573
- przylegające C-stringi 72
- przypisanie 59, 1017
 - › „kaskadowe” 1024
 - › definicja 88
 - › operator 123
 - › przy dziedziczeniu 1196-1197
 - › „bit po bicie” 1197
 - › „składnik po składniku” 1197
- pseudokod 22, 41
- ptrdiff_t 379
- public 509-511
- push_back - f. skł. std:::vector 168
- pusta klamra { } 111-112
- put(char) - f. w kl. strumienia 1387
- putback - f. w kl. strumienia 1386

Q

QtCreator 12

R

rachunek Lambda (Alonzo Church) 1133

radiansy 250

raw (surowy) string 74

rdstate, f. sprawdzająca stan strumienia 1405-1406

read(char*, int) 1382

recykling, a konstruktory 859-861

referencja 77, 357

- › a przeładowanie 484
- › argumentem funkcji 185-187
- › definicja jej za pom. auto 253-262
- › do klasy pochodnej 1228-1240
- › jako par. aktualny sz. klas 1518
- › lwartości 190-197
- › martwa 225, 1155-1157
- › rwartości 190-197, 859-861

regex 74

register 92

regular expressions 74

reguła SFINAE 1539-1542

reinterpret_cast 142, 147

- › a przeładowanie 960
- › a wskaźniki 361-363
- › przy ustaw. wskaźnika 390

rekurencja 228-236

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

relacja delegowania 1480

relacji operator 124

replace - f. w kl. std::string 646-648
 reserve - f. w kl. std::string 621
 resetiosflags, manipulator 1357
 resize - f. w kl. std::string 622
 reszta z dzielenia 120
 return 175, 177-180
 > mechanizm 383
 reusability 1234, 1474, 1483
 rezerwacja obszarów pamięci 391-413
 rezultat zwracany przez funkcję 177-180
 rfind - f. w kl. std::string 640
 right, flaga 1339-1340
 right, manipulator 1351
 robot przemysłowy 1602
 rozbudowalność C++ 1475
 rozmiar tablicy (wymagania) 289
 rozmiar unii 1088
 rozmiar, a długość C-stringu 302
 rozmieszczenie klas w plikach 531
 rozmieszczenie szablonów klas w plikach 1524
 rozpad beta 753
 rozpad promieniotwórczy 745
 rozszerzalność C++ 1267, 1475
 rozszerzona dokładność 48
 runtime_error (klasa wyjątku) 1424
 rvalue 188-189
 > zob. też: zob. rwartość
 RVO technika 848
 rwartość 188-189
 > jej referencja 190-197
 rzucenie wyjątku przez operator new 410
 rzutowanie 141-147
 > const_cast 145, 428-431
 > dynamic_cast 146, 1307-1309
 > nowymi operatorami 142
 > reinterpret_cast 147
 > static_cast 143
 > std::move 862-863

S

środowisko programowania 540
 scientific
 > flaga 1339-1340, 1343
 > manipulator 1350
 > notation (notacja) 1327
 seek_dir, typ 1429-1430
 seek, funkcja 1429
 seekp, funkcja 1429
 segment violation 366
 sekwencja działań obiektu 1486
 semantyka przenoszenia tzw. 873
 set_new_handler 412-413
 setbase, manipulator 1356

setf, funkcja 1357, 1364-1369
 setfill, manipulator 1354
 setiosflags, manipulator 1357
 setprecision, manipulator 1354
 setstate, funkcja 1405-1406
 setw, manipulator 1351
 SFINAE reguła 1539-1542
 shared_ptr - sprytny wskaźnik 1057
 short 46
 short int, typ 46
 show positive 1342
 showbase, flaga 1339-1340, 1342
 showbase, manipulator 1349
 showpoint, flaga 1339-1340, 1343
 showpoint, manipulator 1349
 showpos, flaga 1339-1340, 1342
 showpos, manipulator 1349
 shrink_to_fit 622
 signed 46
 silnia 249
 sinus - rozwinięcie w szereg Taylora 250
 size() 618
 size_t 1064
 size_type, typ w klasie std::string 618, 637
 sizeof operator 135-136
 > a przeladowanie 960
 skip white space 1340
 skipws, flaga 1339-1340
 skipws, manipulator 1340, 1349
 „składnik po składniku” 1197
 sklejacz ## 275
 składa się z...
 > czyli ma obiekt składowy 1226
 składana instrukcja 23
 składnik
 > będący klasą 737-743
 > będący obiektem innej klasy 507
 > będący typem 737-743
 > const w klasie 1029
 > dana 506
 > funkcja 507
 > klasy 506
 > klasy, dostęp 509-511
 > odziedziczony 1175
 > odziedziczony, dostęp 1177-1184
 > referencja w klasie 1029
 > statyczny 557-565
 • definicja tegoż 558
 • deklaracja, a definicja 558
 • inicjalizacja w klasie 571
 • kiedy się przydaje 566
 • trzy sposoby odniesienia się 559
 • w szablonie klas 1532
 • wskaźnik doń 925

- › wariantywny 1093
- › zasłonięty 1176
- składnik po składniku 857, 1197
 - › metoda kopiowania 1016
- słowa kluczowe C++ 16
- słowo 1600
 - › jednostka informacji 127
 - › pamięci 114
- spacje 24, 1326, 1328
- spaghetti kod à la 1472, 1485
- specjalizacja
 - › częściowa fun. skł szabl. klas
 - a ref. lwartości, rwartości 1584
 - › częściowa szablonu klas 1579
 - zastosowanie 1581
 - › funkcji składowej szablonu 1585-1586
 - › kompletna (zupelna) 1577
 - › szablonu funkcji
 - zrobiona przez użytkownika 1587-1588
 - › szablonu klas
 - zrobiona przez użytkownika 1574-1584
- specjalizowana klasa szablonowa 1574-1584
- specjalne funkcje składowe 1038
- specjalny znak 69
- specyfikacja wyjątków 733
- specyfikator
 - › zob. też: przydomek, modyfikator
 - › a deklaracja przyjaźni 707
 - › const 87
 - › constexpr 88-91
 - › dostępu 1175
 - › explicit 773
 - a konstruktor konwertujący 930
 - › mutable 591-592
 - › noexcept 731-733
 - › przestrzeni nazw 84
 - › register 92
 - › signed, unsigned 46
 - › static 216
 - › volatile 92
- spis relacji klas 1484, 1498
- sposób dziedziczenia 1180
- sprytny wskaźnik 1051
 - › shared_ptr 1057
 - › unique_ptr 1057
- środowisko programowania 12
- środowisko programowania 531
- stan niski i stan wysoki 1599
- stan wewnętrzny obj. funkcyjnego 1126
- standardowa biblioteka 237
- standardowa biblioteka strumieni we/wy 1320
- static
 - › a funkcja wirtualna 1272
 - › funkcja 566-576
 - › funkcja składowa 965
 - › modyfikator 216
 - › obiekt 214, 217
 - › składnik w klasie 557-565
 - › specyfikator 216
 - › string 441
- static_assert 137-138
- static_cast 142
 - › a przeładowanie 960
 - › jako funkcja szablonowa 1521
- statyczne-
 - › f. składowa szablonu klasy 1533
 - › funkcja składowa 566-576
 - › obiekt globalny 222
 - › obiekt lokalny 214
 - › składnik
 - inicjalizacja w klasie 571
 - zastosowanie 566
 - › są f. składowe new i delete 1064
- stała
 - › dosłowna 62-75
 - C-string 71
 - całkowita 63
 - definiowana przez użytkownika 995-1010
 - klasy 828
 - typu long long 64
 - zmiennoprzecinkowa 66
 - znakowa 68
 - znakowa typu wchar_t 70
 - › tekstowa 71
 - › za pomocą #define 272
- stały
 - › obiekt 87
 - › wskaźnik 418, 885
- std::bad_alloc 795, 1064
- std::bad_alloc, typ wyjątku 410
- std::count_if - algorytm biblioteczny 1127
- std::exception 795, 1424
- std::flush - manipulator 768
- std::for_each - algorytm biblioteczny 1149
- std::initializer_list 804-818
- std::memcopy - f. biblioteczna
 - › f. biblioteczna 1310
- std::move
 - › funkcja, a przenoszenie 862-863
 - › w klasie pochodnej 1210
- std::nullptr_t 67
- std::string 156-160, 609-695
 - › out_of_range, typ wyjątku 627
 - › alfabetyczne porównywanie 658
 - › at 624-628
 - › begin - f. iteratora 675

- › bryk, czyli zestawienie funkcji 681-688
- › c_str() 650-652
- › capacity 619
- › clear 624
- › compare 654
- › copy 662
- › data() 649
- › definiowanie obj. tej klasy 612-616
- › długość tegoż 618-623
- › dopisywanie do końca 617
- › empty 619
- › end - f. iteratora 675
- › erase 643
- › find 637-639
- › find_first_not_of 641
- › find_first_of 641
- › find_last_not_of 641
- › find_last_of 641
- › fragment tegoż 635
- › inicjalizacja 614
- › insert 644-645
- › iteratory 670-679
- › konstruktory 614
- › length 618
- › liczby wczytanie z niego 632-634
- › liczby wpisanie do niego 630-631
- › liter małych na wielkie zamiana 659-661
- › litery wielkie i małe - a porównywanie 653
- › max_size() 619
- › npos, wartość (string::npos) 637
- › operator indeksowania tablicy 625
- › operatory =,+,+= 617
- › operatory porównywania 658
- › pierwsza wzmianka 156-160
- › pojemność 618-623
- › porównywanie tychże (alfabetyczne) 653-658
- › replace 646-648
- › reserve 621
- › resize 622
- › rfind 640
- › rozmiar 618-623
- › shrink_to_fit 621
- › size 618
- › size_type, typ w klasie string 637
- › substr 636
- › swap 662
- › zamiana na C-string 650-652
- std::transform - algorytm biblioteczny 1167
- stdexcept - plik nagłówkowy wyjątków 628, 635
- stdio
 - › biblioteka 1320
 - › flaga ios:: 1340
- sterow. formatem oper. we/wy 1338
- sterowanie formatem operacji we/wy 1338-1344, 1364-1369
- stod 632
- stof 632
- stoi 632
- stol 632
- stold 632
- stoll 632
- stopnie na radiany 250
- stos 183
- stoul 632
- stoull 632
- stowarzyszony typ 1535
- strażnik nagłówka 282, 534
 - › a #pragma once 283
- strcpy 302, 305, 438
- streamsize, typ 1370
- string
 - › dopisywanie do końca 159
 - › klasa biblioteczna 156-160, 609-695
 - › porównywanie ich 158
 - › a przenoszenie 680
 - › surowy 73
 - › to_string 630
- string::iterator 671
- string::npos 637
- stringstream
 - › opis 1464-1468
 - › przykład użycia 1465
- Stroustrup Bjarne 1516-1517, 1520, 1579
- struktura 582, 1182, 1213
- strumienie, a wyjątki 1424-1427
- strumień 1321-1322
 - › a operator bool() 1404
 - › buforowany 1324
 - › jego ujście 1324
 - › niebuforowany 1324
 - › otwieranie 1396
 - › predefiniowany 1323
 - › predefiniowany, domniemania 1325-1327
 - › tie() - funkcja do powiązania 1438-1439
 - › tryb pracy 1396
- strzał na oślepie 366-367
- substr - f. w kl. std::string 636
- substring 635
- suma logiczna 125
- surowe stałe tekstowe 74
- surowy string (stała tekstowa) 73
 - › przykładowy program 728
- swap - f. w kl. std::string 662
- switch 30-32
 - › a constexpr 34

- Sydney Opera 350
 - sygnatura funkcji 1268
 - symboliczna nazwa typu 1517
 - symetria operacji we/wy 1333
 - synonim nazwy typu 93
 - › klasy szablonowej 1534
 - system (część zagadnienia) 1479
 - system liczenia 1597-1606
 - › dwójkowy 1598
 - › dziesiętkowy 1597
 - › pozycyjny 1598
 - › szesnastkowy 1603-1604
 - › wagowy 1598
 - sytuacja wyjątkowa obsługa 710-736
 - szablon funkcji
 - › a funkcje virtual 1560
 - › lista parametrów 1519
 - › o nazwie static_cast 1521
 - › po co? 1519-1522
 - › przepis kucharski 1521
 - › specjalizacja użytkownika 1587-1588
 - › zagnieżdżanie 1556
 - szablon klas
 - › a specjalizacja użytkownika 1574-1584
 - › bardziej specjalizowany 1581
 - › definicja 1514-1515
 - › deklaracje przyjaźni 1567-1573
 - › destruktor 1532
 - › funkcja składowa 1531
 - › kiedy generuje klasę szablonową 1543
 - › konstruktor 1531
 - › linkowanie 1524
 - › parametr aktualny 1515
 - › parametr formalny 1515
 - › składniki statyczne 1532
 - › specjalizacja częściowa 1579
 - › specjalizacja funkcji składowej 1585-1586
 - › statyczna funkcja składowa 1533
 - › unikalna nazwa 1525
 - › using - wykorzystanie 1535
 - › zagnieżdżanie 1556
 - szablonowa klasa 1513
 - szablony 1513-1594
 - szeroki znak 47
 - szesnastkowy system liczenia 1603-1604
 - sześcioopak 1530
 - szybkie mnożenie 133
- T**
- tablica 76, 289-311
 - › bardzo duża, przykład 1432-1437
 - › dwa sposoby wysyłania jej do funkcji 385
 - › dynamiczna alokacja 391, 398
 - › element jest lwartością 190
 - › inicjalizacja 292
 - › jej adres 294
 - › jej elementy 290-291
 - › jej nazwa 294, 885
 - › new, kasowanie jej 399, 886
 - › numeracja elementów 290
 - › obiektów
 - definiowana new 884-885
 - inicjalizacja 886-892
 - jakiejś klasy 883-898
 - stałych, jej inicjalizacja 293
 - › określanie rozmiaru w definicji 289
 - › przekazywanie do funkcji 293-296
 - › rezerwowana dynamicznie 391-413
 - › rozmiar 297
 - › tablic 317
 - › tablic, a new 400
 - › wielowymiarowa 312-321, 482, 1045
 - a przetwarzanie 482
 - a new 399
 - jej typ 316-317
 - wysyłanie do funkcji 318-319
 - › wskaźników 432-433
 - do danych składowych 920
 - do funkcji 459-463
 - do funkcji składowych 921-924
 - › wymaganie konstr. domniemanego 893
 - › wysłanie do funkcji jednego jej elementu 297
 - › z czego można tworzyć 290
 - › zerowanie klamrami {} 399
 - › znakowa 299-306
 - tabliczka czekolady (detektor) 744
 - tabulator (znak specjalny) 69
 - Taylor'a szereg 250
 - technika RVA 848
 - technika obiektowo orientowana 1471
 - › a przetwarzanie nazw funkcji 476
 - technika proceduralna 1487
 - teksańska masakra 195
 - tekst źródłowy programu 11
 - tekstowa stała 71
 - tekstowy tryb
 - › odczytu pliku 1419
 - › zapisu pliku 1418
 - telegraf maszynowy 99
 - tellg() 1429
 - tellp() 1429
 - template słowo kluczowe 1515
 - terminate funkcja biblioteczna 719
 - this 523-524
 - › dla destruktorów 781

- › przykładowe zastosowanie 525
- › typ tego wskaźnika 524, 581
- throw 411, 712
- tie() - powiązanie strumieni 1438-1439
- to_string - funkcja pomocnicza (std::) 159, 630
- tolower 661
- transform - algorytm biblioteczny 1167
- true 48, 63
- trunc, tryb otwarcia pliku 1396
- try 411-412
 - › a lista inicjalizacyjna konstruktora 796
 - › blok 711
 - › dla listy inicj. konstruktora 795
- tryb otwarcia pliku 1396
- tryb pracy pliku
 - › binarny odczyt 1422
 - › binarny zapis 1421
 - › tekstowy odczyt 1419
 - › tekstowy zapis 1418
- trywialna klasa 1311
- trywialna konwersja 494
- trójwymiarowe wektory 338-340
- typ 44-118
 - › C-stringu 72
 - › arytmetyczny 49
 - › definiowany przez użytkownika 504-505
 - › dosłowny 585
 - › fundamentalny 46-54, 76
 - › a klasa 505
 - › literalny 574, 837
 - › niekompletny 742
 - › pochodny 506
 - › polimorficzny 1307
 - › size_t 1064
 - › składowy 737-743
 - › stowarzyszony (z szablonem) 1535
 - › systematyka 45
 - › void 77
 - › void* 77
 - › wbudowany 46
 - › wskaźnika do funkcji 454
 - › wyliczeniowy enum 96-105
 - jako indeks tablicy 543
 - › zależny
 - w szab. klasy 1536
 - › zdefiniowany przez użytkownika 46
 - › złożony 46, 76
- typedef 93-95, 949
 - › a przeładowanie 481
 - › deklaracja 93-95
 - › lokalne 758
 - › w operatorze wej/wyj 1333

- typeid operator
 - › a przeładowanie 960
- typename
 - › a słowo 'class' 1520
 - › dodane do typu zależnego 1535
 - › słowo kluczowe 1515

U

- u16string 611
- u32string 611
- udostępnianie wybiórcze 1182
- 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 1321, 1324
- ukrywanie danych, programowanie 1472
- ukrywanie informacji 509-511
- układ sprzęgający 1104, 1602
- umiejszczający operator new 402, 1076, 1101
- unset, funkcja 1387
- unia 1088-1089
 - › a deszyfrowanie słów 1107-1113
 - › anonimowa 1090-1091
 - › inicjalizacja 1090
 - › jej wyróżnik 1094
 - › z metryczką 1094
 - › rozmiar 1088
- UNICODE 47
- uniopodobna klasa 1092-1093
- unique_ptr 795
 - › sprytny wskaźnik 1057
- unitbuf, flaga 1339-1340, 1345
- unitbuf, manipulator 1349
- unsetf, funkcja 1357, 1364-1369
- unsigned 46
- uogólnione
 - › klasa 1190, 1514
 - › programowanie 1513-1594
- uppercase, flaga 1339-1340, 1342
- uppercase, manipulator 1350
- using 949
 - › deklaracja 84, 93-95
 - › deklaracja dostępu 1182
 - › do typu wskaźnika 464

- › dyrektywa 83
- › składnikiem kl. szablonowej 1535
- › a wskaźnik do skł. klasy 909

ustawianie wskaźnika 357-359
 UTF-16 47
 UTF-32 47

V

vector

- › a initializer list 813
- › jego budowa 163
- › pierwsza wzmianka 161-168

vertical tabulator (znak specjalny) 69

virtual

- › a konstruktor 762
- › funkcja 1257-1319
- › funkcja, a klasa - różne znaczenie 1262
- › klasa podstawowa 1241-1249

void 77

void* wskaźnik 364-365

volatile 92

- › a const_cast 432
- › a konstruktor 762
- › a przeładowanie 581
- › funkcja składowa 577-581

VXI (standard urządzeń) 1108

W

w klasie inicjalizacja 514-516

wagowy system liczenia 1598

wahadło (na ekranie symbolicznie) 462

wariantywny składnik 1093

warunek zatrzymujący rekurencję 228, 233

warunkowa kompilacja 276-279

warunkowe wyrażenie 134

wbudowany typ 46

wchar_t 47, 611

wchar_t - stałe znakowe tego typu 70

wczytywanie formatowane 1385

wczytywanie nieformatowane 1385

wczytywanie z klawiatury długiego stringu 663-669

wdowa po prenumeratrze (metafora) 1238

wejścia/wyjścia operacje 1320-1393

wektor

- › 2D 323
 - nieprostokątny 336-337
- › 3D 338-340
 - nieprostokątnościenny 348-351
- › agregatów 895
- › dwuwymiarowy, definiowanie 323
- › dwuwymiarowy, inicjalizacja 327

- › inicjalizacja go 897
- › jako agregat 895
- › nieagregatów 897
- › obiektów 893-897
- › trójwymiarowy 338-340

what - funkcja składowa wyjątku 730, 795, 1427

while 26

wiatraczek - na ekranie (symbolicznie) 462

wide character 47

widmo promieniowania 964

widoczne własności 1487

width, funkcja 1351, 1365, 1371

wielka szóstka funkcji składowych 1038

wielka tablica 960

wielodziedziczenie 1212

wielokropek 499

wielokrotne dziedziczenie 1212

wielorakie dziedziczenie 1212

wielowariantowy wybór 25, 31, 33

wielowymiarowa

- › tablica 312-321, 1045
- › tablica new 399
- › tablica, a przeładowanie 482
- › wektor 322-353

wielowątkowy program 91

wieloznaczność

- › a dziedziczenie wirtualne 1244
- › przy dziedziczeniu 1216
- › przy konwersji 945, 952, 954

wierzchni const 420, 485

wirtualne-

- › czysto (funkcja) 1297
- › destruktor 1277-1278
- › dziedziczenie 1241-1249
- › funkcja 1257-1319, 1500
- › klasa podstawowa 1241-1249

wirtualność

- › a operatory we/wy 1335
- › a przeładowanie 1279-1280
- › a wczesne wiązanie 1274

wiązanie

- › późne 1272-1273
- › wczesne 1272-1273
- › wczesne, a wirtualność 1274

wolny format zapisu 9

write(const char*, int) 1388

ws, manipulator 1349

wskaźnik 76, 354-369, 418-445

- › a auto (ostrzeżenie) 360
- › a nazwa tablicy 376
- › a reinterpret_cast 361-363
- › argumentem funkcji 381-389
- › arytmetyka 378

- › definiowanie 356
- › do const 419-420
- › do funkcji 446-469
 - argumentem innej funkcji 455-458
 - definiowanie i odczytywanie deklaracji 449-454
 - orzekającej 1119
 - składowej
 - Zob. niżej: 'wsk. do fun. składowej klasy'*
 - tablica takich wsk. 459-463
 - typ tegoż 454
- › do klasy pochodnej 1228-1240
- › do obiektów klasy 884
- › do przebiegania (iterator) 1535
- › do składnika-danej 900-910
- › do składników klasy 899-927
 - tablica/wektor tychże 920
- › do składników statycznych 925
- › do stałej 419-420
- › dodawanie i odejmowanie liczby całkowitej 378
- › domeny zastosowania 370-417
- › dostęp do komórek pamięci 390
- › globalny 366
- › konwersja 498
- › odejmowanie dwóch od siebie 378
- › pisania i czytania 1428-1431
- › poruszanie nim 370
- › porównywanie 380
- › sposoby ustawiania 426-427
- › sprytny 1051
- › stały 418, 885
 - inicjalizacja go 419
- › tablica tychże 432-433
- › this 523-524
 - jego typ 524, 581
- › typu void* 364-365
- › ustawianie 357-359
- › ustawianie za pom. reinterpret_cast 390
- › w zastosowaniu do tablic 370-380
- › zwykły, we wnętrzu obiektu 900

wskaźnik czytania get 1428

wskaźnik do funkcji składowej klasy 911-919

- › a auto 913
- › a decltype 913
- › tablica/vector tychże 921-924

wskaźnik pisania put 1429

wstring 611

wszystkożerny catch 729

wybiórcze udostępnianie 1182, 66

wybór wielowariantowy 25, 31, 33

wychwytywania lista 1140

wyciek pamięci 1098

wygaśnięcie dynastii 1177

wyjątek

- › a strumienie 1424-1427
- › nowy sposób powiadamiania 410
- › przechwytywanie go 627
- › rzucany z konstruktorowej listy inic. 793
- › specyfikacja 733
- › z destruktora? 782
- › z funkcji string::at 627
- › z wyrażenia lambda 1166-1169

wyjatków specyfikacja

- › w wyrażeniu lambda 1143

wykładnicza notacja 66, 1327, 1343

wykładnik i cecha 1344

wyliczeniowy typ enum 96-105

- › jako indeks tablicy 543

wyrażenie

- › inicjalizujące 360
- › lambda 1118-1173
 - cztery formy zapisu 1137-1142
- › logiczne 20
- › regularne (regex) 74
- › warunkowe 134
 - a w nim definicja obiektu 61-62
 - w funkcji constexpr 246

wyrównanie adresu 115

wyróżnik unii 1094

wywołania funkcji - operator 449

wywoływalny obiekt 1133, 1154

X

xwartość 873-875

Z

zacieranie funkcji 1279

zagnieżdżanie

- › a szablony 1556-1566
- › klasy w szablonie 1556-1566
- › komentarzy 14
- › zakresów 1175-1176

zagnieżdżona

- › deklaracja przyjaźni 1556
- › klasa 737-743
- › klasa, a przyjaciele 743

zagnieżdżony

- › szabł. funkcji składowych 1557
- › szablon funkcji 1556
- › szablon klas 1556

zakres

- › blok funkcji 79
- › klasy 1175
 - pochodnej 1175
 - podstawowej 1175

- › kwalifikator tegoż 1176
- › leksykalny 701, 743, 1333
 - szablonu kl. 1538
- › obszar pliku 80
- › przestrzeń nazw 80
- › ważności
 - Zob. niżej: 'zakres ważności'
- › zagnieżdżanie ich 1176
- zakres instrukcji 79
- zakres ważności
 - › definicja 78
 - › etykiety 212
 - › funkcja 79
 - › klasa 80, 508
 - › lokalny 78
 - › namespace 80
 - › nazw w funkcji 212
 - › nazw w klasie 507
 - › nazwy funkcji, a przeładowanie 478-479
 - › nazwy obiektu 212-217, 776
 - › obiektów globalnych 775
 - › obiektów lokalnych 774
 - › obiektów new 393, 775
 - › pliku 80
 - › zagnieżdżanie 1175
- zakresowe enum 103
- zakresowe for 169-171, 629
 - › a funkcje begin, end 1537
 - › a wektor wielowymiarowy 326
- zakresu operator 913
 - › a zasłanianie 86
- zależny typ w szablonie klasy 1536
- zapas pamięci 395, 884, 1061
 - › wyczerpanie go 409
- zapis binarny 1599
- zapis binarny liczby 127
 - › jak dokonać zmiany 235
- zapowiadająca deklaracja 700
- zaprzyjażniona funkcja 697
- zaprzyjażniona klasa 705-706
- zasada trzech 859, 1023
- zasłanianie nazw 85-86, 526-529
- zasłanianie składnika 1176
- zasłonięcie i przeładowanie 530
- zasłonięta nazwa globalna - dostęp 528
- zatrzymanie rekurencji 228
- zawieranie
 - › obiektu 1480, 1483, 1497
 - › obiektów, a dziedziczenie klas - różnica 1226-1227
- zawieszający się program 378
- zawężająca konwersja 60
- zbieracz śmieci (garbage collector) 1062
- zdarzenie (w fizyce jądrowej) 1112
- zerowanie nowej tablicy 399
- zerowy adres 67
- zespolone liczby 928, 957-958
- zgrupowanie danych 886
- zjawisko Dopplera 914
- zmiana formatu operacji we/wy 1364-1369
- zmienna 15
 - › automatyczna 213
 - › statyczna globalna 222
 - › statyczna lokalna 214
- znak
 - › specjalny 69
 - › szeroki 47
 - › wypełniający 1354
- znakowe stałe dosłowne 68
- źródłowy kod programu 11
- źródło strumienia 1321
- złożony typ 46, 76

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ 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 WYDAWNICZA

 **Helion SA**

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 języka C++ 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 jest 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 książki o tym języku, która w prostym, wręcz przyjacielskim stylu wprowadza czytelnika w fascynujący świat programowania zorientowanego obiektowo. Zobacz, jak potężny jest dzisiaj C++11.

Helion

księgarnia Internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/nowosci>

ISBN 978-83-283-4214-9



9 788328 342149

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ



KOD KORZYŚCI