

Tom 3

28	Przeładowanie operatorów <i>new</i> i <i>delete</i> na użytek klasy	1061
28.1	Po co przeładowujemy 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 przeładowanymi <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 starym 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	Operatorzy <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 przeładowanie operatorów	1085
28.6	Cwiczenia	1086
29	Unie i pola bitowe	1088
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	Cwiczenia	1114



30 Wyrażenia lambda i wysłanie kodu do innych funkcji1118

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	Ćwiczenia	1171

31 Dziedziczenie klas1174

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 przeladowane, 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	Strumień płynący 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	Strumień 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 obiektowo.....1471

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 uogólnione.....1513

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 destruktora 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 szablony 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	Cwiczenia	1589

38 Posłowie 1595

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

A Dodatek: Systemy liczenia 1597

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

B Skorowidz 1607