

"Opus magnum C++11. Programowanie w języku C++" - zestaw obejmuje 3 podręczniki.

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 instytutach 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.

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

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 orientowane obiektowo
 - Szablony
-

Spis treści

Proszę tego nie czytać! (1)

- 0.1. Zaprzyjaźnijmy 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 bool w roli warunku (21)
 - 2.1.3. Stare dobre sposoby z dawnego C++ (21)
- 2.2. Instrukcja warunkowa if (22)
- 2.3. Pętla while (26)
- 2.4. Pętla do...while. (27)
- 2.5. Pętla for (28)
- 2.6. Instrukcja switch (31)
- 2.7. Co wybrać: switch czy if...else? (33)
- 2.8. Instrukcja break (36)
- 2.9. Instrukcja goto (37)
- 2.10. Instrukcja continue (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 bool (63)
 - 3.7.2. Stałe będące liczbami całkowitymi (63)
 - 3.7.3. Stałe reprezentujące liczby zmiennoprzecinkowe (66)
 - 3.7.4. Stała dosłowna nullptr - 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 void (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) const (87)
- 3.13. Specyfikator (przydomek) constexpr (88)
- 3.14. Obiekty register (92)
- 3.15. Specyfikator volatile (92)
- 3.16. using oraz typedef - tworzenie dodatkowej nazwy typu (93)
- 3.17. Typy wyliczeniowe enum (96)
 - 3.17.1. Dawne zwykłe enum a nowe zakresowe enum class (103)
 - 3.17.2. Kilka uwag dla wtajemniczonych (105)
- 3.18. auto, czyli automatyczne rozpoznawanie typu definiowanego obiektu (106)
- 3.19. decltype - operator do określania typu zadanego wyrażenia (109)
- 3.20. Inicjalizacja z pustą klamrą { }, czyli wartością domniemaną (111)
- 3.21. Przydomek alignas - adresy równe i równiejsze (113)
- 3.22. Ćwiczenia (115)

4. Operatory (119)

- 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. Wrażenie warunkowe (134)
- 4.8. Operator sizeof (135)
- 4.9. Operator noexcept (137)
- 4.10. Deklaracja static_assert (137)
- 4.11. Operator alignof 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 static_cast (143)
- 4.12.4. Operator const_cast (145)
- 4.12.5. Operator dynamic_cast (146)
- 4.12.6. Operator reinterpret_cast (147)
- 4.13. Operator: przecinek (148)
- 4.14. Priorytety operatorów (148)
- 4.15. Łączność operatorów (151)
- 4.16. Ćwiczenia (152)

5. Typ string i typ vector - pierwsza wzmianka (156)

- 5.1. Typ std::string do pracy z tekstami (156)
- 5.2. Typ vector - długi rząd obiektów (161)
- 5.3. Zakresowe for (169)
- 5.4. Ćwiczenia (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 auto, a inicjalizowany rezultatem funkcji (179)
 - 6.4.2. O zwracaniu (lub niezwracaniu) rezultatu przez funkcję main (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 inline (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 constexpr (240)
 - 6.21.1. Wymogi, które musi spełniać funkcja constexpr (w standardzie C++11) (242)
 - 6.21.2. Przykład pokazujący aspekty funkcji constexpr (243)
 - 6.21.3. Argumenty funkcji constexpr, będące referencjami (252)
- 6.22. Definiowanie referencji przy użyciu słowa auto (253)
 - 6.22.1. Gdy inicjalizatorem jest wywołanie funkcji zwracającej referencję (260)
- 6.23. Ćwiczenia (263)

7. Preprocesor (270)

- 7.1. Dyrektywa pusta # (270)
- 7.2. Dyrektywa #define (270)
- 7.3. Dyrektywa #undef (272)
- 7.4. Makrodefinicje (273)
- 7.5. Sklejacz nazw argumentów, czyli operator ## (275)
- 7.6. Parametr aktualny makrodefinicji - w postaci tekstu (276)
- 7.7. Dyrektywy kompilacji warunkowej (276)
- 7.8. Dyrektywa #error (280)
- 7.9. Dyrektywa #line (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. Tablice (289)

- 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 enum (297)
- 8.6. Tablice znakowe (299)
- 8.7. Ćwiczenia (307)

9. Tablice wielowymiarowe (312)

- 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 vector (322)
- 10.2. Jak za pomocą klasy vector 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 `resize` (329)
- 10.8. Zmiany rozmiaru wektora 2D funkcjami `push_back`, `pop_back` (330)
- 10.9. Zmniejszanie rozmiaru wektora dwuwymiarowego funkcją `pop_back` (333)
- 10.10. Funkcje mogące modyfikować treść wektora 2D (333)
- 10.11. Wysyłanie 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 `resize` (345)
 - 10.16.2. Zmiana rozmiarów wektora 3D funkcjami `push_back` (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 `auto` (360)
- 11.5. Wyrażenie `*wskaźnik` jest wartością (361)
- 11.6. Operator rzutowania `reinterpret_cast` a wskaźniki (361)
- 11.7. Wskaźniki typu `void*` (364)
- 11.8. Strzał na oślep - wskaźnik zawsze na coś wskazuje (366)
 - 11.8.1. Wskaźnik wolno porównać z adresem zero - `nullptr` (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 `const` (387)
- 12.3. Zastosowanie wskaźników przy dostępie do konkretnych komórek pamięci (390)
- 12.4. Rezerwacja obszarów pamięci (391)
 - 12.4.1. Operatory `new` i `delete` albo Oratorium Stworzenie Świata (392)
 - 12.4.2. Operator `new` a słowo kluczowe `auto` (396)
 - 12.4.3. Inicjalizacja obiektu tworzonego operatorem `new` (396)
 - 12.4.4. Operatorem `new` możemy także tworzyć obiekty stałe (397)
 - 12.4.5. Dynamiczna alokacja tablicy (398)
 - 12.4.6. Tablice wielowymiarowe tworzone operatorem `new` (399)

- 12.4.7. Umiejscawiający operator new (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 std::bad_alloc (410)
- 12.4.11. Funkcja set_new_handler (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 auto (421)
 - 13.3.1. Symbol zastępczy auto a opuszczanie gwiazdki przy definiowaniu wskaźnika (424)
- 13.4. Sposoby ustawiania wskaźników (426)
- 13.5. Parada kłamców, czyli o rzutowaniu const_cast (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 funkcji (446)

- 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 using i typedef 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 auto lub decltype 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 using lub typedef oraz typy enum (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: T, const T, volatile T (485)

- 15.7.6. Przeładowanie a typy: T*, volatile T*, const T* (486)
 - 15.7.7. Przeładowanie a typy: T&, volatile T&, const T& (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. Klasy (504)

- 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? (this) (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 using namespace 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. Statyczna funkcja składowa (566)
 - 16.18.1. Deklaracja składnika statycznego mająca inicjalizację "w klasie" (571)
- 16.19. Funkcje składowe typu const oraz volatile (577)

- 16.19.1. Przeładowanie a funkcje składowe const i volatile (581)
- 16.20. Struktura (582)
- 16.21. Klasa będąca agregatem. Klasa bez konstruktora (582)
- 16.22. Funkcje składowe z przydomkiem constexpr (585)
- 16.23. Specyfikator mutable (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 std::string to przecież nasz stary znajomy (611)
- 17.3. Definiowanie obiektów klasy string (612)
- 17.4. Użycie operatorów =, +, += w pracy ze stringami (617)
- 17.5. Pojemność, rozmiar i długość stringu (618)
 - 17.5.1. Bliźniacze funkcje size() i length() (618)
 - 17.5.2. Funkcja składowa empty (619)
 - 17.5.3. Funkcja składowa max_size (619)
 - 17.5.4. Funkcja składowa capacity (619)
 - 17.5.5. Funkcje składowe reserve i shrink_to_fit (621)
 - 17.5.6. resize - zmiana długości stringu "na siłę" (622)
 - 17.5.7. Funkcja składowa clear (624)
- 17.6. Użycie operatora [] oraz funkcji at (624)
 - 17.6.1. Działanie operatora [] (625)
 - 17.6.2. Działanie funkcji składowej at (626)
 - 17.6.3. Przebieganie po wszystkich literach stringu zakresowym for (629)
- 17.7. Funkcje składowe front i back (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 substr (636)
- 17.12. Szukanie zadanego substringu w obiekcie klasy string - funkcje find (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 - erase i pop_back (643)
- 17.16. Wstawianie znaków do istniejącego stringu - funkcje insert (644)
- 17.17. Zamiana części znaków na inne znaki - replace (646)
- 17.18. Zagłębienie do wnętrza obiektu klasy string funkcją data (649)
- 17.19. Zawartość obiektu klasy string a C-string (650)
- 17.20. W porządku alfabetycznym, czyli porównywanie stringów (653)
 - 17.20.1. Porównywanie stringów za pomocą funkcji compare (654)
 - 17.20.2. Porównywanie stringów przy użyciu operatorów ==, !=, <, >, <=, >= (658)
- 17.21. Zamiana treści stringu na małe lub wielkie litery (659)
- 17.22. Kopiowanie treści obiektu klasy string do tablicy znakowej - funkcja copy (662)
- 17.23. Wzajemna zamiana treści dwóch obiektów klasy string - funkcja swap (662)

- 17.24. Wczytywanie z klawiatury stringu o nieznanej wcześniej długości - getline (663)
 - 17.24.1. Pułapka, czyli jak getline może Cię zaskoczyć (666)
- 17.25. Iteratory stringu (670)
 - 17.25.1. Iterator do obiektu stałego (674)
 - 17.25.2. Funkcje składowe klasy string pracujące z iteratorami (675)
- 17.26. Klasa string korzysta z techniki przenoszenia (680)
- 17.27. Bryk, czyli "pamięć zewnętrzna" programisty (681)
- 17.28. Ćwiczenia (689)

18. Deklaracje przyjaźni (696)

- 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 otręźwiających słów na zakończenie (707)
- 18.8. Ćwiczenia (708)

19. Obsługa sytuacji wyjątkowych (710)

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

20. Klasa-składnik oraz klasa lokalna (737)

- 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 destruktory (761)

- 21.1. Konstruktor (761)
 - 21.1.1. Przykład programu zawierającego klasę z konstruktorami (762)
- 21.2. Specyfikator (przydomek) explicit (773)
- 21.3. Kiedy i jak wywoływany jest konstruktor (774)
 - 21.3.1. Konstruowanie obiektów lokalnych (774)
 - 21.3.2. Konstruowanie obiektów globalnych (775)
 - 21.3.3. Konstrukcja obiektów tworzonych operatorem new (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 rzucajcie wyjątków z destruktorów (781)
- 21.6. Konstruktor domniemany (783)
- 21.7. Funkcje składowe z przypiskami = default i = delete (784)
- 21.8. Konstruktorowa lista inicjalizacyjna składników klasy (786)
 - 21.8.1. Dla wtajemniczonych: wyjątki rzucane z konstruktorowej listy inicjalizacyjnej (793)
- 21.9. Konstruktor delegujący (797)
- 21.10. Pomocnicza klasa std::initializer_list - 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 constexpr mogą wytwarzać obiekty constexpr (828)
- 21.14. Ćwiczenia (838)

22. Konstruktory: kopiujący i przenoszący (841)

- 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 rwartości daje zezwolenie na recykling (859)
- 22.10. Funkcja std::move, która nie przenosi, a tylko rzutuje (862)
- 22.11. Odebrana rwartość 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. decltype - operator rozpoznawania typu bardzo wyszukanych wyrażeń (876)
- 22.16. Ćwiczenia (881)

23. Tablice obiektów (883)

- 23.1. Definiowanie tablic obiektów i praca z nimi (883)
- 23.2. Tablica obiektów definiowana operatorem new (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 klasy (899)

- 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 użytkownika (928)

- 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 operatorów (957)

- 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. Przeładowanie: =, [], (), -> (1015)

- 27.1. Cztery operatory, które muszą być niestatycznymi funkcjami składowymi (1015)
- 27.2. Operator przypisania = (wersja kopiująca) (1015)
 - 27.2.1. Przykład na przeładowanie (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. "Sprytny wskaźnik" wykorzystuje przeładowanie właśnie tego operatora (1051)
- 27.8. Ćwiczenia (1058)

----- TOM 3 -----

28. Przeładowanie operatorów new i delete na użytek klasy (1061)

- 28.1. Po co przeładowujemy operatory new i new[] (1061)
- 28.2. Funkcja operator new i operator new[] w klasie K (1062)
- 28.3. Jak się deklaruje operatory new i delete w klasie? (1065)
- 28.4. Przykładowy program z przeładowanymi new i delete (1067)
 - 28.4.1. Gdy dopuszczamy rzucanie wyjątku std::bad_alloc (1068)
 - 28.4.2. Po staremu nadal można (1073)
 - 28.4.3. Rezerwacja tablicy obiektów naszej klasy Twektorek (1073)
 - 28.4.4. Nasze własne argumenty wysłane do operatora new (1075)
 - 28.4.5. X Operatory new i delete 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 new, które nie rzucają wyjątku std::bad_alloc (1080)
- 28.5. Rzut oka wstecz na przeładowanie operatorów (1085)
- 28.6. Ćwiczenia (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. 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 `std::count_if` (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 `mutable` 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 `auto` (1147)
 - 30.5.1. Tworzenie obiektów na lambda słowem kluczowym `auto` (1148)
 - 30.5.2. Tworzenie (nazwanych) obiektów lambda szablonem `std::function` (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 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 `using`, 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 const (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 składowe (1257)

- 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 inline (1277)
- 32.9. Destruktor? Najlepiej wirtualny! (1277)
- 32.10. Pojedynek - funkcje przeładowane, zasłaniające się i wirtualne (zacierające się) (1279)

- 32.11. Kontekstowe słowa kluczowe override i final (1281)
 - 32.11.1. Przykład użycia override i final, a także wirtualnych destruktorów (1282)
- 32.12. Klasy abstrakcyjne (1294)
- 32.13. Wprawdzie konstruktor nie może być wirtualny, ale. (1301)
- 32.14. Rzutowanie dynamic_cast 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 - podstawy (1320)

- 33.1. Biblioteka iostream (1321)
- 33.2. Strumień (1321)
- 33.3. Strumienie zdefiniowane standardowo (1323)
- 33.4. Operatory >> i << (1324)
- 33.5. Domniemania w pracy strumieni zdefiniowanych standardowo (1325)
- 33.6. Uwaga na priorytet (1328)
- 33.7. Operatory << oraz >> 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 setw(int) (1351)
 - 33.11.4. Manipulator setfill (1354)
 - 33.11.5. Manipulator setprecision(int) (1354)
 - 33.11.6. Manipulator std::setbase(int) (1356)
 - 33.11.7. Manipulatory setiosflags, resetiosflags (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 setf, unsetf (1364)
- 33.14. Dodatkowe funkcje do zmiany parametrów formatowania (1370)
 - 33.14.1. Funkcja width (1371)
 - 33.14.2. Funkcja składowa fill (1372)
 - 33.14.3. Funkcja precision (1373)
 - 33.14.4. Funkcja copyfmt (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 read (1382)

- 33.16.3. Funkcja ignore (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 pisanie 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 string (1443)
 - 35.1.1. Przykłady ilustrujące użycie klasy ostream (1447)
- 35.2. Strumień czytający z obiektu klasy string (1450)
 - 35.2.1. Prosty przykład użycia strumienia istream (1452)
 - 35.2.2. Strumień istream a wczytywanie parametrów-danych (1455)
 - 35.2.3. Wczytywanie argumentów wywołania programu (1460)
- 35.3. Ożenek: strumień stringstream czytający i zapisujący do stringu (1464)
 - 35.3.1. Przykładowy program posługujący się klasą stringstream (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 Świątami 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 new (1534)
 - 37.9.4. Dyrektywa using 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 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. Ćwiczenia (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. Ćwiczenia (1605)

B. Skorowidz (1607)