

Spis treści

Przedmowa	11
O autorach	13
O korektorach merytorycznych	14
Wprowadzenie	15
Dla kogo przeznaczona jest ta książka?	16
Zawartość książki	16
Jak najlepiej skorzystać z tej książki?	17
Pobieranie plików z przykładowym kodem	18
Stosowane konwencje typograficzne	19
Rozdział 1. Krótkie wprowadzenie do języka C++	21
Dlaczego C++?	21
Bezkosztowe abstrakcje	22
Przenośność	24
Odporność	24
Język C++ dzisiaj	25
Porównanie C++ z innymi językami	25
Konkurencyjne języki i wydajność	25
Mechanizmy języka C++ niezwiązane z wydajnością	27
Wady języka C++	33
Biblioteki i kompilatory używane w tej książce	33
Podsumowanie	34

Rozdział 2. Podstawowe techniki języka C++	35
Automatyczne wykrywanie typów za pomocą słowa kluczowego <code>auto</code>	36
Stosowanie słowa kluczowego <code>auto</code> w sygnaturach funkcji	36
Stosowanie słowa kluczowego <code>auto</code> dla zmiennych	38
Semantyka przenoszenia	41
Tworzenie przez kopiowanie, wymienianie i przenoszenie	42
Pozyskiwanie zasobów i reguła pięciu	44
Zmienne nazwane i r-wartości	47
Domyślna semantyka przenoszenia i reguła zera	48
Stosowanie modyfikatora <code>&&</code> do funkcji składowych klas	52
Nie przenoś obiektów, jeśli kopiowanie i tak jest pomijane	52
Jeśli to możliwe, stosuj przekazywanie przez wartość	53
Projektowanie interfejsów z obsługą błędów	55
Kontrakty	56
Obsługa błędów	60
Obiekty funkcyjne i wyrażenia <code>lambda</code>	66
Podstawowa składnia <code>lambda</code> w języku C++	66
Klauzula przechwytywania	67
Przypisywanie do <code>lambda</code> wskaźników do funkcji języka C	73
Typy <code>lambda</code>	73
Lambdy i typ <code>std::function</code>	73
Generyczne lambdy	77
Podsumowanie	78
Rozdział 3. Analizowanie i pomiar wydajności	79
Złożoność asymptotyczna i notacja dużego O	80
Tempo wzrostu	84
Zamortyzowana złożoność czasowa	85
Co mierzyć i w jaki sposób?	88
Aspekty wydajności	90
Przyspieszanie wykonywania kodu	91
Liczniki wydajności	91
Testy wydajności — dobre praktyki	92
Poznaj kod i znajdź hot spoty	93
Profilery z instrumentacją	94
Profilery z próbkowaniem	96
Mikrotesty	98
Prawo Amdahla	99
Pułapki związane z mikrotestami	100
Przykład ilustrujący mikrotesty	101
Podsumowanie	106
Rozdział 4. Struktury danych	107
Cechy pamięci w komputerach	107
Kontenery z biblioteki standardowej	111
Kontenery sekwencyjne	112
Kontenery asocjacyjne	117
Adaptory kontenerów	121

Używanie widoków	123
Unikanie kopiowania dzięki typowi <code>string_view</code>	124
Unikanie utraty informacji o długości tablic dzięki typowi <code>std::span</code>	125
Uwagi na temat wydajności	126
Zapewnianie równowagi między gwarancjami złożoności a dodatkowymi kosztami	126
Znajomość i stosowanie odpowiednich funkcji API	127
Tablice równoległe	129
Podsumowanie	135
Rozdział 5. Algorytmy	136
Wprowadzenie do algorytmów z biblioteki standardowej	137
Ewolucja algorytmów z biblioteki standardowej	137
Rozwiązywanie codziennych problemów	138
Iteratory i zakresy	144
Wprowadzenie do iteratorów	144
Wartość wartownika i iteratory zakończone	145
Zakresy	146
Kategorie iteratorów	147
Cechy algorytmów standardowych	149
Algorytmy nie zmieniają wielkości kontenera	149
Algorytmy zwracające dane wyjściowe wymagają zaalokowanych danych	150
Algorytmy domyślnie używają funkcji <code>operator==()</code> i <code>operator<()</code>	152
W algorytmach z ograniczeniami używane są projekcje	153
Algorytmy wymagają, aby operatory przenoszące nie zgłaszały wyjątków	153
Algorytmy mają gwarantowaną złożoność	154
Algorytmy działają równie dobrze jak analogiczne funkcje z bibliotek języka C	155
Pisanie i stosowanie algorytmów generycznych	155
Algorytmy niegeneryczne	156
Algorytmy generyczne	156
Struktury danych, które mogą być używane przez algorytmy generyczne	157
Dobre praktyki	159
Stosowanie algorytmów z ograniczeniami	159
Sortowanie tylko tych danych, które będą pobierane	159
Stosowanie algorytmów standardowych zamiast surowych pętli <code>for</code>	162
Unikanie tworzenia kopii kontenera	168
Podsumowanie	169
Rozdział 6. Zakresy i widoki	170
Powody powstania biblioteki <code>Ranges</code>	170
Ograniczenia biblioteki <code>Algorithm</code>	171
Widoki z biblioteki <code>Ranges</code>	173
Widoki można łączyć w łańcuch	174
Widoki zakresów i adaptory zakresów	175
Widoki są zakresami bez własności elementów oferującymi gwarancje złożoności	176
Widoki nie modyfikują podstawowego kontenera	176
Widoki można materializować do postaci kontenerów	177
Widoki są przetwarzane leniwie	178

Widoki z biblioteki standardowej	179
Widoki dla zakresów	179
Jeszcze o typach <code>std::string_view</code> i <code>std::span</code>	182
Przyszłość biblioteki Ranges	183
Podsumowanie	183
Rozdział 7. Zarządzanie pamięcią	184
Pamięć w komputerze	185
Wirtualna przestrzeń adresowa	185
Strony pamięci	185
Migotanie	186
Pamięć procesu	187
Pamięć na stosie	188
Pamięć na sterpie	191
Obiekty w pamięci	192
Tworzenie i usuwanie obiektów	192
Wyrównanie pamięci	195
Dopełnienie	199
Własność pamięci	201
Pośrednie zarządzanie zasobami	202
Kontenery	204
Inteligentne wskaźniki	204
Optymalizacja małych obiektów	207
Niestandardowe zarządzanie pamięcią	210
Tworzenie puli pamięci	211
Niestandardowy alokator pamięci	214
Stosowanie polimorficznych alokatorów pamięci	218
Implementowanie niestandardowego zasobu pamięciowego	222
Podsumowanie	223
Rozdział 8. Programowanie z przetwarzaniem w czasie kompilacji	224
Wprowadzenie do metaprogramowania z użyciem szablonów	225
Tworzenie szablonów	226
Stosowanie liczb całkowitych jako parametrów szablonu	228
Tworzenie specjalizacji szablonu	228
Jak kompilator przetwarza funkcję szablonową?	229
Skrócone szablony funkcji	229
Pobieranie typu zmiennej za pomocą specyfikatora <code>decltype</code>	230
Cechy typów	231
Kategorie cech typów	231
Stosowanie cech typów	232
Programowanie z użyciem stałych wyrażeń	233
Działanie funkcji ze słowem kluczowym <code>constexpr</code> w czasie wykonywania programu	234
Deklarowanie funkcji natychmiastowych za pomocą słowa kluczowego <code>constexpr</code>	235
Instrukcja <code>if constexpr</code>	235
Sprawdzanie błędów programistycznych w czasie kompilacji	239

Ograniczenia i koncepty	240
Szablon Point2D bez ograniczeń	241
Omówienie składni ograniczeń i konceptów	243
Wersja szablonu Point2D z ograniczeniami	246
Dodawanie ograniczeń do kodu	247
Koncepty w bibliotece standardowej	248
Praktyczne przykłady metaprogramowania	248
Przykład nr 1: tworzenie generycznej bezpiecznej funkcji rzutowania	248
Przykład nr 2: obliczanie skrótów łańcuchów znaków w czasie kompilacji	251
Podsumowanie	256
Rozdział 9. Podstawowe narzędzia	257
Reprezentowanie wartości opcjonalnych za pomocą typu std::optional	258
Opcjonalne zwracane wartości	258
Opcjonalne zmienne składowe	259
Unikanie pustych stanów w wyliczeniach	259
Sortowanie i porównywanie wartości typu std::optional	260
Kolekcje niejednorodne o stałej wielkości	261
Stosowanie typu std::pair	261
Typ std::tuple	262
Kolekcje niejednorodne o dynamicznie zmienianej wielkości	270
Typ std::variant	271
Kolekcje niejednorodne z obiektami typu std::variant	275
Dostęp do wartości z kontenera elementów typu VariantType	276
Praktyczne przykłady	277
Przykład nr 1: projekcje i operatory porównania	277
Przykład nr 2: refleksja	278
Podsumowanie	281
Rozdział 10. Obiekty pośredniczące i leniwe przetwarzanie	282
Wprowadzenie do leniwego przetwarzania i obiektów pośredniczących	282
Przetwarzanie leniwe i przetwarzanie zachłanne	283
Obiekty pośredniczące	284
Stosowanie obiektów pośredniczących do zapobiegania tworzeniu obiektów	284
Porównywanie skalanych łańcuchów znaków z wykorzystaniem obiektu pośredniczącego	284
Implementowanie obiektu pośredniczącego	285
Modyfikator r-wartości	286
Przypisywanie połączonej wartości do obiektu pośredniczącego	287
Ocena wydajności	287
Odraczanie obliczania kwadratów	288
Prosta klasa reprezentująca wektory dwuwymiarowe	288
Obliczenia matematyczne	289
Implementowanie klasy LengthProxy	291
Porównywanie długości za pomocą klasy LengthProxy	292
Obliczanie długości za pomocą klasy LengthProxy	293
Ocena wydajności	294
Pomysłowe przeciążanie operatorów i obiekty pośredniczące	295
Operator potoku jako metoda rozszerzająca	296
Podsumowanie	297

Rozdział 11. Współbieżność	298
Podstawy współbieżności	299
Co sprawia, że programowanie współbieżne jest trudne?	299
Współbieżność a równoległość	300
Porcjowanie czasu	300
Współdzielona pamięć	302
Sytuacja wyścigu	303
Muteks	305
Zakleszczenie	306
Zadania synchroniczne i zadania asynchroniczne	307
Programowanie współbieżne w języku C++	308
Biblioteka obsługi wątków	308
Dodatkowe podstawowe mechanizmy synchronizacji w standardzie C++20	320
Obsługa zmiennych atomowych w języku C++	329
Model dostępu do pamięci w języku C++	337
Programowanie bez blokad	341
Przykład: kolejka bez blokad	341
Wskazówki dotyczące wydajności	344
Zapobieganie rywalizacji	344
Unikanie operacji blokujących	344
Liczba wątków i rdzeni procesora	345
Priorytety wątków	345
Koligacja wątków	346
Niezamierzone współdzielenie	347
Podsumowanie	347
Rozdział 12. Korutyny i leniwe generatory	348
Kilka przykładów uzasadniających stosowanie korutyn	349
Abstrakcja reprezentująca korutyny	351
Podprocedury i korutyny	351
Wykonywanie podprocedur i korutyn w procesorze	353
Korutyny bezstosowe i korutyny stosowe	360
Czego Czytelnik nauczył się do tego miejsca?	363
Korutyny w języku C++	363
Co zostało uwzględnione w standardzie języka C++ (i co zostało pominięte)?	364
Co sprawia, że funkcja w języku C++ jest korutyną?	365
Minimalny, lecz kompletny przykład	366
Alokowanie stanu korutyny	371
Zapobieganie wiszącym referencjom	372
Obsługa błędów	376
Punkty konfiguracyjne	377
Generatory	377
Implementowanie generatora	377
Stosowanie klasy Generator	381
Praktyczny przykład zastosowania generatorów	387
Wydajność	393
Podsumowanie	393

Rozdział 13. Programowanie asynchroniczne z użyciem korutyn	395
Jeszcze o typach awaitable	396
Automatyczne punkty wstrzymywania	397
Implementowanie prostego typu reprezentującego zadanie	398
Obsługa zwracanych wartości i wyjątków	400
Wznawianie oczekującej korutyny	401
Dodawanie obsługi zadań zwracających void	403
Synchroniczne oczekiwanie na ukończenie zadania	404
Testowanie asynchronicznych zadań z użyciem funkcji sync_wait()	408
Tworzenie nakładki na API opartym na wywołaniach zwrotnych	409
Współbieżny serwer zbudowany za pomocą biblioteki Boost.Asio	412
Implementowanie serwera	412
Uruchamianie serwera i nawiązywanie z nim połączenia	414
Co osiągnęliśmy za pomocą serwera (i czego wciąż brakuje)?	415
Podsumowanie	416
Rozdział 14. Algorytmy równoległe	417
Znaczenie równoległości	418
Algorytmy równoległe	418
Ocena algorytmów równoległych	418
Jeszcze o prawie Amdahla	419
Implementowanie równoległego algorytmu std::transform()	420
Implementowanie równoległej wersji algorytmu std::count_if()	429
Implementowanie równoległej wersji algorytmu std::copy_if()	430
Algorytmy równoległe z biblioteki standardowej	436
Strategie wykonywania	436
Obsługa wyjątków	440
Dodatki i zmiany w algorytmach równoległych	441
Zrównoleglanie pętli for opartej na indeksie	443
Wykonywanie algorytmów w procesorze graficznym	444
Podsumowanie	445