helion.pl
KSIĘGARNIA INTERNETOWA WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

C++. Kanony wiedzy programistycznej

Autor: Stephen C. Dewhurst
Tłumaczenie: Przemysław Szeremiota
ISBN: 83-246-0024-8
Tytuł oryginału: C++ Common Knowledge: Essential Intermediate Programming
Format: B5, stron: 248

Wiadomości niezbędne każdemu programiście

- Zasady programowania obiektowego
- Stosowanie wzorców projektowych
- Korzystanie z mechanizmu szablonów

C++ jest jednym z najpopularniejszych języków programowania. Jego potężne możliwości idą w parze ze złożonością, która powoduje, że nauka programowania w C++ nie jest łatwym zadaniem. Programista, który chce opanować zasady tworzenia aplikacji w C++, musi w pełni opanować przynajmniej część związanych z tym językiem pojęć i technik. Napisanie prawidłowo i wydajnie działającego programu wymaga wykorzystania tej wiedzy w praktyce.

„C++. Kanony wiedzy programistycznej" to przegląd zagadnień, których znajomość jest nieodzowna dla każdego programisty korzystającego z C++. Czytając tę książkę, poznasz reguły projektowania i programowania obiektowego, sposoby wykorzystywania funkcji i szablonów oraz zasady stosowania wzorów projektowych. Przeczytasz o dyrektywach kompilatora, wskaźnikach i rzutowaniu. Dowiesz się wszystkiego, co jest uważane za sedno języka C++.

- Projektowanie obiektowe
- Polimorfizm
- Wykorzystywanie podstawowych wzorców projektowych
- Deklarowanie funkcji i tablic
- Zarządzanie pamięcią
- Sterowanie przebiegiem kompilacji
- Korzystanie z szablonów
- Obsługa błędów za pomocą wyjątków

Ta książka pozwoli Ci się stać programistą doskonałym

O autorze:
Stephen C. Dewhurst był jednym z pierwszych użytkowników języka C++ w laboratoriach Bell Labs. Ma ponad dwudziestoletnie doświadczenie w stosowaniu C++ do rozwiązywania problemów w takich dziedzinach, jak projektowanie kompilatorów, zabezpieczanie handlu elektronicznego czy telekomunikacja implementowana na bazie urządzeń wbudowanych. Jest autorem książki C++ Gotchas (Addison-Wesley, 2003) i współautorem książki Programming in C++, Second Edition (Prentice Hall, 1995). Dewhurst jest członkiem grupy doradczej przy The C++ Source i współredaktorem magazynu C/C++ Users Journal; wcześniej prowadził swoją kolumnę w C++ Report. Jest również twórcą dwóch kompilatorów języka C++ i wielu artykułów traktujących o projektowaniu kompilatorów i technikach programowania w C++.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Podziękowania 7
Wstęp 9
Konwencje typograficzne 15
Zagadnienie 1. Abstrakcje 17
Zagadnienie 2. Polimorfizm 19
Zagadnienie 3. Wzorce projektowe 23
Zagadnienie 4. Standardowa biblioteka szablonów 27
Zagadnienie 5. Referencje są aliasami, nie wskaźnikami 29
Zagadnienie 6. Parametry tablicowe 33
Zagadnienie 7. Wskaźniki const i wskaźniki na const 37
Zagadnienie 8. Wskaźniki na wskaźniki 41
Zagadnienie 9. Nowe operatory rzutowania 43
Zagadnienie 10. Semantyka metod deklarowanych z const 47
Zagadnienie 11. Kompilator nadziewa klasy farszem 51
Zagadnienie 12. Przypisanie to nie to samo co inicjalizacja 55
Zagadnienie 13. Operacje kopiowania 59
Zagadnienie 14. Wskaźniki funkcji 63
Zagadnienie 15. Wskaźniki składowych klas nie są wskaźnikami 67
Zagadnienie 16. Wskaźniki metod nie są wskaźnikami 71
Zagadnienie 17. Deklaratory funkcji i tablic 75
Zagadnienie 18. Obiekty funkcyjne 77
Zagadnienie 19. Wzorzec Command i dewiza hollywoodzka 81
Zagadnienie 20. Obiekty funkcyjne STL 85

Zagadnienie 21. Przeciążanie to nie to samo co przesłanianie ... 89
Zagadnienie 22. Wzorzec Template Method ... 91
Zagadnienie 23. Przestrzenie nazw ... 93
Zagadnienie 24. Wyszukiwanie metod ... 97
Zagadnienie 25. Wyszukiwanie ADL ... 99
Zagadnienie 26. Wyszukiwanie funkcji operatorów ... 101
Zagadnienie 27. Odpytywanie klasy ... 103
Zagadnienie 28. Semantyka porównywania wskaźników ... 107
Zagadnienie 29. Konstruktory wirtualne i wzorzec Prototype ... 109
Zagadnienie 30. Wzorzec Factory Method ... 113
Zagadnienie 31. Kowariancja typów zwracanych ... 117
Zagadnienie 32. Blokowanie kopiowania ... 121
Zagadnienie 33. Wytwarzanie abstrakcyjnych klas bazowych ... 123
Zagadnienie 34. Blokowanie przydziału na stercie ... 125
Zagadnienie 35. Miejscowa wersja new ... 127
Zagadnienie 36. Zarządzanie pamięcią w klasie ... 131
Zagadnienie 37. Przydział tablicowy ... 135
Zagadnienie 38. Aksjomaty odporności na wyjątki ... 139
Zagadnienie 39. Funkcje odporne na wyjątki ... 143
Zagadnienie 40. Reguła RAII ... 147
Zagadnienie 41. Operator new, konstruktory i wyjątki ... 151
Zagadnienie 42. Inteligentne wskaźniki ... 153
Zagadnienie 43. Niezwykłości auto_ptr ... 155
Zagadnienie 44. Arytmetyka wskaźników ... 157
Zagadnienie 45. Terminologia szablonów ... 161
Zagadnienie 46. Jawna specjalizacja szablonu klasy ... 163
Zagadnienie 47. Częściowa specjalizacja szablonu ... 167
Zagadnienie 48. Specjalizacja metody szablonu klasy ... 171
Zagadnienie 49. Niepewność co do nazw typów ... 175
Zagadnienie 50. Szablony składowych ... 179
Zagadnienie 51. Niepewność co do nazw szablonów ... 183
Zagadnienie 52. Specjalizacja dla informacji o typie ... 185

Zagadnienie 53. Osadzanie informacji o typie ... 189
Zagadnienie 54. Klasy cech ... 193
Zagadnienie 55. Szablony parametrów szablonu ... 199
Zagadnienie 56. Klasy wytycznych ... 205
Zagadnienie 57. Dedukcja argumentów szablonu ... 209
Zagadnienie 58. Przeciążanie szablonów funkcji ... 213
Zagadnienie 59. Reguła SFINAE ... 217
Zagadnienie 60. Algorytmy uogólnione ... 221
Zagadnienie 61. Konkretyzuje się tylko to, co używane ... 225
Zagadnienie 62. Bariery #include ... 229
Zagadnienie 63. Opcjonalne słowa kluczowe ... 231
Bibliografia ... 235
Skorowidz ... 237

Zagadnienie 7.

Wskaźniki const i wskaźniki na const

W swobodnych konwersacjach programiści C++ często mówiąc „wskaźnik const" mają tak naprawdę na myśli „wskaźnik na const". Szkoda, bo to dwa różne pojęcia.

```
T *pt = new T;          // wskaźnik na T
const T *pct = pt;      // wskaźnik na const T
T *const cpt = pt;      // wskaźnik const na T
```

Zanim zaczniemy gorączkowo tasować const w deklaracjach wskaźników, powinniśmy najpierw zdecydować, o co nam chodzi. Co ma być stałe: wskaźnik, obiekt wskazywany, czy może oba? W deklaracji pct wskaźnik nie jest stały, stały (niemodyfikowalny) ma być za to obiekt przezeń wskazywany; kwalifikator const modyfikuje typ bazowy T, a nie modyfikator wskaźnika *. W przypadku cpt deklarowany jest niemodyfikowalny (stały) wskaźnik obiektu modyfikowalnego; kwalifikator const modyfikuje tu modyfikator wskaźnika *, zostawiając w spokoju typ bazowy T.

Aby jeszcze zaciemnić składnię wskaźników i kwalifikatora const, można powiedzieć, że kolejność specyfikatorów deklaracji, czyli kolejność tego, co występuje w deklaracji wskaźnika przed pierwszym modyfikatorem *, jest nieistotna. Na przykład, obie poniższe deklaracje wprowadzają do programu zmienne dokładnie tego samego typu:

```
const T *p1;    // wskaźnik na const T
T const *p2;    // również wskaźnik na const T
```

Pierwsza postać jest bardziej klasyczna, ale wielu ekspertów C++ zaleca teraz stosowanie postaci drugiej. W uzasadnieniu podnoszą, że owa postać zmniejsza ryzyko nieporozumień, bo można sobie pomóc odczytaniem deklaracji od tyłu: „wskaźnik na const T". Wybór jednej z powyższych form jest dowolny, ważne, aby trzymać się raz przyjętej konwencji. Słowem, należy uważać na częsty błąd mylenia deklaracji wskaźnika stałego z deklaracją wskaźnika na obiekt stały (niemodyfikowalny).

```
T const * p3;           // wskaźnik na obiekt stały
T *const p4 = pt;       // stały wskaźnik na obiekt modyfikowalny
```

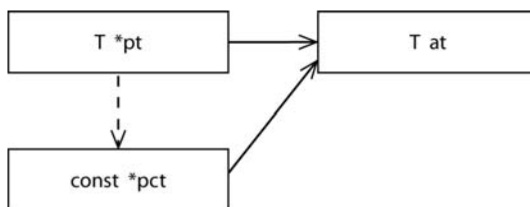
Możliwe jest oczywiście zadeklarowanie również stałego (`const`) wskaźnika na obiekt stały (`const`):

```
const T * const cpct1 = pt;     // wszystko const
T const * const cpct2 - cpct1; // identyczny typ
```

Zauważmy, że często zamiast wskaźnika `const`, prościej zastosować referencję:

```
const T &rct = *pt;      // zamiast const T * const
T &rt = *pt;             // zamiast T *const
```

W niektórych z powyższych przykładów mieliśmy możliwość konwersji wskaźnika bez `const` (modyfikowalnego) na wskaźnik `const` (niemodyfikowalny). Możemy na przykład zainicjalizować `pct` (typu `const T *`) wartością `pt` (typu `T *`). Jest to dozwolone, ponieważ — mówiąc potocznie — nie grozi to niczym złym. Sprawdźmy, co stanie się w wyniku skopiowania adresu obiektu modyfikowalnego do wskaźnika na obiekt niemodyfikowalny (rysunek 7.1).



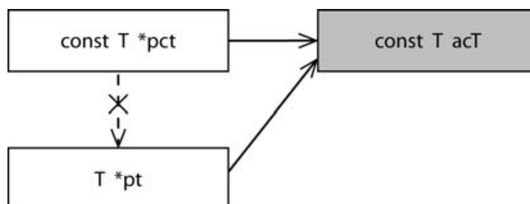
Rysunek 7.1. *Wskaźnik na const może wskazywać obiekt modyfikowalny (bez const)*

Wskaźnik `pct` (wskaźnik na `const`) wskazuje obiekt typu `T` niebędący stałym (bez `const`), ale nie wprowadza to żadnej sprzeczności. W rzeczy samej, sytuacja odwoływania się do obiektów modyfikowalnych za pośrednictwem wskaźników na `const` jest dość częsta:

```
void aFunc( const T *arg1, const T &arg2 );
// ...
T *a = new T;
T b;
aFunc( a, b );
```

Przy wywołaniu `aFunc` następuje inicjalizacja `arg1` adresem `a` i `arg2` adresem `b`. Nie sposób jednak powiedzieć, czy `a` faktycznie wskazuje obiekt niemodyfikowalny, albo czy `b` jest `const`; zakładamy jedynie, że tak będą traktowane w obrębie funkcji `aFunc`. To bardzo użyteczne.

Konwersja odwrotna, czyli ze wskaźnika na obiekt niemodyfikowalny do postaci wskaźnika na obiekt modyfikowalny nie jest dozwolona, bo byłaby niebezpieczna (patrz rysunek 7.2).



Rysunek 7.2. *Wskaźnik na obiekt bez const nie może odnosić się do obiektu const*

W takim układzie pct może faktycznie wskazywać obiekt niemodyfikowalny, zdefiniowany z kwalifikatorem const. Gdybyśmy ten wskaźnik chcieli skonwertować do postaci wskaźnika na obiekt modyfikowalny, umożliwilibyśmy potencjalnie zmianę wartości acT za pośrednictwem pt:

```
const T acT;
pct = &act;
pt = pct;    // błąd, na szczęście…
*pt = aT;    // próba zmiany obiektu const!
```

Standard C++ mówi, że takie przypisanie będzie dawało niezdefiniowane rezultaty; nie wiadomo więc dokładnie, co się stanie, ale zapewne nic dobrego. Oczywiście możemy wykonać konwersję uciekając się do jawnego rzutowania:

```
pt = const_cast<T *>(pct);  // odradzane, choć to nie błąd
*pt = aT;                   // próba zmiany obiektu const!
```

Jednak jeśli pt będzie odnosić się do obiektu, który (jak acT) został zadeklarowany jako niemodyfikowalny (const), efekt przypisania wciąż będzie niezdefiniowany (zobacz „Nowe operatory rzutowania” [9, 43]).