Dawid Farbaniec

C++20

Laboratorium

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek 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 wydawca 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 wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne w książce i na okładce zostały wykorzystane
za zgodą Shutterstock.com

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: https://helion.pl (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/cpp20l
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-8838-3

Copyright © Helion S.A. 2022

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

SPIS TREŚCI

SŁOWEM WSTĘPU ____ 11

CZĘŚĆ 1. WPROWADZENIE

ROZDZIAŁ 1.
SCHEMAT BLOKOWY I PSEUDOKOD ____ 15

1.1. Infekcja plików przez wirusy komputerowe ____ 15

ROZDZIAŁ 2.
KOD ŹRÓDŁOWY PROGRAMU I KOMPILACJA ____ 19

CZĘŚĆ 2. C++, CZYLI POZNAJ JĘZYK HAKERÓW

ROZDZIAŁ 3.
NOT-A-VIRUS.VIRAL. HELLO.MSVC++ ____ 23

ROZDZIAŁ 4.
BUDOWANIE I URUCHAMIANIE PROJEKTU ____ 27

Kup ksi k

Pole ksi k

ROZDZIAŁ 5.
KOMENTARZE W JĘZYKU C++ ____ 38

ROZDZIAŁ 6.
TYPY DANYCH, ZMIENNE I STAŁE ____ 40

6.1. Zmienne ____ 41
6.2. Stałe ____ 45
6.3. Zakresy zmiennych ____ 46

ROZDZIAŁ 7.
TYPY PODSTAWOWE ____ 50

7.1. Inicjalizacja ____ 50
7.2. Typy całkowitoliczbowe ____ 52
7.3. Typy zmiennoprzecinkowe ____ 56
7.4. Typy znakowe ____ 59
7.5. Typ logiczny ____ 62
7.6. Typ wyliczeniowy ____ 63
7.7. Typ void ____ 65

ROZDZIAŁ 8.
DEFINIOWANIE WŁASNYCH NAZW TYPÓW ____ 66

ROZDZIAŁ 9.
DEDUKCJA TYPU ____ 68

Kup ksi k

Pole ksi k

ROZDZIAŁ 10.
RZUTOWANIE I KONWERSJA TYPÓW 71
10.1. static_cast 75
10.2. const_cast 76
10.3. dynamic_cast 78
10.4. reinterpret_cast 79

ROZDZIAŁ 11.
NAPISY 82
11.1. Surowe napisy 85

ROZDZIAŁ 12.
STRUKTURY 86
12.1. Pola bitowe 89

ROZDZIAŁ 13.
UNIE 91

ROZDZIAŁ 14.
INSTRUKCJE STERUJĄCE PRZEPŁYWEM 94
14.1. Instrukcja warunkowa if 94
14.2. Instrukcja warunkowa switch 99
14.3. Grupowanie warunków 101

Kup ksi k Pole ksi k

ROZDZIAŁ 15.
PĘTLE 104
15.1. Instrukcja for 104
15.2. Instrukcja while 107
15.3. Instrukcja do-while 110
15.4. Instrukcje break i continue 111
15.5. Instrukcja goto 112

ROZDZIAŁ 16.
OPERATORY 115
16.1. Operatory logiczne i bitowe 117
16.2. Inkrementacja i dekrementacja 119
16.3. Operator ternarny 121
16.4. Priorytety operatorów 122
16.5. Przeciążanie operatorów 127

ROZDZIAŁ 17.
TABLICE I WSKAŹNIKI 130
17.1. Tablice w stylu języka C 130
17.2. Kontener std::array 132
17.3. Wskaźniki do tablic 134
17.4. Wskaźniki void oraz nullptr 137
17.5. Referencje (odwołania) 138
17.6. Inteligentne wskaźniki 142

Kup ksi k Pole ksi k

ROZDZIAŁ 18.
FUNKCJE 150

18.1. Definiowanie funkcji 151
18.2. Argumenty i zwracanie wartości 157
18.3. Argumenty domyślne 164
18.4. Przeciążanie funkcji 167
18.5. Funkcje i zmienne inline 168
18.6. Wskaźnik na funkcję 173
18.7. Funkcje ze zmienną liczbą argumentów 177
18.8. Wyrażenia lambda 180
18.9. Koprocedury (ang. coroutines) 183

ROZDZIAŁ 19.
KLASY I OBIEKTY 188

19.1. Klasy i obiekty 189
19.2. Operator dostępu 191
19.3. Modyfikatory dostępu 193
19.4. Słowo kluczowe this 195
19.5. Składowe statyczne 196
19.6. Konstruktor i destruktor 198
19.7. Klasy pochodne i zagnieżdżone 204
19.8. Elementy stałe, zmienne i ulotne 208
19.9. Jawne usuwanie funkcji 209
19.10. Przeciążanie operatorów w klasach oraz trójdrożny operator porównania (<=>) 210
19.11. Przyjaciele 213

Kup ksi k

Pole ksi k

19.12. Funkcje wirtualne ____ 213
19.13. Klasy abstrakcyjne ____ 217

ROZDZIAŁ 20.
PRZESTRZENIE NAZW ____ 220

20.1. Tworzenie przestrzeni nazw ____ 220
20.2. Dyrektywa using ____ 222
20.3. Aliasy przestrzeni nazw ____ 223

ROZDZIAŁ 21.
SZABLONY ____ 224

21.1. Szablony zmiennych ____ 224
21.2. Szablony klas ____ 226
21.3. Szablony funkcji ____ 229
21.4. Szablony w wyrażeniach lambda ____ 229
21.5. Wymagania nazwane (słowo kluczowe „concept”) ____ 229

ROZDZIAŁ 22.
OBSŁUGA WYJĄTKÓW ____ 236

22.1. Blok try-catch ____ 236
22.2. Rzucanie wyjątku ____ 238

Kup ksi k

Pole ksi k

CZĘŚĆ 3. PRZYKŁADOWA APLIKACJA W C++/WINRT DLA UNIVERSAL WINDOWS PLATFORM

ROZDZIAŁ 23.
WITAJ, ŚWIECIE C++/WINRT! 243
23.1. MainPage.xaml 244
23.2. MainPage.cpp 246
23.3. Uruchomienie rozwiązania 248

ROZDZIAŁ 24.
NAUKA C++, CO DALEJ? 251

DODATKI

DODATEK 1.
VISUAL STUDIO — WYBÓR WERSJI STANDARDU ISO JĘZYKA C++ DLA PROJEKTU 255

DODATEK 2.
DEFINICJA CZY DEKLARACJA? (C++) 257

DODATEK 3.
GRA KOMPUTEROWA 259

BIBLIOGRAFIA 260

Kup ksi k

Pole ksi k

ROZDZIAŁ 10.

RZUTOWANIE I KONWERSJA TYPÓW

Tak jak w niektórych językach programowania nie ma znaczenia, czy obiekt jest liczbą, napisem, czy jeszcze innym typem, tak w języku C++ istnieje kontrola typów danych. Prędzej czy później można się spotkać z sytuacją, że mamy element (np. zmienną) innego typu, niż potrzebujemy. Prostym przykładem może być sytuacja, w której aplikacja pobrała od użytkownika tekst (z klawiatury), który z założenia powinien być liczbą, gdyż przeprowadzane będą na nim operacje arytmetyczne. Pomijając błędne wartości (użytkownik podał np. literę `'A'` zamiast liczby), mamy teoretycznie poprawne dane, ale takie, które są tekstem, a nie liczbą (różnią się typem). Tak więc zarówno w tym, jak i w wielu innych przypadkach zachodzi potrzeba dokonania konwersji typów nazywanej też rzutowaniem.

Kod źródłowy na rysunku 10.1 odnosi się tylko do wartości liczbowych. Charakteryzuje się następującymi cechami:

- Notacja klamrowa dla definiowania zmiennych jest bezpieczniejsza niż operator przypisania (znak równości).
- Typ liczbowy o mniejszym zakresie możemy bezpiecznie wpisać do typu o większym zakresie (przedziale wartości).

Kup ksi k Pole ksi k

```
#include <iostream>

int main()
{
    /* Maksymalna wartość dla typu „unsigned int” */
    constexpr auto max =
                 std::numeric_limits<unsigned>::max();

    char8_t at { u8'@' };

    /* Bezpieczna konwersja typu „char8_t”
       na typ „unsigned int” przez notację klamrową */
    unsigned mrAt { at };

    std::cout << "@ = " << mrAt << std::endl;

    /* Tutaj dzięki notacji klamrowej otrzymamy
       błąd przy próbie kompilacji, gdyż
       przypisujemy typ o większym zakresie
       do typu o mniejszym zakresie */
    //char8_t bad1 { unsigned { max } }; //błąd

    /* Niejawna konwersja z typu „unsigned int”
       na typ „char8_t” zniszczy oryginalną
       wartość zmiennej „max” */
    char8_t bad2 = max; //błąd (przycięcie wartości)

    /* Konwersja notacją z nawiasami (w stylu języka C)
       nie wypisze oryginalnej wartości zmiennej „max” */
    std::cout << "bad2 = " << (unsigned)bad2 << std::endl;

    return EXIT_SUCCESS;
}
```

```
Konsola debugowania programu Microsoft Visual Studio
@ = 64
bad2 = 255
```

Rysunek 10.1. Notacja klamrowa bezpieczna dla typów
i przykładowy błąd przycięcia wartości liczbowej w języku C++

- Przypisanie wartości spoza zakresu do zmiennej liczbowej może spowodować błąd taki jak np. przycięcie wartości, przez co traci się oryginalną wartość zmiennej.

Kup ksi k   Pole ksi k

Język C++ pozwala na rzutowanie (konwersję typów) w stylu swojego poprzednika — języka C, ale nie tylko. Rzutowanie wywodzące się z języka C ma składnię:

```
(docelowy-typ)wyrażenie
```

czyli na przykład:

```
float my_float = 2.5f;
int my_int = (int)my_float; //nie polecam [przyp. Mr. At]
```

Lepszym sposobem jest unikanie rzutowania i korzystanie z funkcji przeznaczonych do konwersji wartości. Czasami jednak trzeba świadomie dokonać konwersji i wtedy w języku C++ można skorzystać z rzutowania typu:

- `static_cast<docelowy-typ>(wyrażenie)` — pozwala na konwersję pomiędzy typami, które są dla siebie pokrewne („podobne”), a sama konwersja jest kontrolowana przy budowaniu programu.
- `const_cast<docelowy-typ>(wyrażenie)` — dotyczy konwersji pomiędzy typami, które różnią się od siebie zastosowaniem słów kluczowych const i volatile. Pozwala to „ze stałej stworzyć zmienną”, ale nie zawsze w bezpieczny sposób [!], o czym będzie dalej.
- `dynamic_cast<docelowy-typ>(wyrażenie)` — dotyczy konwersji pomiędzy typami użytkownika (klasami). Jest to dość rozległy temat. Jednak aby mieć jakieś wyobrażenie o tym, to można podać przykład, że istnieje typ Urządzenie i dziedziczący typ Drukarka, a tym rzutowaniem można z Urządzenia zrobić Drukarkę oraz vice versa. Kontrola następuje w czasie działania programu.
- `reinterpret_cast<docelowy-typ>(wyrażenie)` — jest to niekontrolowana konwersja typów. Można w ten sposób np. z ciągu bajtów zrobić kod programu i go wykonać (rysunek 10.2).

Kup ksi k Pole ksi k

```
#define ℂ++ c_plus_plus
visual_ℂ++::fanatic_mode = true;
//początek treści dotyczącej Visual C++
```

```
 1  #include <cstddef>
 2  #include <Windows.h>
 3
 4  using ExecuteFunc = void(*)();
 5  std::byte bytes[] =
 6  {   //0xC3 = ret opcode
 7      std::byte { 0xC3 }
 8  };
 9
10  int main()
11  {
12      DWORD prevProtect = NULL;
13
14      VirtualProtect(bytes, sizeof(bytes),
15          PAGE_EXECUTE_READ, &prevProtect);
16
17      ExecuteFunc(ExecuteRawMemory) =
18          reinterpret_cast<ExecuteFunc>(&bytes);
19
20      ExecuteRawMemory();
21
22      return EXIT_SUCCESS;
23  }
```

```
visual_ℂ++::fanatic_mode = false;
//koniec treści dotyczącej Visual C++
```

Rdzeniem powyższego przykładu jest ustawienie zmiennej `bytes` praw do wykonywania się jako kod, a następnie rzutowanie ciągu bajtów (zmiennej `bytes`) na wskaźnik do funkcji w celu wywołania (wykonania). Jest to przykład pozwalający wykonać rozkazy procesora (bajty) jako kod programu. Kod powyżej (ciąg bajtów) zawiera tylko jedną instrukcję (ret), która po prostu wraca z wywołanej funkcji.

Rysunek 10.2. Konwersja (rzutowanie reinterpret_cast) ciągu bajtów na wskaźnik na funkcję oraz wywołanie (wykonanie) tej funkcji (Visual C++)

Kup książkę Poleć książkę

10.1. STATIC_CAST

Przykład konwersji pomiędzy pokrewnymi typami danych przedstawiono na rysunku 10.3. Można zauważyć, że typ logiczny `bool` ma wartość `false` dla liczby zero, a dla innych liczb przyjmuje wartość `true`.

```
#include <iostream>

int main()
{
    /* Dwie zmienne logiczne */
    bool truth { true };
    bool falsehood { false };

    /* Rzutowanie typu logicznego na liczbę całkowitą */
    auto truthNum { static_cast<unsigned int>(truth) };
    auto falseNum { static_cast<unsigned int>(falsehood) };

    /* Wartość „true”  jako liczba całkowita to „1”   */
    std::cout << "true = " << truthNum << std::endl;

    /* Wartość „false” jako liczba całkowita to „0”   */
    std::cout << "false = " << falseNum << std::endl;

    /* Rzutowanie przykładowych liczb całkowitych
       na wartości logiczne typu „bool”             */
    auto a { static_cast<bool>(7) };
    auto b { static_cast<bool>(0) };
    auto c { static_cast<bool>(-16) };

    std::cout << "static_cast<bool>(7)   =  " << a << std::endl;
    std::cout << "static_cast<bool>(0)   =  " << b << std::endl;
    std::cout << "static_cast<bool>(-16) =  " << c << std::endl;

    return EXIT_SUCCESS;
}
```

```
Konsola debugowania programu Microsoft Visual Studio
true = 1
false = 0
static_cast<bool>(7)   =  1
static_cast<bool>(0)   =  0
static_cast<bool>(-16) =  1
```

Rysunek 10.3. Przykład dla static_cast w języku C++

10.2. CONST_CAST

Za pomocą rzutowania `const_cast` można usunąć modyfikator `const` z elementu, aby móc do niego zapisać (rysunek 10.4). Nie zawsze jest to bezpieczne. Należy bardzo uważać, aby za pomocą `const_cast` nie usuwać modyfikatora `const` ze wskaźnika, ponieważ może to prowadzić do niezdefiniowanego zachowania [9]. Między innymi dlatego przykładowy kod z rysunku 10.4 korzysta z referencji (odwołania), a nie wskaźników.

Jeśli mnie pamięć nie myli, to określenie „referencja” (inaczej: odwołanie) pojawia się tutaj pierwszy raz w tej książce. Referencja w języku C++ pozwala m.in. utworzyć nową nazwę dla zmiennej w celu „przekazania dalej” jej wartości.

Tak jak zwykła zmienna typu `int` definiowana jest:

```
int num { 5 };
```

tak referencję (odwołanie) do tej zmiennej utworzymy za pomocą znaku ampersand (`&`), który umieszcza się po nazwie typu, np.:

```
int& num_ref { num }; // odwołanie do zmiennej num
```

Teraz wykonując przypisanie w stylu:

```
num_ref = 8;
```

modyfikowana jest zmienna `num` (przyjmuje wartość 8).

Operacja dodawania w stylu:

```
num_ref = num_ref + 2;
```

również zmodyfikuje zmienną `num` (doda do niej wartość 2).

Przykład usuwania modyfikatora `const` z referencji za pomocą rzutowania `const_cast` przedstawiono na rysunku 10.4.

Kup książkę Poleć książkę

```
#include <iostream>

int main()
{
    /* Definicja zmiennej typu „int” o wartości
       początkowej siedem (7).                          */
    int value { 7 };

    /* Odwołanie (referencja, znak &) do zmiennej
       „value” z modyfikatorem „const”.                 */
    const int& numConst { value };

    /* Błąd: Nie możemy przypisać nowej wartości
             do stałej (const).                         */
    //numConst = 8;

    /* Usuwamy modyfikator „const” z odwołania
       (referencji) „numConst”, aby mieć możliwość
       zapisu (modyfikacji wartości). Następuje
       tutaj konwersja z „const int&” na „int&”.    */
    int& num = const_cast<int&>(numConst);

    /* Może się wydawać, że nadajemy nową wartość
       dla „num”, jednak są to tylko odwołania
       (referencje), a modyfikujemy wartość
       zmiennej „value” [!].                            */
    num = 128;

    /* Nazwy „num” i „numConst” są odwołaniami do
       zmiennej „value”, a nie jej kopią.
       Dlatego gdy przypiszemy wartość do „num”,
       to zmodyfikowana zostanie zmienna „value”. */
    std::cout << "num = "       << num << std::endl;
    std::cout << "numConst = " << numConst << std::endl;
    std::cout << "value = "     << value << std::endl;

    return EXIT_SUCCESS;
}
```

```
Konsola debugowania
num = 128
numConst = 128
value = 128
```

Rysunek 10.4. Przykład dla const_cast w języku C++

Kup ksi k Pole ksi k

10.3. DYNAMIC_CAST

Opisanie nawigacji po hierarchii klas (typów własnych/użytkownika) za pomocą konwersji `dynamic_cast` jest trudne bez wprowadzenia nowych terminów, które też trzeba choć trochę wyjaśnić.

Klasy w języku C++ nazywane są też typami własnymi, użytkownika czy definiowanymi przez programistę. Ten rodzaj danych można sobie wyobrazić jako „kompozyt", czyli typ składający się z wielu elementów. Takie abstrakcje służą przeważnie odwzorowaniu rzeczywistości (lub świata gry w grach komputerowych).

Na razie przedstawię temat czysto teoretycznie.

Mechanizm klas pozwala tworzyć różne typy obiektów. Możemy np. powołać do istnienia obiekt typu „Samochód", który będzie miał wewnątrz zmienną „Licznik kilometrów", funkcję „Uruchom silnik" i wiele innych elementów. W dalszej kolejności możliwe jest stworzenie klasy bazowej „Pojazd", a potem klasy „Motocykl", który będzie miał niektóre elementy wspólne z klasą „Samochód".

W odniesieniu do gier komputerowych możliwe jest stworzenie klasy „Potwór", która będzie miała „Punkty zdrowia", zmienną, do której możemy przypisać „Broń" (kolejna klasa), dodać pole typu logicznego (`bool`) decydujące, czy potwór może używać magii, a nawet można zrobić listę przedmiotów, które „wypadną" z potwora po jego pokonaniu.

Jeśli chcemy odwzorować rzeczywistość lub stworzyć własną, ogranicza nas tylko wyobraźnia i składnia języka C++. Im więcej elementów składni poznasz i zrozumiesz, jak działają, tym większe możliwości będziesz miał jako programista. Co innego napisać „od góry do dołu" program rozkaz po rozkazie, a co innego zaprojektować wszystko, korzystając z bogatej składni języka programowania. Nie oznacza to jednak, że jestem fanem przesadnej inżynierii (ang. *overengineering*). Sam

Kup ksi k Pole ksi k

tworzyłem kiedyś dla siebie różne programy w języku Asembler (MASM), które były rozkazami procesora „od góry do dołu" podzielonymi na funkcje i które działają niezawodnie do dzisiaj. [Trochę odchodzisz od tematu — przyp. Mr. At].

Wróćmy do rzutowania `dynamic_cast`, które zapewnia kontrolę w czasie działania programu i jest przydatne do przemieszczania się po hierarchii klas. Przykładowy kod z rysunku 10.5 zawiera klasę `program` oraz klasę `virus`, która dziedziczy po klasie `program` (inaczej: klasa `program` jest bazowa dla klasy `virus`). Typy te są puste, aby można było skupić się na poznaniu ogólnej idei rzutowania `dynamic_cast`. Na początku kodu (rysunek 10.5) tworzone są obiekty `virus1` oraz `notVirus`. Dalej jest utworzenie referencji (odwołania) do obiektu `virus1` i rzutowanie (konwersja) typu `virus` na `program` poprzez `dynamic_cast`. Wykonała się tutaj wspominana wcześniej nawigacja po hierarchii klas. Mając referencję do typu `virus` nastąpiło rzutowanie na typ `program`, który znajduje się głębiej w hierarchii klas, gdyż jest tutaj typem bazowym. Na końcu kodu z rysunku 10.5 sprawdzane są typy (operator `typeid`) wraz z ich wyświetleniem na standardowym wyjściu.

10.4. REINTERPRET_CAST

Inna konwersja nazwana to `reinterpret_cast`. W dużym skrócie można powiedzieć, że dzięki niej programista wie o tym, że typy nie są pokrewne, oraz świadomie wykonuje rzutowanie (konwersję między typami). Przykład dla tej konwersji przedstawiono wcześniej (rysunek 10.2).

Kod źródłowy z rysunku 10.2 dokonuje rzutowania ciągu bajtów na wskaźnik do funkcji. Programista był tutaj pewien, że bajty umieszczone w zmiennej są rozkazami procesora, i postanowił je przekonwertować

Kup ksi k Pole ksi k

```
#include <iostream>

/* Własny typ danych o nazwie „program” */
class program { };
/* Własny typ danych o nazwie „virus”,
   którego typem bazowym jest „program” */
class virus : public program { };

int main()
{
    /* Obiekt (zmienna) typu „virus” */
    virus virus1;
    /* Obiekt (zmienna) typu „program” */
    program notVirus;

    /* Odwołanie (referencja) do zmiennej „virus1” */
    virus& virusRef = virus1;

    /* Rzutowanie dynamiczne z typu „virus&” na typ „program&” */
    program& virusProgram = dynamic_cast<program&>(virusRef);

    /* Powinno zapewnić poprawne wyświetlanie polskich
       znaków na konsoli tekstowej systemu Windows */
    setlocale(LC_ALL, "");

    /* Operator „typeid” pobiera typ obiektu,
       a wywołanie funkcji „name()” zwraca
       nazwę tego typu w postaci napisu. */
    std::cout << "Wirus komputerowy jest wirusem: ";
    std::cout << typeid(virus1).name() << std::endl;
    std::cout << "Wirus jest też programem: ";
    std::cout << typeid(virusProgram).name() << std::endl;
    std::cout << "Na szczęście nie każdy program jest wirusem: ";
    std::cout << typeid(notVirus).name() << std::endl;

    return EXIT_SUCCESS;
}
```

```
Konsola debugowania programu Microsoft Visual Studio
Wirus komputerowy jest wirusem: class virus
Wirus jest też programem: class program
Na szczęście nie każdy program jest wirusem: class program
```

Rysunek 10.5. Przykład dla dynamic_cast w języku C++

na wskaźnik do funkcji, aby móc je wywołać i wykonać jako kod programu [Viral behaviour! o_O — przyp. Mr. At].

Kup książkę Poleć książkę

Nieco innym przykładem może być sytuacja, gdy mamy blok pamięci (zmienną) i wiemy, że są w nim dane np. typu `Image` (jakaś klasa reprezentująca rysunek). Jednak dane są zwykłym ciągiem bajtów, nie mają jasno określonego typu. Możliwe jest wtedy rzutowanie (konwersja `reinterpret_cast`), ale w zależności od źródła, z którego pochodzą dane, może być wymagane sprawdzenie poprawności formatu tych danych.

Kup ksi k

Pole ksi k

Kup ksi k Pole ksi k

PROGRAM PARTNERSKI

GRUPY 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

Wydajny i niezależny od platformy język wysokopoziomowy? To C++!

- **Poznaj konstrukcje składniowe języka C++**
- **Naucz się je stosować w praktyce**
- **Napisz swoje pierwsze programy**

Programowanie to w dużym uproszczeniu wydawanie komputerowi odpowiednich poleceń. Aby jednak móc to robić, trzeba opanować trudną sztukę komunikacji z maszyną, co w praktyce sprowadza się do posługiwania się zrozumiałym dla niej językiem. Obecnie niemal nikt nie musi się już uczyć języków niskopoziomowych, które są minimalistyczne i niezawodne, ale trudne do nauki i zastosowania w przypadku złożonych projektów. Z pomocą przychodzą tu języki wysokopoziomowe, a zwłaszcza łączący dużą wydajność z potężnymi możliwościami C++.

Jeśli chcesz go poznać lub odświeżyć swoją wiedzę, rusz w drogę z tym przewodnikiem! Odbędziesz dzięki niemu podróż po składni C++, zapoznasz się z jego instrukcjami i nauczysz się czytać kod. Dowiesz się, jak stosować podstawowe i złożone typy danych, odkryjesz sposoby użycia pętli, wkroczysz w świat funkcji i programowania obiektowego, a także opanujesz sztukę obsługi wyjątków. Poznasz też najważniejsze elementy standardu C++20, takie jak wymagania nazwane (ang. *concepts*), koprocedury (ang. *coroutines*), trójdrożny operator porównania <=> (tzw. statek kosmiczny), atrybuty [[nodiscard]] z komunikatem, [[likely]] i [[unlikely]], a także typ znakowy char8_t.

Prosto do celu i na praktycznych przykładach — z tą książką szybko rozgryziesz język, dzięki któremu kariera w IT stanie przed Tobą otworem. Nie zwlekaj, chwyć C++ za rogi!

- **Podstawy algorytmiki**
- **Składnia i konstrukcje języka C++**
- **Budowanie i uruchamianie projektów**
- **Typy podstawowe i złożone**
- **Rzutowanie i konwersja typów**
- **Instrukcje warunkowe i pętle**
- **Operatory i funkcje**
- **Tablice i wskaźniki**
- **Klasy i obiekty**
- **Obsługa wyjątków**
- **Zastosowanie szablonów**
- **Aplikacje WinRT**

Naucz się programować jak prawdziwy haker!