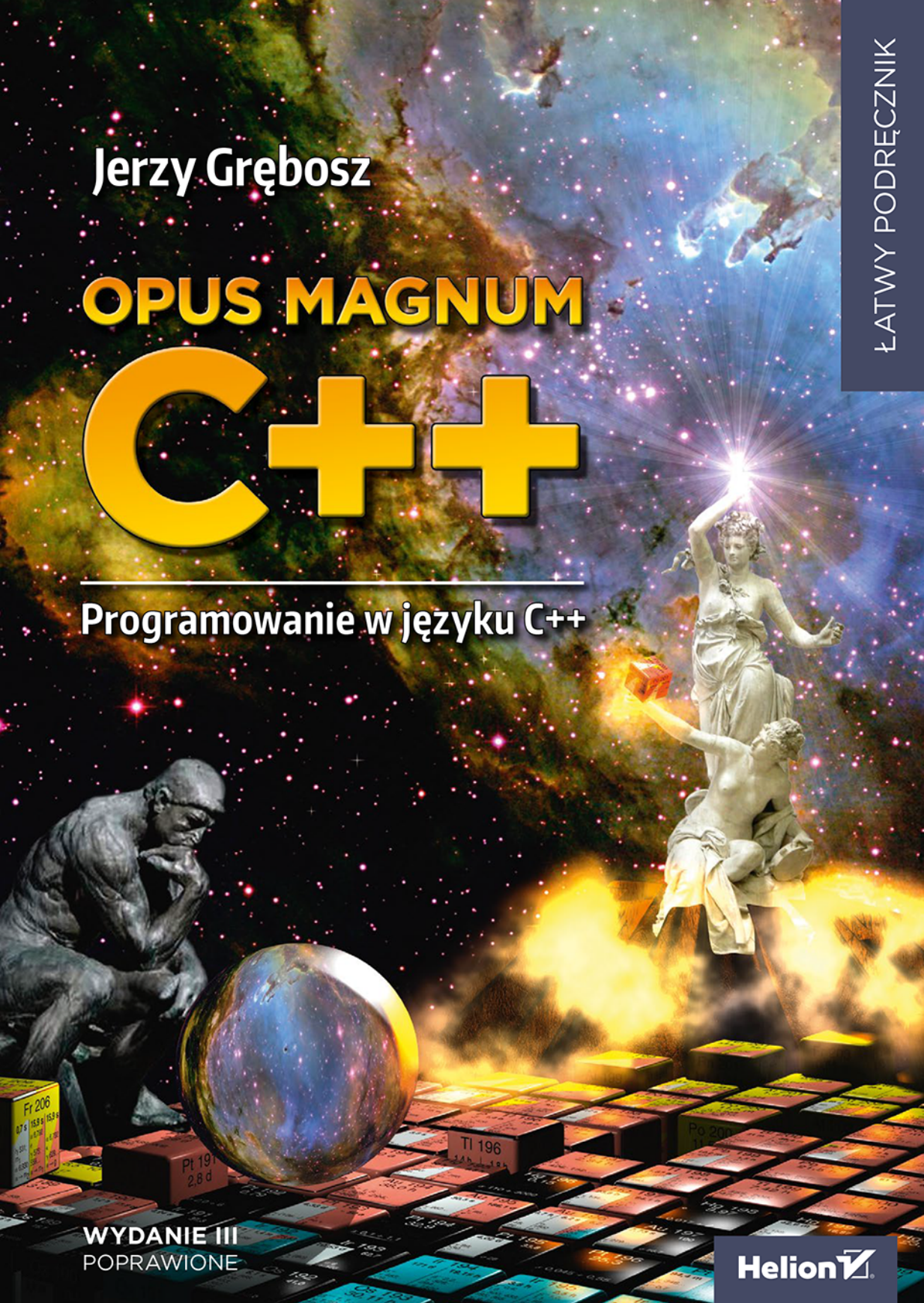
ŁATWY PODRĘCZNIK
Jerzy Grębosz
OPUS MAGNUM
C++
Programowanie w języku C++
WYDANIE III
POPRAWIONE
Helion

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 i opracowanie graficzne książki: Jerzy Grębosz

Zdjęcie Mgławicy Orzeł w grafice na okładce oraz zdjęcia łazika Curiosity i powierzchni Marsa – wykorzystane w tytułach rozdziałów – dzięki uprzejmości NASA.

Wydanie trzecie, poprawione

ISBN: 978-83-289-1131-4

Copyright © Jerzy Grębosz 2024
Printed in Poland

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 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/ocppk3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:
https://ftp.helion.pl/przyklady/ocppk3.zip

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści trzech

Tom 1

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Tom 2

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Tom 3

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

18 Deklaracje przyjaźni

Funkcja zaprzyjaźniona z klasą to funkcja, która – mimo że nie jest składnikiem tej klasy – ma dostęp do jej wszystkich (nawet prywatnych) składników.

18.1 Przyjaciele w życiu i w C++

Wyobraź sobie taką sytuację. W Twoim domu jest dużo roślin. Rośliny te są prywatnym składnikiem obiektu klasy `dom`. Pewnego dnia wyjeżdżasz na wakacje na Majorkę. Chcesz jednak, by kwiatki Ci nie „zdechły". Masz dwa wyjścia:

- Ewentualność pierwsza: sprawić, by kwiatki stały się publiczne, czyli wystawić je na klatkę schodową (kwiatki globalne). Każdy wtedy może wykonać na nich funkcję „podlewanie". Ryzykujesz jednak, że ktoś nieproszony wykona na nich funkcję „modyfikacja", czyli przerobi je na pokarm dla swojego królika czy węża boa. Wyjście – jeśli kochasz swoje kwiatki – nie jest dobre.
- Ewentualność druga: masz zaufanego przyjaciela. Dajesz mu klucze do swojego mieszkania i prosisz go, by podlewał kwiatki. Przyjaciel ma dostęp do wszystkich Twoich prywatnych składników (np. brylantowej kolii w komodzie), ale ponieważ mu ufasz, więc nie boisz się o nic. Przyjaciel przychodzi co drugi dzień i podlewa. Jak dotąd obrazek jest idylliczny.

Wścibscy sąsiedzi zauważają jednak, że ktoś obcy wchodzi do Twojego domu i dzwonią na policję, która pewnego popołudnia urządza zasadzkę – wykopuje przed drzwiami trzymetrowy dół i nakrywa go gałęziami. Przyjaciel wpada w zasadzkę. Policja zaciera ręce, że złapała złodzieja. Co prawda przyjaciel krzyczy coś z dna dołu o przyjaźni, ale nikt go nie słucha. I słusznie, prawdziwy złodziej krzyczałby to samo. Nadjeżdża specjalnym wozem nadinspektor. Ocenia sytuację i mówi:

„*Chwileczkę: oto mam w ręce definicję klasy pod tytułem Dom_Czytelnika i widzę, że na liście składników jest deklaracja, iż pana X uznaje się za przyjaciela Domu_Czytelnika. Ma on zatem prawo zrobić wszystko ze składnikami tej klasy. Proszę go więc wyciągnąć z dołu, bo jeszcze kwiatki zwiędną*".

Przełóżmy ten obrazek na język pojęć C++

Konstruując klasę, ustalamy, że pewne składniki będą prywatne. Mogą więc na nich pracować funkcje składowe tej klasy. Inne nie.

Kup książkę Poleć książkę

W pewnych sytuacjach jednak może być korzystne, by jakaś funkcja spoza zakresu tej klasy miała także dostęp do składników prywatnych. Robi się to bardzo prosto. Wewnątrz definicji klasy wystarczy umieścić deklarację tej funkcji poprzedzoną słowem friend[1]. Dzięki temu zwykła funkcja ma prawo dostępu do prywatnych składników klasy. To tak, jakby składniki te stały się dla niej publiczne.

> Ważne jest, że to nie funkcja ma twierdzić, iż jest zaprzyjaźniona. To klasa ma zadeklarować, że przyjaźni się z funkcją i tym samym nadaje jej prawo dostępu do swoich składników prywatnych. Zatem słowo friend pojawia się tylko wewnątrz definicji klasy.

Funkcja zaprzyjaźniona ma oczywiście także na mocy tej przyjaźni dostęp do składników protected.

Przykład deklaracji przyjaźni z funkcją:

Oto klasa Tpionek, w której jest deklaracja przyjaźni z funkcją raport:

```
class Tpionek
{
private:
    int kolor, pozycja;
    // dotychczasowe deklaracje
    // .....
    friend void raport(Tpionek );
};
```

Sama funkcja jest gdzieś w programie zdefiniowana następująco:

```
void raport (Tpionek p)
{
    cout << p.kolor << " pionek jest na pozycji " << p.pozycja << endl;
}
```

Funkcja raport wywoływana jest z argumentem typu Tpionek. Najważniejsze jest, że:

> Wewnątrz tej zaprzyjaźnionej funkcji raport możemy odwoływać się do prywatnych składników obiektu klasy Tpionek. To tak, jakby te składniki były publiczne. To wszystko.

Jeśli chcemy, by funkcja wypisała dane o jakimś Tpionku, to po prostu wysyłamy go jako argument funkcji.

```
Tpionek niebieski;      // definicja obiektu
...
raport(niebieski);      // wywołanie funkcji
```

Może Ci się nasunąć pytanie: „Skoro chcemy, by funkcja pracowała na danych składowych klasy, to dlaczego nie zrobić z niej po prostu funkcji składowej tej klasy?”.

Pochwalam ten pomysł. Tak właśnie powinno być w tym przypadku. Lepiej mieć funkcję jako składnik, bo łatwiej wtedy nad nią panować . Tak samo jak lepiej, by to domownik podlewał kwiatki, niż przychodził w tym celu ktoś obcy.

Funkcje zaprzyjaźnione mają pewne cechy, które je wyróżniają i czynią z nich bardzo dobre narzędzie

Najważniejsza cecha to:

1) ang. *friend* – przyjaciel [czytaj: *„frend”*]

Kup książkę Poleć książkę

Dzięki deklaracji przyjaźni możemy nadać dostęp do prywatnych składników naszej klasy nawet takiej funkcji, która nie mogłaby być funkcją składową naszej klasy z powodów zasadniczych.
Na przykład dlatego, że:
– jest już funkcją składową innej klasy
– albo musi być funkcją globalną (np. przeładowany operator <<).

Funkcja może być przyjacielem więcej niż jednej klasy. (Tak się dzieje, gdy więcej niż jedna klasa zadeklaruje z nią przyjaźń). Wtedy taka funkcja może mieć dostęp do prywatnych składników *kilku* klas.

18.2 Przykład: dwie klasy deklarują przyjaźń z tą samą funkcją

W programie mamy dwie klasy. Klasa Tpunkt opisuje współrzędne jakiegoś punktu. Klasa Tkwadrat opisuje współrzędne lewego dolnego rogu kwadratu i długość jego boku. W obu są deklaracje przyjaźni z funkcją globalną o nazwie sędzia.

```
#include <iostream>
#include <string>
using namespace std;
//--------------------------
class Tkwadrat;                          // deklaracja zapowiadająca      ❶
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
class Tpunkt
{
    int x, y;
    string nazwa;
public:
    Tpunkt(int a, int b, string opis);
    void ruch(int n, int m)
    {
        x += n;
        y += m;
    }
    // ...może coś jeszcze...

    friend int sedzia(Tpunkt & p, Tkwadrat & k);                          ❷
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
class Tkwadrat
{
    int x, y;
    int bok;
    string nazwa;
public:
    Tkwadrat(int a, int b, int dd, string opis);
    // ...może coś jeszcze...

    friend int sedzia (Tpunkt & p, Tkwadrat & k);                         ❸
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
Tpunkt::Tpunkt(int a, int b, string opis)                  // konstruktor
```


Kup książkę Poleć książkę

```
{
    x = a;
    y = b;
    nazwa = opis;
}
//*************************************************************************************************************
Tkwadrat::Tkwadrat(int a, int b, int dd, string opis)          // konstruktor
{   x = a;
    y = b;
    bok = dd;
    nazwa = opis;
}
//*************************************************************************************************************
// Z tą funkcją przyjaźnią się obie klasy.
int sedzia (Tpunkt & pt, Tkwadrat & kw)                                    ❹
{
    if(   (pt.x >= kw.x) && (pt.x <= (kw.x + kw.bok) )
          &&
          (pt.y >= kw.y) && (pt.y <= (kw.y + kw.bok) )
     )
    {
          cout << pt.nazwa << " lezy na tle " << kw.nazwa << endl;
          return 1;
    }else {
          cout << "AUT! " << pt.nazwa << " jest na zewnatrz " << kw.nazwa << endl;
          return 0;
    }
}
//*************************************************************************************************************
int main()
{
    Tkwadrat      bo(10, 10, 40, "boiska");
    Tpunkt        pi(20, 20, "pilka");

    sedzia(pi, bo );                                                       ❺
    cout << "kopiemy pilke!\n";
    while(sedzia(pi, bo))
    {
          pi.ruch(20,20);
    }
}
```

Po wykonaniu programu na ekranie zobaczymy:

```
pilka lezy na tle boiska
kopiemy pilke!
pilka lezy na tle boiska
pilka lezy na tle boiska
AUT! pilka jest na zewnatrz boiska
```

Przyjrzyjmy się ciekawszym miejscom programu

❷ Wewnątrz definicji klasy Tpunkt widzimy deklarację przyjaźni. Klasa Tpunkt stwierdza tutaj, że ma zaufanie do funkcji sędzia.

Bardzo ważna uwaga:

Kup książkę Poleć książkę

> Zauważ, że na liście argumentów tej funkcji jest nazwa klasy Tkwadrat. Do tej pory klasa ta jeszcze nie została zdefiniowana. Pamiętamy jednak, że w C++ każda nazwa, zanim zostanie użyta po raz pierwszy, musi zostać zadeklarowana.

Jak ten problem rozwiązać?

❶ Oto rozwiązanie. Jest to tak zwana **deklaracja zapowiadająca** (zwiastująca). Mówi ona: „Jakby co, to nazwa Tkwadrat jest nazwą klasy". To wszystko. Nie ma tu nic więcej na temat wewnętrznej struktury klasy Tkwadrat, ale to nie szkodzi, bo w momencie deklaracji przyjaźni te detale nie są jeszcze kompilatorowi potrzebne.

❸ To deklaracja przyjaźni w drugiej klasie. Podobnie: klasa Tkwadrat stwierdza tutaj, że ma zaufanie do funkcji sędzia.

❹ Oto definicja funkcji sędzia. Jest zdefiniowana jak najzwyklejsza funkcja.

> Różnica polega tylko na tym, że funkcja ta pracuje sobie na prywatnych składnikach obiektów obu klas – tak jakby były one publiczne.

Czym zajmuje się funkcja sedzia? Łatwo się zorientować, że po prostu sprawdza, czy obiekt klasy Tpunkt leży na tle obiektu klasy Tkwadrat („czy piłka leży na boisku"). Robi to przez porównanie współrzędnych punktu z obszarem zajmowanym przez obiekt klasy Tkwadrat.

❺ W funkcji main korzystamy z tej funkcji. Zdefiniowaliśmy dwa obiekty i wywołujemy funkcję sędzia. Zauważ, że obiekty te wysyłamy do funkcji jako zwykłe argumenty – nie ma tu żadnego zapisu w stylu

obiekt.funkcja( )

bowiem funkcja sędzia nie jest funkcją składową żadnej klasy.

18.3 W przyjaźni trzeba pamiętać o kilku sprawach

Funkcja jest zaprzyjaźniona z klasą, a nie tylko z jakimś konkretnym obiektem danej klasy. To znaczy, że funkcja zaprzyjaźniona otrzymuje prawa przyjaciela w stosunku do wszystkich obiektów tej klasy.

✧ Deklaracja przyjaźni tylko deklaruje przyjaźń – i nic więcej. Konkretnie: nazwa funkcji zaprzyjaźnionej nie staje się przez to nazwą z zakresu tej klasy.

> *Po prostu w deklaracji przyjaŸni tylko rozmawiamy z kompilatorem, t³umacz¹ c mu, ¿e taka funkcja ma dostêp...*

✧ Funkcja zaprzyjaźniona nie jest składnikiem klasy, dlatego nie ma wskaźnika this do obiektów klasy, która obdarza ją przyjaźnią.

Dla nas oznacza to, że jeśli chcemy w ciele tej funkcji odnieść się do składnika jakiegoś obiektu klasy, która uznaje nas za przyjaciela, musimy powiedzieć:

- jak ten obiekt się nazywa (wtedy posługujemy się składnią *obiekt.składnik*)
- lub pokazać na niego wskaźnikiem (wtedy posługujemy się składnią *wskaźnik–>składnik*).

Kup książkę Poleć książkę

Powtarzam więc wniosek:

Funkcja zaprzyjaźniona to zwykła funkcja, której wyjątkowo nie obowiązują słowa *private* i *protected* w klasach uznających ją za przyjaciela.

Zwykle wewnątrz klasy funkcja zaprzyjaźniona jest tylko deklarowana. Jest to jedynie deklaracja przyjaźni. Nie ma znaczenia, w którym miejscu klasy (public, protected, private) taka deklaracja nastąpiła. Słowa public, protected i private nie mają na to wpływu. Przyjacielem albo się jest, albo nie jest.

Może się tak zdarzyć, że kompilator (pracując nad jakimś plikiem) zobaczy deklarację pewnej funkcji po raz pierwszy dopiero w miejscu deklaracji przyjaźni. Nie jest to błąd, ale uwaga: w tym miejscu kompilator uzna, że chodzi o jakąś funkcję globalną, dostępną ogólnie – także z innych plików tego programu.

Jeśli jednak zostanie przez nas oszukany, czyli gdzieś dalej zobaczy, że definiujemy tę funkcję jako funkcję static (a więc widzialną tylko dla jednego konkretnego pliku), zasygnalizuje błąd.

Nie byłoby problemu, gdybyśmy wcześniej zamieścili deklarację tej funkcji jako static, bo wtedy przy deklaracji przyjaźni kompilator już wiedziałby, z czym ma do czynienia, i nie musiałby niczego zakładać w ciemno, a potem zmieniać zdania.

Jak to zrobić w naszym niedawnym programie?

U nas w punkcie ❷ w deklaracji przyjaźni po raz pierwszy pojawia się nazwa funkcji sędzia, nieznana jeszcze kompilatorowi. Skoro nie było jeszcze deklaracji tej funkcji, kompilator zakłada, że chodzi o jakąś funkcję sędzia z zakresu globalnego.

Lepszą praktyką jest jednak deklarowanie wszystkich funkcji, które potem mają wystąpić w deklaracji przyjaźni.

U nas polegałoby to na postawieniu deklaracji funkcji sędzia pod linijką ❶. Niestety w funkcji sędzia jeden z argumentów jest typu Tpunkt, a także ten typ jest tu jeszcze kompilatorowi nieznany. Rozwiązanie jest proste: i on powinien mieć deklarację zapowiadającą. Łącznie więc: w programie, w miejscu ❶, dobrze by było mieć takie deklaracje:

```
class Tkwadrat;                          // deklaracja zapowiadająca      ❶
class Tpunkt;                            // deklaracja zapowiadająca
int sedzia (Tpunkt & pt, Tkwadrat & kw); // deklaracja funkcji
```

Wiele kompilatorów nie wymaga surowo wcześniejszych deklaracji funkcji, które mają zostać potem obwołane przyjaciółmi, i wybaczy nam ich brak. No ale według standardu C++ te deklaracje powinny jednak być.

Przyjaciel-rezydent, czyli: funkcja zaprzyjaźniona „goszcząca” w klasie

Możemy tak zrobić, że wewnątrz klasy jest nie tylko deklaracja funkcji zaprzyjaźnionej, ale wręcz jej definicja (czyli całe ciało funkcji). Mimo że jest ona umieszczona „w środku” ciała klasy, funkcja jest nadal tylko przyjacielem, a nie składnikiem.

Taka definicja funkcji zaprzyjaźnionej ma następujące konsekwencje:

- funkcja zaprzyjaźniona jest typu inline,
- funkcja leży w zakresie ***leksykalnym*** deklaracji tej klasy; oznacza to, że można w definicji tej zaprzyjaźnionej funkcji:

Kup książkę Poleć książkę

a) skorzystać z właśnie obowiązujących (wewnątrz deklaracji tej klasy) instrukcji using lub typedef,

b) skorzystać ze zdefiniowanych w tej klasie typów wyliczeniowych enum.

Nie zawsze można takim chwytem się posłużyć. Aby to było możliwe:

- Klasa, w której ma się to zdarzyć, nie może być tak zwaną *klasą lokalną*.

  Dotychczasowe nasze klasy by³y nielokalne (czyli zwyk³e), tutaj wiêc nie ma przeszkód. (O klasach lokalnych porozmawiamy na str. 736).

- Tak definiowana funkcja zaprzyjaźniona nie może mieć nazwy z kwalifikatorem zakresu twierdzącej, że jest ona jakoby funkcją składową jakiejś innej klasy, podczas gdy naprawdę takiej funkcji składowej w tamtej klasie nie ma.

  To by by³y wrêcz podchody! WyobraŸsobie: ktoœzdefiniowa³ klasê i jest z niej dumny, a my – za pomoc¹ deklaracji przyjaŸni w naszej klasie – cichaczem dodefiniowujemy tej jego klasie piêtnaœcie dodatkowych funkcji sk³adowych!

Jeśli skorzystamy z tej możliwości, aby w klasie K, przy okazji deklaracji przyjaźni ze zwykłą funkcją f, od razu umieścić także definicję tej funkcji zaprzyjaźnionej f, to pamiętajmy, że wtedy ta funkcja nie jest widoczna na zewnątrz klasy K. Aby tak było, trzeba ją wcześniej deklarować:

```
void rezydent();                                                    ❶
////////////////////////////////////////////////////////////////////////////////
class K
{
    friend void rezydent()   { cout << „Jestem rezydent funkcja” << endl; }
};
////////////////////////////////////////////////////////////////////////////////
int main()
{
    rezydent();                                                     ❷
}
```

Zatem bez deklaracji ❶ kompilator w miejscu ❷ wykryje błąd, mówiąc, że nazwa rezydent nie jest mu znana. Wniosek: w każdej sytuacji najlepiej, żeby funkcja, którą klasa ma obdarzyć przyjaźnią, była już wcześniej zadeklarowana.

Przy przeładowaniu

W przypadku funkcji przeładowanych przyjacielem klasy K jest tylko ta wersja funkcji, która odpowiada liście argumentów widocznej w deklaracji przyjaźni w definicji danej klasy K.

```
class K
{
    //...
    friend void alarm(K obj, int k);
};

void alarm(float*, K obiekt);
void alarm(void);
void alarm(K obiekt, int i);                    //    to jest przyjaciel klasy K
```


Kup książkę Poleć książkę

18.4 Obdarzenie przyjaźnią funkcji składowej innej klasy

Funkcja zaprzyjaźniona może być zwykłą funkcją, a może być też funkcją składową zupełnie innej klasy.

Oto tak zmodyfikowany poprzedni przykład, że funkcja sędzia jest składnikiem klasy Tkwadrat, a klasa Tpunkt deklaruje z tą funkcją przyjaźń.

```
#include <iostream>
#include <string>
using namespace std;
//---------------------------
class Tpunkt;                    // deklaracja zapowiadająca                    ❶
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
class Tkwadrat
{
    int x, y;
    int bok;
    string nazwa;
public:
    Tkwadrat(int a, int b, int dd, string opis);
    // ...może coś jeszcze...

    int sedzia (Tpunkt & p);                                                     ❷
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
class Tpunkt
{
    int x, y;
    string nazwa;
public:
    Tpunkt(int a, int b, string opis);
    void ruch(int n, int m) {
        x += n;
        y += m;
    }
    // ...może coś jeszcze...

    friend int Tkwadrat::sedzia(Tpunkt & p);                                     ❸
};
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
Tpunkt::Tpunkt(int a, int b,string opis)                    // konstruktor
{
    x = a;
    y = b;
    nazwa = opis;
}
//**************************************************************************************************************************
Tkwadrat::Tkwadrat(int a, int b, int dd, string opis)       // konstruktor
{
    x = a;
    y = b;
```


Kup książkę Poleć książkę

```
    bok = dd;
    nazwa = opis;
}
//*****************************************************************************************************************
int Tkwadrat::sedzia (Tpunkt & pt)                                                          ❹
{
    if(   (pt.x >= x) && (pt.x <= (x + bok) )                                               ❺
          &&
          (pt.y >= y) && (pt.y <= (y + bok) )
      )
    {
          cout << pt.nazwa << " lezy na tle " << nazwa << endl;
          return 1;
    }else {
          cout << "AUT! " << pt.nazwa << " jest na zewnatrz " << nazwa << endl;
          return 0;
    }
}
//*****************************************************************************************************************
int main()
{
    Tkwadrat      bo(10,10, 40, "boiska");
    Tpunkt        pi( 20, 20, "pilka");
    bo.sedzia(pi);                                                                          ❻
}
```

Po wykonaniu programu na ekranie pojawi się:

```
pilka lezy na tle boiska
```

Komentarz

❷ To jest deklaracja zwykłej funkcji składowej w klasie Tkwadrat. Argumentem jest obiekt klasy Tpunkt, stąd też konieczna była deklaracja zapowiadająca tę klasę ❶.

❸ To deklaracja przyjaźni. Tutaj jednak ważna uwaga. Jeśli przyjacielem ma być *funkcja składowa z innej klasy*, to ta klasa musi być już w tym momencie znana kompilatorowi. Dlatego najpierw w programie umieszczona jest deklaracja klasy Tkwadrat (z funkcją sędzia), a potem dopiero deklaracja klasy Tpunkt i niniejsze ogłoszenie przyjaźni.

❹ Oto definicja funkcji sędzia. Na pewno już przy deklaracjach zauważyłeś, że zmieniła się lista argumentów. Teraz argumentem jest tylko obiekt klasy Tpunkt. A co z obiektem klasy Tkwadrat, funkcja go przecież także potrzebuje?! Zapominasz, że teraz funkcja jest funkcją składową klasy Tkwadrat, a więc jest wywoływana na rzecz obiektu klasy Tkwadrat. Zresztą spójrz poniżej.

❻ Tak właśnie w main wywołujemy funkcję sędzia. Obiekt klasy Tpunkt wysyłany jest jako argument, a obiekt klasy Tkwadrat – przez ukryty wskaźnik this.

❺ Z faktu, iż funkcja sędzia jest funkcją składową klasy Tkwadrat, wynika, że odnosząc się do danej składowej swojej klasy, można posługiwać się zapisem: *składnik*, a nie zapisem: *obiekt*.*składnik* – widać to wyraźnie w tej linijce.
W stosunku do składników obiektu klasy zaprzyjaźnionej Tpunkt stosujemy tu zapis pt.x, ale w stosunku do składników swojej klasy: x, bok. Przedtem w tym miejscu było konieczne kw.x, kw.bok.

Kup książkę Poleć książkę

To dlatego, że przecież gdy piszemy x, to jest tam naprawdę this->x.

„Coœkrêcisz!” – zawo³a³eœzapewne – *„parê stron wczeœniej wmawia³eœmi, ¿e funkcja zaprzyjaŸniona z klas¹ nie zawiera wskaŸnika this!”.*

Podtrzymuję to!

> Funkcja F zaprzyjaźniona z klasą K nie zawiera wskaźnika this do klasy K, która uznaje ją za przyjaciela, bo nie jest składnikiem tej klasy.

Jeśli jednak sama funkcja F jest zwykłą funkcją składową jakiejś innej klasy, to zawiera wskaźnik this do obiektu *swojej* klasy. Za jego pomocą pracuje przecież na swoich składnikach.

Posłużmy się analogią do podlewania kwiatków. Załóżmy, że to Ty jesteś osobą podlewającą kwiatki i Twoja znajoma Perfidia zadeklarowała z Tobą przyjaźń.

Gdy określasz swoje czynności, to mówisz: „Idę podlać kwiatki Perfidii”. Jak do tej pory rzeczywiście nie ma wskaźnika this. Mówisz przecież o składnikach tych klas Perfidia.kwiatki.

Teraz uwaga: okazuje się, Czytelniku, że dostałeś mieszkanie i nie mieszkasz już więcej pod mostem. Nie jesteś już funkcją globalną, tylko należysz do klasy pod nazwą „mój_dom”. Załóżmy, że Perfidia deklaruje Cię nadal jako swojego przyjaciela.

Mówisz: podlewam „kwiatki Perfidii” (podlewam Perfidia.kwiatki).

Możesz jednak powiedzieć też: „podlewam kwiatki”, myśląc o podlewaniu *swoich* kwiatków

kwiatki *czyli* this->kwiatki

gdzie this oznacza wskaźnik do obiektu „mój_dom”.

Jeśli projektujesz program i widzisz, że przydałoby się, żeby funkcja miała dostęp do składników prywatnych dwóch klas, to masz do wyboru jedno z rozwiązań:

obie klasy deklarują tę funkcję (globalną) jako zaprzyjaźnioną,

funkcja jest składnikiem jednej klasy, a druga klasa deklaruje ją jako funkcję zaprzyjaźnioną.

Który z wariantów wybrać, decydujesz, rozważając wspomniane zalety funkcji zaprzyjaźnionej z cechami funkcji składowej. O podejmowaniu takich wyborów porozmawiamy jeszcze w jednym z następnych rozdziałów. („Przeładowanie operatorów” – str. 956).

18.5 Klasy zaprzyjaźnione

Klasa K może deklarować przyjaźń z więcej niż jedną funkcją składową klasy M. Może nawet deklarować przyjaźń ze wszystkimi funkcjami klasy M. Jest to trochę tak, jakbyś wytrwale zadeklarował przyjaźń klasy K z każdą funkcją składową klasy M. Łatwo te deklaracje zrobić, ale wymaga to dużo pisania.

Zamiast tego możemy zadeklarować, że klasa K uznaje za przyjaciela *całą* klasę M.

```
class K
{
    friend class M;
    // ...
};
```


Kup książkę Poleć książkę

Od tej pory *wszystkie* funkcje składowe klasy M mają dostęp do prywatnych składników klasy K deklarującej tę przyjaźń. To już Cię nie dziwi – przyzwyczaiłeś się do tego przy okazji funkcji zaprzyjaźnionych. Jest tu jednak coś jeszcze, coś, czego przy funkcjach być nie mogło. Załóżmy, że mamy zwykłą klasę K, która deklaruje przyjaźń z klasą PRZYJACIEL.

Klasa PRZYJACIEL może używać składników prywatnych klasy K nie tylko w swych funkcjach składowych, ale **także przy inicjalizacji swych składników, nawet tych statycznych**. Jak pewnie jeszcze pamiętasz, ich definicje są umieszczane osobno, jakby na zewnątrz definicji klasy PRZYJACIEL.

> Nawet więc tam, jakby na zewnątrz ciała klasy PRZYJACIEL, przyjaciel może skorzystać z wartości prywatnego składnika klasy K.

✧ Jeśli klasa K ma w sobie jakieś definicje typów enum lub typedef czy using, to klasa PRZYJACIEL też może z nich skorzystać przy deklaracji swoich składników.

✧ Mówiliśmy o tym, że definicję *funkcji* zaprzyjaźnionej z klasą K można umieścić nawet w samym miejscu deklaracji przyjaźni, czyli wewnątrz klasy K (funkcja: przyjaciel – rezydent, str. 701).

> Jednak z klasą-przyjacielem tej sztuczki zrobić się nie da. Ta klasa-przyjaciel musi mieć swoją definicję gdzieś na zewnątrz.

Deklaracja przyjaźni jest oczywiście jednostronna

Wyraża ją klasa K wobec klasy PRZYJACIEL i już. Natomiast klasa PRZYJACIEL nie wyraża niczego szczególnego w stosunku do klasy K. Konkretnie – wcale jej nie upoważnia do grzebania w swoich składnikach prywatnych.

Dwie klasy mogą się przyjaźnić także z wzajemnością

Jedyną możliwością zadeklarowania takiej przyjaźni jest właśnie deklaracja przyjaźni z klasą jako całością sposobem, jaki pokazaliśmy.

> Nie ma możliwości zadeklarowania w jednej klasie, że przyjaźni się ona z funkcjami innej klasy, a w tej innej klasie – że przyjaźni się z wybranymi funkcjami klasy pierwszej.

To z powodu, o którym już wspomnieliśmy: jeśli deklarujemy przyjaźń z funkcją, która jest funkcją składową innej klasy, to kompilator życzy sobie już znać deklarację tej klasy (na przykład, żeby sprawdzić, czy taka funkcja rzeczywiście tam jest).

Żebyśmy się nie wiem jak gimnastykowali, to zawsze definicja jednej klasy będzie znana wcześniej od drugiej, bo tak przecież piszemy tekst programu. Siłą rzeczy klasa, która definiowana jest wcześniej, musiałaby zawierać w sobie deklaracje przyjaźni z funkcjami składowymi klasy drugiej, chwilowo jeszcze nieznanymi. (Sama deklaracja zapowiadająca klasę nie wystarcza kompilatorowi – musi on znać wnętrze klasy).

Nie ma problemu. Wyjście z tego błędnego koła załatwia nam deklaracja przyjaźni z całą klasą.

```
class Tdruga;                    //    deklaracja zapowiadająca

class Tpierwsza {
    friend class Tdruga;
        // ...reszta ciała klasy pierwszej
```


Kup książkę Poleć książkę

```
};

class Tdruga {
    friend class Tpierwsza;
        // ...reszta ciała klasy drugiej
};
```

Przyjaźń nie jest przechodnia

> Przyjaciel mojego przyjaciela nie jest moim przyjacielem.

Inaczej mówiąc: jeśli klasa A deklaruje przyjaźń z klasą B, natomiast klasa B deklaruje przyjaźń z klasą C, to wcale nie oznacza, że klasa A uznaje klasę C za swojego przyjaciela.

Gdyby o to chodziło, to należałoby w klasie A zamieścić deklarację takiej przyjaźni:

```
friend class C;
```

Przechodniość przyjaźni byłaby bardzo niebezpieczna. Zresztą w życiu także się nią nie posługujemy.

Przyjaźń nie jest dziedziczna

> Przyjaciel mojej prababki nie jest moim przyjacielem.

Wtajemniczeni wiedz¹ , ¿e klasa mo¿e mieæ „potomstwo” (klasy pochodne). PrzyjaŸñ nie jest dziedziczna – jeœli jakaœklasa chce mieæprzyjaciela, to powinna to powiedzieæwyraŸnie sama.

W deklaracji przyjaźni nie mogą pojawić się przydomki (specyfikatory)...

✧ ...określające sposób, w jaki przyjaciel został (przez kompilator) umieszczony w pamięci. Te niedozwolone przydomki to static, register, extern, thread_local i mutable.

W ¿yciu codziennym jest podobnie. Wypada powiedzieæ „PrzyjaŸniê siê z Tomaszem”, a nie wypada powiedzieæ „PrzyjaŸniê siê z Tomaszem, bo ma willê z basenem” albo „PrzyjaŸniê siê z Tomaszem, bo mieszka za granic¹ ”.

18.6 Konwencja umieszczania deklaracji przyjaźni w klasie

Spotyka się często konwencję definiowania klasy w taki sposób, że najpierw w definicji klasy wyszczególnia się wszystkie składniki publiczne (czyli widziane z zewnątrz klasy). Dopiero dalej występują składniki prywatne, czyli takie, o których zwykły użytkownik klasy nie musi już wiedzieć. (Używając pralki automatycznej, użytkownik nie musi wiedzieć o wszystkich jej elementach elektronicznych i mechanicznych).

Funkcje zaprzyjaźnione są tym, co powinno się od razu zauważyć, patrząc na definicję klasy, więc przyjęło się umieszczać je na samym początku, na samej górze definicji klasy.

Jak mówię – jest to tylko konwencja, która może czasem ułatwić „czytanie” definicji klas.

18.7 Kilka otrzeźwiających słów na zakończenie

Poznaliśmy tu nowe narzędzie pozwalające na dostęp do schowanych składników klasy. Nie daj się jednak ponieść. Przyjaźń jest przecież naruszeniem czegoś, z czego

Kup książkę Poleć książkę

jesteśmy bardzo dumni, czyli schowania części danych w klasie. Schowania po to, żebyśmy w przypadku gdy program „chodzi źle”, nie martwili się: „Któż to zmienił mi wartość tego składnika bez mojej wiedzy...”. Im mniej przyjaciół, tym łatwiej panować nad działaniem danej klasy.

Zatem szału nie ma, przesadzanie z przyjaźnią jest złą praktyką.

Naprawdę więc deklaracje przyjaźni będziemy stosowali głównie w sytuacjach:

- gdy będziemy chcieli, aby obiekt cout mógł wypisywać na ekranie treść składników obiektu naszej klasy (obdarzymy wtedy przyjaźnią klasę ostream),
- gdy będziemy chcieli, żeby na obiektach naszej klasy mogły sprawnie pracować globalne funkcje tzw. operatorowe (poznamy je na str. 956).

18.8 Ćwiczenia

I Przyjaźń polega na tym, że dana klasa K udziela zezwolenia innej funkcji/klasie:
a) na dostęp do pracy z obiektami klasy K,
b) na modyfikację jej składników w klasie K,
c) na dostęp do składników niepublicznych w obiektach klasy K,
d) na dostęp do składników niepublicznych w wyznaczonych obiektach klasy K.

II Klasa K oznajmia przyjaźń z funkcją/klasą P. Słowo friend umieszczone jest:
a) w deklaracji funkcji/klasy P uznanej za przyjaciela,
b) w klasie K, która ogłasza przyjaźń z funkcją/klasą P,
c) w klasie/funkcji P, w instrukcjach, które odnoszą się do prywatnych składników klasy K.

III Przyjaźń może deklarować:
a) klasa, b) funkcja globalna, c) funkcja składowa, d) każde z wymienionych.

IV Co może być przyjacielem?
a) funkcja globalna, c) funkcja składowa,
b) funkcja statyczna, d) klasa.

V Jedna funkcja może być przyjacielem
a) tylko jednej klasy, b) nawet wielu klas.

VI Czy w deklaracji przyjaźni wyrażonej w klasie K wobec funkcji f, funkcja ta musi być wcześniej zadeklarowana?

VII Jeśli funkcja-przyjaciel klasy K odnosi się do składników klasy K, to skąd wiadomo, którego obiektu klasy K to dotyczy?
a) działa to na wszystkie obiekty klasy K,
b) funkcja musi powiedzieć, o który konkretny obiekt jej chodzi.

VIII Przyjaciel klasy K otrzymuje prawo dostępu do składników niepublicznych klasy K
a) we wszystkich obiektach klasy K,
b) w wybranym obiekcie klasy K.

IX Jakie ma konsekwencje fakt, że funkcja-przyjaciel klasy K ma swoją definicję dołączoną do deklaracji przyjaźni w klasie K („funkcja-rezydent”)?

X W lokalnej klasie KL umieszczamy deklarację przyjaźni z funkcją f i równocześnie definicję tej funkcji f („funkcja-rezydent”). Czy funkcja f może zwracać rezultat będący obiektem typu KL?

Kup książkę Poleć książkę

XI Klasa K deklaruje przyjaźń z funkcją składową klasy P. Jaka deklaracja klasy P musi poprzedzać definicję tej klasy?

a) wystarczy deklaracja zapowiadająca P, b) konieczna jest definicja klasy P.

XII Wybierz wszystkie poprawne zakończenia następującego zdania: Klasa K deklaruje przyjaźń z funkcją składową klasy P; ta funkcja składowa:

a) jest wywoływana na rzecz obiektu klasy K, która obdarzyła ją przyjaźnią,
b) jest wywoływana na rzecz obiektu swojej klasy P,
c) może pracować na składnikach swego obiektu klasy P,
d) może pracować na obiekcie klasy K pod warunkiem, że wie na którym.

XIII Klasa K deklaruje przyjaźń z klasą P. Czy funkcja składowa klasy P otrzymuje wskaźnik this pozwalający jej pracować na składnikach klasy K?

XIV Co to znaczy, że klasa K deklaruje przyjaźń z klasą P?

XV Co to znaczy, że przyjaźń nie jest przechodnia?

XVI Czy przyjaźń jest wzajemna?

XVII Gdzie należy umieszczać deklarację przyjaźni w klasie K. W jej części public czy private?

XVIII Funkcja składowa klasy P ma przydomek static. W klasie K chcemy zamieścić deklarację przyjaźni z tą funkcją. Gdzie umieszczamy ten przydomek?

a) po słowie friend, a przed typem rezultatu,
b) na samym końcu deklaracji.

Kup książkę Poleć książkę

Kup książkę Poleć książkę


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
GRUPA
Helion

OPUS MAGNUM C++

Programowanie w języku C++

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 zorientowane obiektowo
- Szablony

Jeżeli chcesz uczyć się języka C++ w łatwy i przyjazny sposób, ta książka jest dla Ciebie!

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 jest on 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 publikacji o tym języku, która w prostym, obrazowym stylu wprowadza czytelnika w fascynujący świat programowania zorientowanego obiektowo. Zobacz, jak potężny jest **C++**.

Patroni:

BULLDOGJOB Think IT

programista

Helion