
Zadania z programowania w języku C/C++, cz. I



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY



Projekt „Programowa i strukturalna reforma systemu kształcenia na Wydziale Mat-Fiz-Inf”.
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Człowiek-najlepsza inwestycja

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
WYDZIAŁ MATEMATYKI, FIZYKI I INFORMATYKI
INSTYTUT INFORMATYKI

Zadania z programowania w języku C/C++, cz. I

Jacek Krzaczkowski



LUBLIN 2011

**Instytut Informatyki UMCS
Lublin 2011**

Jacek Krzaczkowski
ZADANIA Z PROGRAMOWANIA W JĘZYKU C/C++, CZ. I

Recenzent: Grzegorz Matecki

Opracowanie techniczne: Marcin Denkowski
Projekt okładki: Agnieszka Kuśmierska

Praca współfinansowana ze środków Unii Europejskiej w ramach
Europejskiego Funduszu Społecznego

Publikacja bezpłatna dostępna on-line na stronach
Instytutu Informatyki UMCS: informatyka.umcs.lublin.pl.

Wydawca

Uniwersytet Marii Curie-Skłodowskiej w Lublinie
Instytut Informatyki
pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin
Redaktor serii: prof. dr hab. Paweł Mikołajczak
www: informatyka.umcs.lublin.pl
email: dyrii@hektor.umcs.lublin.pl

Druk

ESUS Agencja Reklamowo-Wydawnicza Tomasz Przybylak
ul. Ratajczaka 26/8
61-815 Poznań
www: wwwesus.pl

ISBN: 978-83-62773-06-0

SPIS TREŚCI

PRZEDMOWA	vii
1 PODSTAWY JĘZYKÓW C I C++, OPERACJE STERUJĄCE	1
1.1. Wprowadzenie	2
1.2. Podstawy języków C i C++, operacje wejścia wyjścia.	2
1.3. Instrukcje warunkowe i wyboru	3
1.4. Pętle	4
2 FUNKCJE	7
2.1. Wprowadzenie	8
2.2. Zadania	8
3 WSKAŹNIKI I REFERENCJE	13
3.1. Wprowadzenie	14
3.2. Zadania	14
4 TABLICE JEDNOWYMIAROWE	17
4.1. Wprowadzenie	18
4.2. Zadania	18
5 NAPISY	23
5.1. Wprowadzenie	24
5.2. Zadania	25
6 TABLICE WIELOWYMIAROWE	31
6.1. Wprowadzenie	32
6.2. Zadania	32
7 ZŁOŻONE TYPY DANYCH, LISTY WSKAŹNIKOWE	37
7.1. Wprowadzenie	38
7.2. Złożone typy danych	38
7.3. Listy jednokierunkowe	41
8 OPERACJE NA PLIKACH	47

8.1.	Wprowadzenie	48
8.2.	Zadania	48
9	INSTRUKCJE PREPROCESORA, APLIKACJE WIELOPLIKOWE, MAKEFILE.	51
9.1.	Wprowadzenie	52
9.2.	Makra	52
9.3.	Aplikacje wieloplikowe, makefile	53
10	ROZWIĄZANIA I WSKAZÓWKI	57
10.1.	Rozwiązania do zadań z rozdziału 1.2	58
10.2.	Rozwiązania do zadań z rozdziału 1.3	62
10.3.	Rozwiązania do zadań z rozdziału 1.4	65
10.4.	Rozwiązania do zadań z rozdziału 2.2	70
10.5.	Rozwiązania do zadań z rozdziału 3.2	77
10.6.	Rozwiązania do zadań z rozdziału 4.2	79
10.7.	Rozwiązańa do zadań z rozdziału 5.2	86
10.8.	Rozwiązańa do zadań z rozdziału 6.2	97
10.9.	Rozwiązańa do zadań z rozdziału 7.2	105
10.10.	Rozwiązańa do zadań z rozdziału 7.3	113
10.11.	Rozwiązańa do zadań z rozdziału 8.2	129
10.12.	Rozwiązańa do zadań z rozdziału 9.2	138
10.13.	Rozwiązańa do zadań z rozdziału 9.3	139
	BIBLIOGRAFIA	151

PRZEDMOWA

Książka ta jest adresowana przede wszystkim do czytelników uczących się języka C lub strukturalnego C++ jako swojego pierwszego języka programowania. Ponadto może być użyteczna dla programistów C i programistów C++ pragnących szybko nauczyć się podstaw drugiego z omawianych języków. Ponieważ niniejszy zbiór jest przeznaczony przede wszystkim dla początkujących programistów, wiele spośród zawartych w nim zadań ma służyć nie tyle sprawdzeniu znajomości języka C lub C++, co wyrobieniu umiejętności algorytmicznego myślenia i programowania w ogóle. Z tego samego powodu w zbiorze tym jest niewiele zadań sprawdzających znajomość funkcji bibliotecznych. Nie znaczy to, że w skrypcie zabrakło zadań trudniejszych, wymagających znajomości bardziej zaawansowanych możliwości języków C i C++.

Niniejszy zbiór zadań może być używany zarówno w ramach kursu na uczelni, jak i przez osoby uczące się programować samodzielnie. Skrypt ten został napisany przy założeniu, że czytelnik korzysta jednocześnie z książki lub materiałów zawierających systematyczny opis składni języka C lub C++. Przykłady takich książek i materiałów, także takich, które są bezpłatnie dostępne w internecie, czytelnik znajdzie w bibliografii.

Pisząc niniejszy zbiór zadań autor wykorzystał doświadczenia nabycie w ciągu kilku lat prowadzenia zajęć z przedmiotów „Programowanie w języku C” i „Programowanie w języku C++” na kierunku informatyka na Uniwersytecie Marii Curie-Skłodowskiej w Lublinie. W pierwszych dziewięciu rozdziałach skryptu znajdują się podzielone tematycznie zadania. Kolejność rozdziałów odpowiada kolejności, w jakiej zdaniem autora należy uczyć się strukturalnego programowania w językach C i C++. Czytelnik uczący się języka C od podstaw może ominąć rozdział 3 i wrócić do niego w trakcie czytania rozdziału 5.

Zadania w poszczególnych rozdziałach są zazwyczaj ułożone od najprostszych do najtrudniejszych. Gwiazdką zostały oznaczone zadania trudniejsze lub wymagające szczególowej znajomości języka C lub C++. Czytelnik, który dopiero zaczyna swoją przygodę z programowaniem, może pominać w trakcie pierwszego czytania te zadania i wróć do nich później.

W ostatnim rozdziale czytelnik znajdzie rozwiązania wielu spośród zadań zawartych w niniejszym zbiorze zadań. Rozwiązania te są integralną częścią skryptu. Dołączone do rozwiązań komentarze mają na celu m.in. zwrócenie uwagi czytelnika na ciekawy sposób rozwiązania zadania, pojawiające się w trakcie rozwiązywania zadania problemy czy też błędy często popełniane przez niedoświadczonych programistów. W przypadkach gdy ma to wartość dydaktyczną przedstawiono kilka wariantów rozwiązań poszczególnych zadań.

W rozdziale z rozwiązaniami prezentowane są zarówno rozwiązania w języku C, jak i w C++. W wypadku gdy rozwiązania danego zadania w obu językach są takie same lub podobne, prezentowane jest tylko jedno rozwiązanie. Dla zadań, dla których wzorcowe rozwiązania w językach C i C++ istotnie się różnią, prezentowane są rozwiązania w obu językach. Często jednak nawet w takich sytuacjach rozwiązanie w języku C jest równocześnie poprawnym rozwiązaniem w C++.

Wszystkie rozwiązania zawarte w tym skrypcie były komplilowane przy pomocy GNU Compiler Collection w wersji 4.4.1 na komputerze z zainstalowanym systemem operacyjnym OpenSUSE 11.2. O ile nie podano inaczej, przykładowe programy napisane w C były komplilowane przy pomocy polecenia `gcc <nazwa programu>`. Przykładowe programy w C++ były komplilowane przy pomocy polecenia `g++ <nazwa programu>`.

Aby ułatwić posługiwanie się zbiorem zadań wprowadzono następujące oznaczenia:

- * trudne zadanie,
- r zadanie rozwiążane w ostatnim rozdziale,
- ! zadanie, którego rozwiązanie z różnych powodów jest szczególnie interesujące,
- C zadanie do rozwiązania wyłącznie w języku C,
- C++ zadanie do rozwiązania wyłącznie w języku C++,
- róż zadanie ilustrujące różnicę pomiędzy językiem C a językiem C++.

ROZDZIAŁ 1

PODSTAWY JĘZYKÓW C I C++, OPERACJE STERUJĄCE

1.1.	Wprowadzenie	2
1.2.	Podstawy języków C i C++, operacje wejścia wyjścia.	2
1.3.	Instrukcje warunkowe i wyboru	3
1.4.	Pętle	4

1.1. Wprowadzenie

W tym rozdziale czytelnik znajdzie zadania pozwalające przećwiczyć najbardziej podstawowe elementy języków C i C++. W podrozdziale 1.2 znajdują się zadania sprawdzające podstawową wiedzę na temat pisania programów w językach C i C++ oraz podstawy operacji wejścia/wyjścia w tych językach. Podrozdziały 1.3 i 1.4 zawierają zadania pozwalające przećwiczyć używanie instrukcji sterujących. Podrozdziały te są przeznaczone szczególnie dla tych, którzy uczą się swojego pierwszego imperatywnego języka programowania. W rozwiązaniach zadań celowo nie użyto instrukcji takich jak `goto`, `break` (za wyjątkiem wnętrza instrukcji `switch`) czy `continue`, które są uważane za niezgodne z zasadami programowania strukturalnego.

1.2. Podstawy języków C i C++, operacje wejścia wyjścia.

- 1.2.1 **(r)** Napisz program wypisujący na standardowym wyjściu napis „Hello World”.
- 1.2.2 Napisz program wypisujący w kolejnych wierszach standardowego wyjścia pojedyncze słowa następującego napisu „Bardzo dlugi napis”.
- 1.2.3 Napisz program wypisujący na standardowym wyjściu następujący napis: „Napis zawierajacy rozne dziwne znaczki // \ \\$ & %”.
- 1.2.4 **(r,!)** Napisz program, który wczytuje ze standardowego wejścia liczbę całkowitą i wypisuje ją na standardowym wyjściu.
- 1.2.5 **(r,!)** Napisz program, który wczytuje ze standardowego wejścia liczbę wymierną i wypisuje ją na standardowym wyjściu
- 1.2.6 Napisz program, który wczytuje ze standardowego wejścia trzy liczby całkowite, a następnie wypisuje je w oddzielnych liniach na standardowym wyjściu.
- 1.2.7 Napisz program, który wczytuje ze standardowego wejścia liczbę całkowitą i wypisuje na standardowym wyjściu liczbę o jeden większą.
- 1.2.8 **(r,!)** Napisz program, który wczytuje ze standardowego wejścia trzy liczby całkowite i wypisuje na standardowym wyjściu ich średnią arytmetyczną.
- 1.2.9 **(r,!rów)** Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę wymierną x i wypisuje na standardowym wyjściu \sqrt{x} .
- 1.2.10 Napisz program, który wczytuje ze standardowego wejścia liczbę wymierną x i wypisuje na standardowym wyjściu wartość bezwzględną z x .

- 1.2.11 (r) Napisz program, który wczytuje ze standardowego wejścia liczbę wymierną i wypisuje ją na standardowym wyjściu z dokładnością do dwóch miejsc po przecinku.
- 1.2.12 Napisz program, który wczytuje ze standardowego wejścia liczbę wymierną i wypisuje ją na standardowym wyjściu w notacji wykładniczej (czyli takiej, w której 0.2 to $2.0\text{e-}1$).

1.3. Instrukcje warunkowe i wyboru

- 1.3.1 (r) Napisz program, który wczytuje ze standardowego wejścia liczbę całkowitą n i wypisuje na standardowe wyjście wartość bezwzględną z n .
Do rozwiązania zadania nie używaj funkcji bibliotecznych za wyjątkiem operacji wejścia/wyjścia.
- 1.3.2 Napisz program, który wczytuje ze standardowego wejścia dwie liczby całkowite i wypisuje na standardowym wyjściu większą z nich (w przypadku gdy podane liczby są równe, program powinien wypisać każdą z nich).
- 1.3.3 Napisz program, który wczytuje ze standardowego wejścia trzy liczby całkowite i wypisuje na standardowym wyjściu największą z ich wartości (pamiętaj o przypadku gdy wszystkie podane liczby lub dwie z nich są równe).
- 1.3.4 Napisz program, który wczytuje ze standardowego wejścia dwie liczby całkowite i wypisuje tą o większej wartości bezwzględnej.
- 1.3.5 (r) Napisz program obliczający pole trójkąta na podstawie wymiarów podanych przez użytkownika. Użytkownik powinien mieć możliwość podania długości podstawy i wysokości lub wszystkich boków trójkąta.
- 1.3.6 Napisz program, który wczytuje ze standardowego wejścia współczynniki układu dwóch równań liniowych z dwoma niewiadomymi i wypisuje na standardowym wyjściu rozwiązanie układu równań. W przypadku nieskończonej liczby lub braku rozwiązań program powinien wypisać na standardowym wyjściu odpowiednią informację.

Podpowiedź: zaimplementuj algorytm rozwiązywania układów równań metodą wyznaczników (inaczej nazywaną wzorami Cramera).

- 1.3.7 Napisz program, który wczytuje ze standardowego wejścia współczynniki równania kwadratowego z jedną niewiadomą i wypisuje na standardowym wyjściu wszystkie rozwiązania rzeczywiste tego równania lub odpowiednią informację w przypadku braku rozwiązań.
- 1.3.8 (r) Napisz program, który w zależności od wyboru użytkownika wczytuje ze standardowego wejścia wymiary: kwadratu, prostokąta lub trójkąta.

kąta i wypisuje na standardowym wyjściu pole figury o wczytanych wymiarach.

- 1.3.9 Napisz program kalkulator, który wykonuje wybraną przez użytkownika operację arytmetyczną na dwóch wczytanych liczbach. Program powinien wczytywać dane ze standardowego wejścia i wypisywać wynik na standardowym wyjściu.

1.4. Pętle

- 1.4.1 (**r,!**) Napisz program wczytujący ze standardowego wejścia dwie dodatnie liczby całkowite n i m , i wypisujący w kolejnych wierszach na standardowym wyjściu wszystkie dodatnie wielokrotności n mniejsze od m .
- 1.4.2 Napisz program wczytujący ze standardowego wejścia dwie dodatnie liczby całkowite n i m , i wypisujący na standardowym wyjściu m pierwszych wielokrotności liczby n .
- 1.4.3 Napisz program wczytujący ze standardowego wejścia trzy dodatnie liczby całkowite n , m i k , i wypisujący w kolejnych wierszach wszystkie wielokrotności n większe od m i mniejsze od k .
- 1.4.4 (**r**) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu liczbę $n!$.
- 1.4.5 Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu sumę kwadratów liczb od 0 do n , czyli wartość $0^2 + 1^2 + 3^2 + \dots + n^2$.
- 1.4.6 Napisz program, który wczytuje ze standardowego wejścia liczbę całkowitą n ($n > 2$) i wypisuje na standardowym wyjściu iloczyn liczb parzystych z zakresu od 2 do n (czyli $2 * 4 * \dots * n$ dla n parzystych i $2 * 4 * \dots * (n - 1)$ w przeciwnym wypadku).
- 1.4.7 Napisz program, który wczytuje ze standardowego wejścia dwie liczby całkowite n i m (zakładamy, że $n < m$) i wypisuje na standardowym wyjściu liczbę $n * \dots * m$.
- 1.4.8 (*,r) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu element ciągu Fibonacciego o indeksie n .
- 1.4.9 (**r,!**) Napisz program, który wczytuje ze standardowego wejścia dodatnie liczby całkowite n i m , i wypisuje na standardowym wyjściu największy wspólny dzielnik tych liczb.
- 1.4.10 (**r,!**) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę n i wypisuje na standardowym wyjściu wartość $\lfloor \sqrt{n} \rfloor$

(wartość \sqrt{n} zaokrągloną w dół do najbliższej wartości całkowitoliczbowej). Program napisz bez użycia funkcji z biblioteki matematycznej.

- 1.4.11 Napisz program, który wczytuje ze standardowego wejścia liczby a , b , c , d i:
- wypisuje na standardowe wyjście najmniejszą nieujemną liczbę całkowitą x taką, że $|a| * x^2 + b * x + c > d$.
 - wypisuje na standardowe wyjście największą nieujemną liczbę całkowitą x taką, że $5*x^2+a*x+b < c$. Zakładamy, że taka nieujemna całkowita liczba x istnieje.
 - wypisuje na standardowe wyjście największą nieujemną liczbę całkowitą x taką, że $5*x^2+a*x+b \leq c$. Zakładamy, że taka nieujemna całkowita liczba x istnieje.
- 1.4.12 (*,r) Napisz program, który wczytuje ze standardowego wejścia dodatnią liczbę n i wypisuje na standardowym wyjściu sumę wszystkich liczb mniejszych od n , względnie pierwszych z n .
- 1.4.13 (*, r, !) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu wartość $0! + 1! + \dots + n!$.
- 1.4.14 (*) Napisz program, który wczytuje ze standardowego wejścia liczbę n i wypisuje na standardowym wyjściu wszystkie trójkę pitagorejskie (tj. trójkę liczb całkowitych a , b , c takich, że $a^2 + b^2 = c^2$), składające się z liczb mniejszych od n .

ROZDZIAŁ 2

FUNKCJE

2.1.	Wprowadzenie	8
2.2.	Zadania	8

2.1. Wprowadzenie

Funkcje są ważnym elementem większości języków imperatywnych. W niniejszym rozdziale znajdują się zadania pozwalające przećwiczyć różne aspekty pracy z funkcjami, od pisania prostych funkcji po przeciążanie funkcji i pisanie funkcji z domyślnymi wartościami argumentów (dwie ostatnie możliwości udostępnia nam tylko język C++). Czytelnik znajdzie w tym rozdziale także grupę zadań, w których należy napisać funkcję rekurencyjną. Pisanie funkcji rekurencyjnych wymaga szczególnej ostrożności, gdyż w ich przypadku szukanie błędów jest wyjątkowo trudne. Pomimo tego warto przećwiczyć pisanie funkcji rekurencyjnych, ponieważ w wielu przypadkach ich użycie pozwala znacznie uprościć kod programu.

2.2. Zadania

- 2.2.1 (r) Napisz program, który wczytuje ze standardowego wejścia liczbę całkowitą n i wypisuje na standardowe wyjście wartość bezwzględną z n . Do rozwiązania zadania nie używaj funkcji bibliotecznych za wyjątkiem operacji wejścia/wyjścia. W programie użyj samodzielnie zaimplementowanej funkcji liczącej wartość bezwzględną.
- 2.2.2 (r) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu liczbę $n!$. W programie użyj samodzielnie zaimplementowanej funkcji liczącej wartość silni.
- 2.2.3 Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n ($n > 2$) i wypisuje na standardowym wyjściu największą liczbę k taką, że k dzieli n i $k < n$. Algorytm wyszukiwania liczby k spełniającej powyższe warunki umieść w oddzielnej funkcji.
- 2.2.4 Napisz funkcję, która dostaje jako argument nieujemną liczbę całkowitą n i zwraca jako wartość liczbę 2^n .
- 2.2.5 Napisz funkcję, która dostaje jako argument liczbę całkowitą n i zwraca jako wartość liczbę 2^n .
- 2.2.6 Napisz funkcję, która dostaje jako argumenty nieujemne liczby całkowite n i m , z których co najmniej jedna jest różna od zera i zwraca jako wartość n^m .
- 2.2.7 Napisz funkcję, która dostaje jako argumenty liczby całkowite n i m , z których co najmniej jedna jest różna od zera i zwraca jako wartość n^m .
- 2.2.8 Napisz funkcję, która dostaje jako argumenty liczbę dodatnią n i zwraca jako wartość $\lfloor \sqrt{n} \rfloor$. Rozwiąż zadanie nie wykorzystując funkcji bibliotecznych.

- 2.2.9 (*) Napisz funkcję, która dostaje jako argumenty liczbę całkowitą m ($m > 1$) oraz nieujemną liczbę n i zwraca jako wartość $\lfloor \sqrt[m]{n} \rfloor$. Rozwiąż zadanie nie wykorzystując funkcji bibliotecznych.
- 2.2.10 (r) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu sumę liczb mniejszych od n i zarazem względnie pierwszych z n . Algorytm wyliczania sumy podziel na dwie funkcje.
- 2.2.11 Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu następującą sumę $\lfloor \sqrt{0} \rfloor + \lfloor \sqrt{1} \rfloor + \dots + \lfloor \sqrt{n} \rfloor$. Algorytm wyliczania sumy podziel na dwie funkcje.
- 2.2.12 Napisz program, który wczytuje ze standardowego wejścia nieujemne liczby całkowitą n i m ($m > 1$), i wypisuje na standardowym wyjściu następującą sumę $\lfloor \sqrt[m]{0} \rfloor + \lfloor \sqrt[m]{1} \rfloor + \dots + \lfloor \sqrt[m]{n} \rfloor$. Algorytm wyliczania sumy podziel na dwie funkcje.
- 2.2.13 (*,r) Napisz funkcję, która dostaje jako argument dodatnią liczbę całkowitą n i wypisuje na standardowym wyjściu wszystkie możliwe rozkłady liczby n na sumy dwóch kwadratów dodatnich liczb całkowitych. Rozważ dwa przypadki:
- (a) gdy „ $a^2 + b^2$ ” i „ $b^2 + a^2$ ” dla $a \neq b$ traktujemy jako dwa różne rozkłady,
 - (b) gdy „ $a^2 + b^2$ ” i „ $b^2 + a^2$ ” traktujemy jako ten sam rozkład i wypisujemy tylko jedne z nich.
- Jeżeli zajdzie taka potrzeba, możesz w rozwiązaniu używać funkcji pomocniczych.
- 2.2.14 (*) Napisz funkcję, która dostaje jako argument dodatnią liczbę całkowitą n i wypisuje na standardowym wyjściu wszystkie możliwe rozkłady liczby n na sumy kwadratów dodatnich liczb całkowitych. Rozważ dwa przypadki analogiczne do tych z zadania 2.2.13. Jeżeli zajdzie taka potrzeba możesz w rozwiązaniu używać funkcji pomocniczych.
- 2.2.15 (*) Napisz funkcję, która dostaje jako argumenty dodatnie liczby całkowite n i m , i wypisuje na standardowym wyjściu wszystkie możliwe rozkłady liczby n na sumę dwóch dodatnich liczb całkowitych podniesionych do potęgi m . Rozważ dwa przypadki analogiczne do tych z zadania 2.2.13. Jeżeli zajdzie taka potrzeba możesz w rozwiązaniu używać funkcji pomocniczych.
- 2.2.16 (*) Napisz funkcję, która dostaje jako argumenty dodatnie liczby całkowite n i m , i wypisuje na standardowym wyjściu wszystkie możliwe rozkłady liczby n na sumę dodatnich liczb całkowitych podniesionych do potęgi m . Rozważ dwa przypadki analogiczne do tych z zadania 2.2.13. Jeżeli zajdzie taka potrzeba możesz w rozwiązaniu używać funkcji pomocniczych.

- 2.2.17 (*,r,!) Napisz funkcję, która zlicza i wypisuje na standardowym wyjściu liczbę swoich wywołań.
- 2.2.18 (*) Napisz funkcję generującą liczby pseudolosowe. Pierwszą wartością funkcji powinna być dowolna liczba z przedziału $(0, 1)$. Kolejne wartości powinny być wyliczane ze wzoru $x_n = 1 - x_{n-1}^2$, gdzie x_n to aktualna, a x_{n-1} to poprzednia wartość funkcji.
- 2.2.19 (*) Napisz funkcję, która wczytuje ze standardowego wejścia liczbę całkowitą i zwraca ją jako swoją wartość. Dodatkowo funkcja powinna sumować wszystkie dotychczas wczytane wartości i przy każdym swoim wywołaniu wypisywać na standardowym wyjściu ich aktualną sumę .
- 2.2.20 (r,!) Napisz rekurencyjną funkcję, która dla otrzymanej w argumencie nieujemnej całkowitej liczby n zwraca jako wartość $n!$.
- 2.2.21 Napisz rekurencyjną funkcję zwracającą dla otrzymanej w argumencie nieujemnej liczby całkowitej n wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób:

$$a_0 = 1$$

$$a_n = 2 * a_{n-1} + 5 \text{ dla } n > 0.$$

- 2.2.22 Napisz rekurencyjną funkcję zwracającą dla otrzymanej w argumencie nieujemnej liczby całkowitej n wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób:

$$a_0 = a_1 = 1$$

$$a_n = a_{n-1} + 2 * a_{n-2} + 3 \text{ dla } n > 1$$

- 2.2.23 (r,!) Napisz rekurencyjną funkcję zwracającą dla otrzymanej w argumencie nieujemnej liczby całkowitej n wartość elementu ciągu Fibonacciego o indeksie n .
- 2.2.24 (*) Napisz rekurencyjną funkcję zwracającą dla otrzymanej w argumencie nieujemnej liczby całkowitej n wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób

$$a_0 = a_1 = 1$$

$$a_n = a_0 + a_1 + \dots + a_{n-1} \text{ dla } n > 1$$

- 2.2.25 (*) Napisz funkcję rekurencyjną, która dla otrzymanej w argumencie nieujemnej liczby całkowitej n zwraca wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób

$$a_0 = a_1 = 1$$

$$a_n = a_{n-1} + n \text{ dla } n \text{ parzystych}$$

$$a_n = a_{n-1} * n \text{ dla } n \text{ nieparzystych.}$$

- 2.2.26 (*,r,!) Napisz funkcję rekurencyjną, która dla otrzymanej w argumentie nieujemnej liczby całkowitej n zwraca wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób

$$a_0 = a_1 = a_2 = 1$$

oraz dla $k > 2$

$$a_{3 \cdot k} = a_{3 \cdot k - 1} + a_{3 \cdot k - 2}$$

$$a_{3 \cdot k + 1} = 5 * a_{3 \cdot k} + 4$$

$$a_{3 \cdot k + 2} = a_{3 \cdot k + 1}.$$

- 2.2.27 (r) Napisz funkcję rekurencyjną, która dla otrzymanej w argumentach pary nieujemnych liczb całkowitych n i m zwraca wartość $f(n, m)$ gdzie funkcja f jest zdefiniowana w następujący sposób:

$$f(n, 0) = n$$

$$f(0, m) = m$$

$$f(n, m) = f(n - 1, m) + f(n, m - 1) + f(n - 1, m - 1) \text{ dla } n, m > 0.$$

- 2.2.28 Napisz funkcję rekurencyjną, która dla otrzymanej w argumentach pary nieujemnych liczb całkowitych n i m zwraca wartość $f(n, m)$ gdzie funkcja f jest zdefiniowana w następujący sposób:

$$f(n, 0) = n$$

$$f(n, m) = f(m, n)$$

$$f(n, m) = n - m + f(n - 1, m) + f(n, m - 1) \text{ dla } n \geq m > 0.$$

- 2.2.29 (r,!) Napisz rekurencyjną funkcję, która dostaje jako argumenty dwie dodatnie liczby całkowite n i m , i zwraca jako wartość największy wspólny dzielnik tych liczb obliczony algorytmem Euklidesa.

- 2.2.30 (C++,r,!) Napisz funkcję, która dostaje jako argumenty nieujemne liczby całkowite n i m , z których co najmniej jedna jest różna od zera, i zwraca jako wartość n^m . Jeżeli drugi z argumentów nie zostanie podany, funkcja powinna zwrócić wartość n^2 .

- 2.2.31 (C++) Napisz funkcję, która dostaje jako argumenty liczbę całkowitą m ($m > 1$) oraz nieujemną liczbę n i zwraca jako wartość $\lfloor \sqrt[m]{n} \rfloor$. Rozwiąż to zadanie nie wykorzystując funkcji bibliotecznych. W przypadku podania tylko pierwszego argumentu funkcja powinna zwracać $\lfloor \sqrt{n} \rfloor$.

- 2.2.32 (C++) Napisz funkcję, która dostaje jako argumenty pięć liczb całkowitych typu `unsigned int` i zwraca jako wartość maksimum z podanych liczb. Funkcję napisz w taki sposób, żeby można było jej podać także mniejszą liczbę argumentów.
- 2.2.33 (C++) Napisz funkcję, która dostaje jako argumenty pięć liczb typu `unsigned int` i zwraca jako wartość sumę podanych liczb. Funkcję napisz w taki sposób, żeby liczyła sumę także dwóch, trzech i czterech argumentów.
- 2.2.34 (C++) Napisz funkcję, która dostaje jako argumenty pięć liczb typu `int` i zwraca jako wartość iloczyn podanych liczb. Funkcję napisz w taki sposób, żeby liczyła iloczyn także dwóch, trzech i czterech argumentów.
- 2.2.35 (C++) Napisz funkcję, która dla dwóch dodatnich całkowitoliczbowych argumentów m i n zwraca wartość `true` jeżeli n dzieli m oraz `false` w przeciwnym wypadku. W przypadku podania jednego argumentu funkcja powinna sprawdzać czy podana liczba jest parzysta.
- 2.2.36 (C++,r,!) Napisz rodzinę dwuargumentowych funkcji `pot`, z których każda jako argumenty otrzymuje liczbę n i nieujemną liczbę całkowitą m typu `unsigned int` (zakładamy, że co najmniej jeden z argumentów jest różny od zera) i zwraca jako wartość n^m . Przeciąż funkcję `pot` dla n o typach: `double`, `int`, `unsigned int`. Wynik zwrócony przez każdą z funkcji `pot` powinien być tego samego typu co n .
- 2.2.37 (C++) Rodzinę funkcji `pot` z zadania 2.2.36 rozszerz o funkcje, w których m jest typu `int`. Funkcje te powinny zwracać wartości typu `double`.
- 2.2.38 (C++) Napisz rodzinę funkcji `maksimum` zwracających maksimum z dwóch liczb otrzymanych w argumentach. Typem zwracanym powinien być bardziej pojemny z typów argumentów (tak jak to ma miejsce w przypadku operacji arytmetycznych). Przykładowo dla jednego argumentu typu `int` a drugiego typu `double` zwrócony powinien zostać wynik o typie `doble`.
- 2.2.39 (C++) Napisz rodzinę funkcji `maksimum` zwracających maksimum od dwóch do pięciu wartości otrzymanych w argumentach. Argumenty funkcji oraz wartość zwracana przez funkcję powinny być typu `int`.
- 2.2.40 (C++) Napisz rodzinę funkcji `srednia` zwracających średnią arytmetyczną z dwóch lub trzech wartości podanych przez użytkownika. Typem wyniku każdej z funkcji powinien być najbardziej pojemny z typów argumentów.

ROZDZIAŁ 3

WSKAŹNIKI I REFERENCJE

3.1.	Wprowadzenie	14
3.2.	Zadania	14

3.1. Wprowadzenie

Autor niniejszego zbioru jest zwolennikiem tego, aby w procesie nauczania języka C jako pierwszego języka programowania, najpierw mówić o tablicach jednowymiarowych, a dopiero potem mówić o wskaźnikach i ich powiązaniach z tablicami. Od lat w takiej właśnie kolejności był prezentowany materiał na kursie „Programowanie w języku C” na kierunku informatyka na UMCS. Takie ułożenie materiału pozwala studentom oswoić się z podstawami programowania zanim zaczną poznawać trudne zagadnienia związane ze wskaźnikami. Niniejszy zbiór zadań odszedł od takiego ułożenia materiału ze względu na jednoczesną prezentację rozwiązań w językach C i C++. O ile bowiem w języku C możemy używać jednowymiarowych tablic nie wiedząc nic o wskaźnikach ani dynamicznym zarządzaniu pamięcią, o tyle w języku C++ jest to możliwe tylko wtedy, gdy używamy tablic o z góry (t.j. w momencie komplikacji) znanych rozmiarach.

Ci spośród czytelników, którzy uczą się języka C i nie chcą jeszcze poznawać wskaźników mogą przejść od razu do rozdziału 4. Większość z prezentowanych tam zadań nie będzie wymagała od nich znajomości wskaźników.

3.2. Zadania

- 3.2.1 (r) Napisz funkcję otrzymującą jako argumenty wskaźniki do dwóch zmiennych typu `int`, która zwraca jako wartość mniejszą z liczb wskazywanych przez argumenty.
- 3.2.2 (r) Napisz funkcję otrzymującą jako argumenty wskaźniki do dwóch zmiennych typu `int`, która zwraca jako wartość wskaźnik na zmienną przechowującą mniejszą z liczb wskazywanych przez argumenty.
- 3.2.3 (r) Napisz funkcję otrzymującą jako argumenty wskaźniki do dwóch zmiennych typu `int`, która zamienia ze sobą wartości wskazywanych zmiennych.
- 3.2.4 Napisz funkcję otrzymującą jako argumenty wskaźniki do dwóch zmiennych typu `int`, która zamienia ze sobą wartości wskazywanych zmiennych tylko wtedy, gdy wskazywana przez drugi argument zmienna jest mniejsza od zmiennej wskazywanej przez pierwszy argument.
- 3.2.5 Napisz funkcję, której argumentami są dwa wskaźniki do stałych typu `int`, zaś zwracaną wartością jest suma wartości zmiennych wskazywanych przez argumenty.
- 3.2.6 Napisz funkcję, której argumentami są `n` typu `int` oraz `w` wskaźnik do `int`, która przepisuje wartość `n` do zmiennej wskazywanej przez `w`.

- 3.2.7 (**C++,r**) Napisz funkcję otrzymującą jako argumenty referencje do dwóch zmiennych typu **int**, która zamienia ze sobą wartości zmiennych, do których referencje dostaliśmy w argumentach.
- 3.2.8 (**C++**) Napisz funkcję otrzymującą dwa argumenty referencję **a** oraz wskaźnik **b** do zmiennych typu **int**, która zamienia ze sobą wartości zmiennych, do których wskaźnik i referencję dostała w argumentach.
- 3.2.9 (**r,róż**) Napisz bezargumentową funkcję, która rezerwuje pamięć dla pojedynczej zmiennej typu **int** i zwraca jako wartość wskaźnik do niej.
- 3.2.10 Napisz bezargumentową funkcję, która rezerwuje pamięć dla pojedynczej zmiennej typu **double** i zwraca jako wartość wskaźnik do niej.
- 3.2.11 (**r,róż**) Napisz funkcję, która dostaje jako argument dodatnią liczbę całkowitą **n**, rezerwuje w pamięci blok **n** zmiennych typu **int** i zwraca jako wartość wskaźnik do początku zarezerwowanego bloku pamięci.
- 3.2.12 Napisz funkcję, która dostaje jako argument dodatnią liczbę całkowitą **n**, rezerwuje w pamięci blok **n** zmiennych typu **double** i zwraca jako wartość wskaźnik do początku zarezerwowanego bloku pamięci.
- 3.2.13 (*,r,!) Napisz funkcję o dwóch argumentach:
- wskaźnik na funkcję o jednym argumencie typu **int** zwracającą wartość typu **double**,
 - wartość typu **int**,
- która zwraca wartość funkcji otrzymanej w pierwszym argumencie na liczbie całkowitej podanej w drugim argumencie.
- 3.2.14 (*) Napisz funkcję, która otrzymuje trzy argumenty:
- dwa wskaźniki na funkcje o jednym argumencie typu **int** zwracające wartość typu **int**,
 - wartość **n** typu **unsigned int**,
- i zwraca **true**, jeżeli otrzymane w argumentach funkcje są równe dla wartości od 0 do **n** i **false** w przeciwnym wypadku.
- 3.2.15 (**r,!**) Napisz funkcję, która dostaje dwa argumenty: wskaźnik na stałą typu **int** i wskaźnik na zmienną typu **int**, i przepisuje zawartość stałej wskazywanej przez pierwszy argument do zmiennej wskazywanej przez drugi argument.
- 3.2.16 (**r,!**) Napisz funkcję, która dostaje dwa argumenty: wskaźnik na stałą typu **int** i stały wskaźnik na zmienną typu **int**. I przepisuje zawartość stałej wskazywanej przez pierwszy argument do zmiennej wskazywanej przez drugi argument.

ROZDZIAŁ 4

TABLICE JEDNOWYMIAROWE

4.1.	Wprowadzenie	18
4.2.	Zadania	18

4.1. Wprowadzenie

Większość zadań prezentowanych w tym rozdziale nie wymaga od czytelnika uczącego się języka C znajomości wskaźników. Wystarczy umiejętność deklarowania tablic automatycznych. Wszystkie zadania wymagające operowania na wskaźnikach zostały oznaczone gwiazdką.

W języku C++ nie można deklarować tablic automatycznych o rozmiarze podanym przez zmienną. Jedynym sposobem w języku C++ tworzenia tablic o rozmiarze nieznanym w momencie komilacji jest ręczne tworzenie tablic dynamicznych (na przykład za pomocą operatora `new`).

4.2. Zadania

- 4.2.1 Napisz funkcję, która otrzymuje dwa argumenty: nieujemną liczbę całkowitą n oraz n -elementową tablicę `tab` elementów typu `int` i:
- (r) nadaje wartość zero wszystkim elementom tablicy `tab`,
 - (r) zapisuje do kolejnych elementów tablicy wartości równe ich indeksom (do komórki o indeksie i funkcja ma zapisywać wartość i),
 - podwaja wartość wszystkich elementów w tablicy `tab`,
 - do wszystkich komórek tablicy `tab` wstawia wartości bezwzględne ich pierwotnych wartości.
- 4.2.2 Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `int` i zwraca jako wartość:
- (r) średnią arytmetyczną elementów tablicy `tab`.
 - sumę elementów tablicy `tab`,
 - sumę kwadratów elementów tablicy `tab`.
- 4.2.3 (r) Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `const int` i zwraca jako wartość średnią arytmetyczną elementów tablicy `tab`.
- 4.2.4 (*) Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `unsigned int` i zwraca jako wartość średnią geometryczną elementów tablicy `tab`.
- 4.2.5 (*,r!,róż) Napisz funkcję, która otrzymuje jako argument liczbę całkowitą n ($n \geq 3$) i zwraca jako wartość największą liczbę pierwszą mniejszą od n (do wyznaczenia wyniku użyj algorytmu sita Eratostenesa).

- 4.2.6 Napisz funkcję, która otrzymuje trzy argumenty: dodatnią liczbę całkowitą n oraz dwie n -elementowe tablice `tab1`, `tab2` o elementach typu `int` i:
- (r) przepisuje zawartość tablicy `tab1` do tablicy `tab2`,
 - przepisuje zawartość tablicy `tab1` do tablicy `tab2` w odwrotnej kolejności (czyli element `tab1[0]` ma zostać zapisany do komórki tablicy `tab2` o indeksie $n - 1$).
- 4.2.7 Napisz funkcję, która otrzymuje cztery argumenty: dodatnią liczbę całkowitą n oraz trzy n -elementowe tablice `tab1`, `tab2` i `tab3` o elementach typu `int`, i:
- przypisuje elementom tablicy `tab3` sumę odpowiadających im elementów tablic `tab1` i `tab2` (do komórki tablicy `tab3` o indeksie i powinna trafić suma elementów `tab1[i]` i `tab2[i]`),
 - przypisuje elementom tablicy `tab3` większy spośród odpowiadających im elementów tablic `tab1` i `tab2` (do komórki tablicy `tab3` o indeksie i powinien trafić większy spośród elementów `tab1[i]` i `tab2[i]`),
 - przypisuje zawartość tablicy `tab1` do tablicy `tab2`, zawartość tablicy `tab2` do tablicy `tab3` oraz zawartość tablicy `tab3` do tablicy `tab1`.
- 4.2.8 Napisz funkcję, która otrzymuje cztery argumenty: dodatnią liczbę całkowitą n , n -elementowe tablice `tab1` i `tab2` oraz $2 \cdot n$ -elementową tablicę `tab3` o elementach typu `double`.
- Funkcja powinna przepisywać zawartość tablic `tab1` i `tab2` do tablicy `tab3` w taki sposób, że na początku tablicy `tab3` powinny się znaleźć elementy tablicy `tab1`, a po nich elementy tablicy `tab2`.
 - Funkcja powinna przepisywać zawartość tablic `tab1` i `tab2` do tablicy `tab3` w taki sposób, że w komórkach tablicy `tab3` o nieparzystych indeksach powinny się znaleźć elementy tablicy `tab1`, a w komórkach tablicy `tab3` o parzystych indeksach elementy tablicy `tab2`.
- 4.2.9 Napisz funkcję, która otrzymuje cztery argumenty: dodatnią liczbę całkowitą n oraz trzy n -elementowe tablice `tab1`, `tab2` i `tab3` o elementach typu `int` i zamienia zawartości komórek otrzymanych w argumentach tablic w następujący sposób:
- dla dowolnego i komórka `tab1[i]` powinna zawierać największą spośród pierwotnych wartości komórek `tab1[i]`, `tab2[i]` oraz `tab3[i]`,
 - dla dowolnego i komórka `tab2[i]` powinna zawierać drugą co do wielkości spośród pierwotnych wartości komórek `tab1[i]`, `tab2[i]` oraz `tab3[i]`,
 - dla dowolnego i komórka `tab3[i]` powinna zawierać najmniejszą

spośród pierwotnych wartości komórek `tab1[i]`, `tab2[i]` oraz `tab3[i]`.

- 4.2.10 Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `int` i:
- (`r,!`) zwraca największą wartość przechowywaną w tablicy `tab`,
 - zwraca najmniejszą wartość przechowywaną w tablicy `tab`,
 - (`r,!`) zwraca indeks elementu tablicy `tab` o największej wartości,
 - zwraca indeks elementu tablicy `tab` o najmniejszej wartości,
 - zwraca największą spośród wartości bezwzględnych elementów przechowywanych w tablicy `tab`,
 - zwraca indeks elementu tablicy `tab` o największej wartości bezwzględnej.
- 4.2.11 Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz dwie n -elementowe tablice `tab` o elementach typu `double` przechowujące n -wymiarowe wektory i zwraca jako wartość iloczyn skalarny wektorów otrzymanych w argumentach.
- 4.2.12 Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `int` i:
- (`r`) odwraca kolejność elementów tablicy `tab`.
 - (`r`) przesuwa o jeden w lewo wszystkie elementy tablicy (tak, żeby wartość elementu o indeksie $n - 1$ znalazła się w elemencie o indeksie $n - 2$, wartość elementu o indeksie $n - 2$ znalazła się w elemencie o indeksie $n - 3$, zaś wartość elementu o indeksie 0 w elemencie o indeksie $n - 1$),
 - (`r,!`) przesuwa o jeden w prawo wszystkie elementy tablicy (tak, żeby wartość elementu o indeksie 0 znalazła się w elemencie o indeksie 1, wartość elementu o indeksie 1 znalazła się w elemencie o indeksie 2, zaś wartość elementu o indeksie $n - 1$ w elemencie o indeksie 0),
 - (`*,r,!`) sortuje rosnąco elementy tablicy `tab` (porządkuje elementy przechowywane w tablicy w taki sposób aby ciąg `tab[0], tab[1], ..., tab[n-1]` był ciągiem niemalejącym),
 - sortuje malejąco elementy tablicy `tab`.
- 4.2.13 (`*,r,!`) Napisz funkcję, która otrzymuje jako argument dodatnią liczbę całkowitą n , a następnie tworzy dynamiczną n -elementową tablicę o elementach typu `int` i zwraca jako wartość wskaźnik do jej pierwszego elementu.
- 4.2.14 (*) Napisz funkcję, która otrzymuje jako argument dodatnią liczbę całkowitą n , a następnie tworzy dynamiczną n -elementową tablicę o elementach typu `double` i zwraca jako wartość wskaźnik do jej pierwszego elementu.
- 4.2.15 (`*,r,!`) Napisz funkcję, która dostaje jako argument wskaźnik do jed-

nowymiarowej dynamicznej tablicy o elementach typu `int` i zwalnia pamięć zajmowaną przez przekazaną w argumencie tablicę.

- 4.2.16 (*) Napisz funkcję, która dostaje jako argument wskaźnik do jednowymiarowej dynamicznej tablicy o elementach typu `double` i zwalnia pamięć zajmowaną przez przekazaną w argumencie tablicę.
- 4.2.17 (*) Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `double` a następnie tworzy kopię tablicy `tab` i zwraca jako wartość wskaźnik do nowo utworzonej kopii.
- 4.2.18 (*) Napisz funkcję, która otrzymuje trzy argumenty: dodatnią liczbę całkowitą n oraz dwie tablice n -elementowe o elementach typu `int` przechowujące współrzędne wektorów i zwraca jako wartość wskaźnik do pierwszego elementu nowo utworzonej tablicy przechowującej sumę wektorów otrzymanych w argumentach.
- 4.2.19 (*) Napisz funkcję, która dostaje w argumentach dodatnią liczbę całkowitą n oraz n -elementową tablicę liczb całkowitych `tab1` o elementach typu `int` i przepisuje do nowo utworzonej tablicy `tab2` elementy tablicy `tab1` o wartości różnej od zera. Rozmiar tablicy `tab2` powinien być równy liczbie niezerowych elementów tablicy `tab1`. Jako wartość funkcja powinna zwrócić wskaźnik na pierwszy element tablicy `tab2`.

ROZDZIAŁ 5

NAPISY

5.1.	Wprowadzenie	24
5.2.	Zadania	25

5.1. Wprowadzenie

Operacje na napisach, nazywanych też łańcuchami lub z języka angielskiego stringami, to jedna z najslabszych stron języka C. Operacje te wymagają od programisty szczególnej ostrożności. Z tego powodu, pomimo iż napisy są zwykłymi tablicami jednowymiarowymi o elementach typów znakowych,

to poświęcamy im oddzielny rozdział. W języku C++ można używać napisów takich jak w C, jednak w C++ istnieją specjalne klasy służące do ich przechowywania: **string** i **wstring**. Klasy te są znacznie wygodniejsze w użyciu od tablic znaków.

Na początku podrozdziału 5.2 są zadania, które należy rozwiązać bez używania funkcji bibliotecznych. Tą część uczący się języka C++ mogą pominąć, mogą ją też potraktować jako ćwiczenia z operacjami na tablicach jednowymiarowych. W drugiej części podrozdziału 5.2 są zadania, w których można używać funkcji bibliotecznych. Znajdują się tam także zadania, przeznaczone specjalnie dla uczących się C++, w których należy użyć typów **string** i **wstring**.

Napisy w języku C są przechowywane w jednowymiarowych tablicach o elementach typów znakowych. W języku C są dwa podstawowe typy znakowe **char** i **wchar_t** oraz modyfikacje typu **char**: **unsigned char** i **signed char**. Standard języka C nie określa długości zmiennych o typach **char** i **wchar_t**. Typ **char** jest określony jako typ wystarczający do przechowywania podstawowego zestawu znaków na danym komputerze (w praktyce **char** jest typem jednobajtowym), zaś typ **wchart_t** powinien wystarczyć do przechowywania pełnego zestawu znaków dostępnego na danym komputerze.

Na końcu poprawnego napisu w języku C (niezależnie od typu znaków z których się składa) znajduje się znak o kodzie 0. Służy on do zaznaczenia końca napisu. Tablica przechowująca n -znakowy napis musi mieć co najmniej $n + 1$ elementów (może mieć więcej), aby móc przechować kończący napis znak o kodzie 0. Konieczność dbania o to, żeby na końcu napisu zawsze był znak o kodzie 0, jest jedną z dwóch głównych niedogodności przy operowaniu na napisach w języku C.

Operacje na napisach o elementach typu **char** można znaleźć w bibliotece **string**, zaś niektóre operacje na znakach tego typu także w bibliotece **ctype**. Typ **wchar_t** oraz funkcje operujące na zmiennych tego typu, odpowiadające operacjom z innych bibliotek standardowych (w tym z biblioteki **string**) znajdują się w bibliotece **wchar**. Odpowiedniki funkcji z bibliotek **ctype** operujące na typie **wchar_t** znajdują się w bibliotece **wctype**. Opisy funkcji z powyższych bibliotek czytelnik znajdzie w literaturze. Funkcje z bibliotek standardowych wymagają dbania o to, żeby używane tablice znaków

były wystarczających rozmiarów. Jest to druga ze wspomnianych wcześniej dwóch najważniejszych niedogodności przy operowaniu na napisach w języku C.

W języku C++ do przechowywania napisów służą klasy `string` i `wstring`. Jedyną różnicą pomiędzy tymi klasami jest typ znaków, z których składają się napisy. W obiektach klasy `string` znaki są typu `char`, natomiast napisy przechowywane w obiektach klasy `wstring` składają się ze znaków typu `wchart_t`. Klasy `string` i `wstring` są zdefiniowane w bibliotece `string` języka C++.

Uwaga! Biblioteka `string` języka C++ i biblioteka o tej samej nazwie z języka C to dwie różne biblioteki. Biblioteka `string` języka C jest dostępna w języku C++ pod nazwą `cstring`.

Używanie klas `string` i `wstring` znacznie ułatwia operowanie na napisach. W szczególności operowanie na tych klasach uwalnia nas od wspomnianych wcześniej dwóch głównych mankamentów operacji na napisach w języku C, czyli konieczności dbania o obecność znaku o kodzie 0 na końcu napisu i o odpowiedni rozmiar używanych tablic. Napisy przechowywane w obiektach klas `string` i `wstring` mogą zawierać znaki o kodzie 0 jako normalne znaki w napisie, zaś metody zdefiniowane dla tych klas dbają za nas o przydział pamięci dla przechowywanych napisów. O różnicy w łatwości operowania na różnych rodzajach napisów można się przekonać analizując rozwiązania zadań z następnego podrozdziału. Opis klas `string` i `wstring` czytelnik znajdzie w literaturze.

5.2. Zadania

Następujące zadania rozwiąż nie używając funkcji bibliotecznych:

- 5.2.1 (r) Napisz funkcję `wyczysc`, która usuwa z tablicy przechowywany w niej napis (w sensie: umieszcza w niej poprawny napis o długości 0). Napisz dwie wersje funkcji `wyczysc` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.2 (r) Napisz funkcję `dlugosc`, która jako argument otrzymuje napis i zwraca jako wartość jego długość. Napisz dwie wersje funkcji `dlugosc` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.3 Napisz funkcję `porownaj`, która jako argumenty otrzymuje dwa napisy i zwraca 1 gdy napisy są równe i 0 w przeciwnym przypadku. Napisz dwie wersje funkcji `porownaj` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.4 (r,!) Napisz funkcję, która jako argumenty otrzymuje dwa napisy i zwraca wartość 1, gdy pierwszy napis jest wcześniejszy w kolejności leksykograficznej i 0 w przeciwnym przypadku.

Zakładamy, że oba napisy składają się ze znaków typu `char`, zawierają wyłącznie małe litery alfabetu łacińskiego, a system, na którym jest komplikowany i uruchamiany program, używa standardowego kodowania ASCII.

- 5.2.5 Napisz funkcję `przepisz`, która otrzymuje dwie tablice znaków i przepisuje串nypis znajdujący się w pierwszej tablicy do drugiej tablicy. Zakładamy, że w drugiej tablicy jest wystarczająco dużo miejsca. Napisz dwie wersje funkcji `przepisz` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.6 Napisz funkcję `kopiujn`, która dostaje w argumentach dwie tablice znaków `nap1`, `nap2` oraz liczbę n i przekopiowuje n pierwszych znaków z napisu przechowywanego w tablicy `nap1` do tablicy `nap2`. W przypadku gdy串nypis w tablicy `nap1` jest krótszy niż n znaków, funkcja powinna po prostu przepisać串nypis. Funkcja powinna zadbać o to, żeby na końcu napisu w tablicy `nap2` znalazła się znak o kodzie 0.
Zakładamy, że w docelowej tablicy jest wystarczająco dużo miejsca. Napisz dwie wersje funkcji: dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.7 (r) Napisz funkcję `sklej` otrzymującą jako argumenty trzy tablice znaków i zapisującą do trzeciej tablicy konkatenację napisów znajdujących się w dwóch pierwszych tablicach (czyli dla napisów "Ala m" i "a kota" znajdujących się w pierwszych dwóch argumentach do trzeciej tablicy powinien zostać zapisany串nypis "Ala ma kota"). Zakładamy, że w trzeciej tablicy jest wystarczająco dużo miejsca.
Napisz dwie wersje funkcji `sklej` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.8 (r,!) Napisz funkcję, która otrzymuje w argumencie串nypis i podmienia w nim wszystkie małe litery na duże litery.
Zakładamy, że napis przechowywany jest w tablicy o elementach typu `char`, składa się wyłącznie z liter łacińskich i białych znaków, oraz że system, na którym jest komplikowany i uruchamiany program, używa standardowego kodowania ASCII.
- 5.2.9 (r) Napisz funkcję `wytnij`, która dostaje jako argumenty串nypis oraz dwie liczby całkowite n i m , i wycina z otrzymanego napisu znaki o indeksach od n do m ($n \leq m$). Otrzymany w argumencie串nypis może mieć dowolną liczbę znaków (w tym mniejszą od n lub m)
Napisz dwie wersje funkcji `wytnij` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.10 (*,r) Napisz funkcję `wytnij2`, która dostaje jako argument dwa napisy `nap1` i `nap2`, i wycina z napisu `nap1` pierwsze wystąpienie w nim napisu `nap2`. Napisz dwie wersje funkcji `wytnij2` dla napisów składających się ze znaków typu `char` i `wchar_t`.

- 5.2.11 **(*,r)** Napisz funkcję **wytnijzw**, która dostaje jako argument dwa napisy **nap1** i **nap2**, i wycina z napisu **nap1** wszystkie znaki występujące także w napisie **nap2**. Napisz dwie wersje funkcji **wytnijz** dla napisów składających się ze znaków typu **char** i **wchar_t**.
- 5.2.12 **(*)** Napisz funkcję **wytnijzn**, która dostaje jako argument dwa napisy **nap1** i **nap2**, i wycina z napisu **nap1** wszystkie znaki niewystępujące w napisie **nap2**. Napisz dwie wersje funkcji **wytnijzn** dla napisów składających się ze znaków typu **char** i **wchar_t**.
- 5.2.13 **(*,r)** Napisz funkcję **wytnijtm**, która dostaje jako argument dwa napisy **nap1** i **nap2** o一样的 długości i wycina z napisu **nap1** znaki równe znakom występującym na tym samym miejscu w napisie **nap2** (znak o indeksie i usuwamy wtedy i tylko wtedy, gdy **nap1[i]=nap2[i]**). Napisz dwie wersje funkcji **wytnijtm** dla napisów składających się ze znaków typu **char** i **wchar_t**.

Dalsze zadania rozwiąż z użyciem funkcji bibliotecznych:

- 5.2.14 **(r)** Napisz funkcję, która wypisuje na standardowym wyjściu otrzymany w argumencie napis. Napisz dwie wersje funkcji: dla napisów składających się ze znaków typu **char** i **wchar_t**.
- 5.2.15 **(C++,r)** Napisz funkcję, która wypisuje na standardowym wyjściu otrzymany w argumencie napis. Napisz dwie wersje funkcji: dla napisów typu **string** i **wstring**.
- 5.2.16 **(r,!)** Napisz funkcję, która dostaje jako argument tablicę znaków i wczytuje do niej napis ze standardowego wejścia. Napisz dwie wersje funkcji: dla tablicy składających się ze znaków typu **char** i **wchar_t**.
- 5.2.17 **(C++,r)** Napisz funkcję, która dostaje w argumentach referencję do zmiennej typu **string** i wczytuje do niej napis ze standardowego wejścia. Napisz drugą wersję funkcji dla napisu typu **wstring**.
- 5.2.18 **(*,r)** Napisz funkcję, która otrzymuje w argumencie tablicę napisów (tablicę tablic a więc typ **char**** lub **wchar_t****) oraz jej rozmiar i zwraca jako wartość pierwszy leksykograficznie spośród przechowywanych w tablicy napisów (funkcja powinna zwrócić kopię znajdującą się w tablicy napisu).
- Zakładamy, że napisy zawierają wyłącznie małe litery łacińskie. Napisz dwie wersje funkcji: dla napisów składających się ze znaków typu **char** i **wchar_t**.
- 5.2.19 **(C++,r,!)** Napisz funkcję, która otrzymuje w argumencie tablicę napisów (tablicę o elementach typu **string** lub **wstring**) oraz jej rozmiar i zwraca jako wartość pierwszy leksykograficznie spośród przechowywanych w tablicy napisów (funkcja powinna zwrócić kopię znajdującą się w tablicy napisu).

Zakładamy, że napisy zawierają wyłącznie małe litery łacińskie. Napisz dwie wersje funkcji: dla napisów typu **string** i **wstring**.

- 5.2.20 **(r)** Napisz funkcję **godzina**, która dostaje w argumentach trzy liczby całkowite **godz**, **min** i **sek**, zawierające odpowiednio godziny, minuty oraz sekundy, i zwraca jako wartość napis zawierający godzinę w formacie **godz:min:sek**, w którym wartości poszczególnych pól pochodzą ze zmiennych podanych w argumentach.

Napisz dwie wersje funkcji **godzina**: zwracające napisy będące tablicami znaków typu **char** i typu **wchar_t**.

- 5.2.21 **(C++,r)** Napisz funkcję **godzina**, która dostaje w argumentach trzy liczby całkowite **godz**, **min** i **sek**, zawierające odpowiednio godziny, minuty oraz sekundy, i zwraca jako wartość napis zawierający godzinę w formacie **godz:min:sek**, w którym wartości poszczególnych pól pochodzą ze zmiennych podanych w argumentach .

Napisz dwie wersje funkcji **godzina**: zwracające napisy typu **string** i typu **wstring**.

- 5.2.22 **(r)** Napisz funkcję **sklej**, która dostaje w argumentach trzy napisy i zwraca jako wartość napis powstały ze sklejenia napisów otrzymanych w argumentach.

Napisz dwie wersje funkcji **sklej** operujące na napisach składających się ze znaków typu **char** i typu **wchar_t**.

- 5.2.23 **(C++,r,!)** Napisz funkcję **sklej**, która dostaje w argumentach trzy napisy i zwraca jako wartość napis powstały ze sklejenia napisów otrzymanych w argumentach.

Napisz dwie wersje funkcji **sklej** operujące na napisach typu **string** i typu **wstring**.

- 5.2.24 Napisz funkcję **kopiuj**, która dostaje jako argumenty napis oraz tablicę znaków i przepisuje napis do otrzymanej w argumencie tablicy znaków.

Napisz dwie wersje funkcji **kopiuj** operujące na napisach składających się ze znaków typu **char** i typu **wchar_t**.

- 5.2.25 Napisz funkcję **kopiuj**, która dostaje jako argumenty napis oraz wskaźnik do napisu (czyli wskaźnik do wskaźnika), tworzy nową tablicę znaków, kopiuje do niej napis zawarty w pierwszym argumencie, i przypisuje wskaźnik do nowo utworzonej tablicy do zmiennej wskazywanej przez drugi argument.

Napisz dwie wersje funkcji **kopiuj** operujące na napisach składających się ze znaków typu **char** i typu **wchar_t**.

- 5.2.26 **(r)** Napisz funkcję, która dostaje w argumencie napis i zamienia wszystkie występujące w nim małe litery na odpowiadające im duże litery.

Napisz dwie wersje funkcji operujące na napisach składających się ze znaków typu **char** i typu **wchar_t**.

5.2.27 (**C++,r**) Napisz funkcje, która dostaje w argumencie referencję do napisu i zamienia wszystkie występujące w nim małe litery na odpowiadające im duże litery.

Napisz dwie wersje funkcji operujące na napisach typów `string` i `wstring`.

ROZDZIAŁ 6

TABLICE WIELOWYMIAROWE

6.1.	Wprowadzenie	32
6.2.	Zadania	32

6.1. Wprowadzenie

W językach C i C++ są dwa rodzaje tablic dwuwymiarowych. Jeden rodzaj to tablice których elementami są tablice jednowymiarowe zaś drugi to tablice wskaźników do tablic jednowymiarowych. Analogicznie możemy stworzyć w C i C++ cztery rodzaje tablic trójwymiarowych różniące się sposobem utworzenia poszczególnych wymiarów. Nie istnieje ustalone polskie nazewnictwo rozróżniające różne rodzaje dynamicznych tablic wielowymiarowych w C i C++. W niniejszym zbiorze na określenie dwóch głównych typów tablic wielowymiarowych używa się określeń tablice tablic (na przykład dla typów `int**` czy `int **`) i tablice wielowymiarowe (na przykład dla typów `int [] [n]` lub `int [] [n] [m]`).

W język C++ nie ma możliwości używania wielowymiarowych tablic o wymiarach nieznanych w czasie kompilacji. W związku z tym wiele zadań w podrozdziale 6.2 przeznaczonych jest tylko dla uczących się języka C. Powyższy problem nie dotyczy wielowymiarowych tablic tablic.

6.2. Zadania

- 6.2.1 **(r,!)** Napisz funkcję, która dostaje jako argument dodatnie liczby całkowite n i m , tworzy dynamiczną dwuwymiarową tablicę tablic elementów typu `int` o wymiarach n na m , i zwraca jako wartość wskaźnik do niej.
- 6.2.2 **(C,r,!)** Napisz funkcję, która dostaje jako argument dodatnie liczby całkowite n i m , tworzy dynamiczną dwuwymiarową tablicę elementów typu `int` o wymiarach n na m i zwraca jako wartość wskaźnik do niej.
- 6.2.3 **(r,!)** Napisz funkcję, która dostaje jako argumenty wskaźnik do dwuwymiarowej tablicy tablic elementów typu `int` oraz jej wymiary: dodatnie liczby całkowite n i m , i usuwa z pamięci otrzymaną tablicę.
- 6.2.4 **(C,r)** Napisz funkcję, która dostaje jako argumenty wskaźnik do tablicy dwuwymiarowej elementów typu `int` oraz jej wymiary wymiary n i m , i usuwa z pamięci otrzymaną tablicę.
- 6.2.5 Rozwiąż zadania 6.2.1 i 6.2.3 w wersji z trójwymiarowymi tablicami tablic.
- 6.2.6 **(C)** Rozwiąż zadania 6.2.2 i 6.2.4 w wersji z tablicami trójwymiarowymi.
- 6.2.7 **(r)** Napisz funkcję, która dostaje jako argument dodatnią liczbę całkowitą n , tworzy dynamiczną dwuwymiarową trójkątną tablicę tablic elementów typu `int` o wymiarach n na n i zwraca jako wartość wskaźnik do niej.

- 6.2.8 (r) Napisz funkcję, która dostaje w argumentach dwuwymiarową tablicę elementów typu `int`, o pierwszym wymiarze podanym jako drugi argument funkcji oraz drugim wymiarze równym 100 i wypełnia ją zerami.
- 6.2.9 (r) Napisz funkcję, która dostaje w argumentach dwuwymiarową tablicę tablic elementów typu `int` oraz jej wymiary n i m , i wypełnia ją zerami.
- 6.2.10 (C,r) Napisz funkcję, która dostaje w argumentach tablicę dwuwymiarową elementów typu `int` oraz jej wymiary n i m , i wypełnia ją zerami.
- 6.2.11 Napisz funkcję, która dostaje w argumentach tablicę dwuwymiarową elementów typu `int`, o pierwszym wymiarze podanym jako drugi argument funkcji oraz drugim wymiarze równym 100, która to funkcja zwraca jako wartość sumę wartości elementów tablicy.
- 6.2.12 Napisz funkcję, która dostaje w argumentach dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary n i m , i zwraca jako wartość sumę wartości elementów tablicy.
- 6.2.13 (C) Napisz funkcję, która dostaje w argumentach tablicę dwuwymiarową o elementach typu `int` oraz jej wymiary n i m , i zwraca jako wartość sumę wartości elementów tablicy.
- 6.2.14 (r) Napisz funkcję, która dostaje w argumentach tablicę trójwymiarową o elementach typu `int` o wymiarach $100 \times 100 \times 100$, i zwraca jako wartość sumę wartości elementów tablicy.
- 6.2.15 Napisz funkcję, która dostaje w argumentach dodatnią liczbę całkowitą n oraz tablicę trójwymiarową o elementach typu `int` o wymiarach $n \times 100 \times 100$, i zwraca jako wartość sumę wartości elementów otrzymanej tablicy.
- 6.2.16 (r) Napisz funkcję, która dostaje jako argumenty dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i zwraca jako wartość indeks wiersza o największej średniej wartości elementów. Przyjmujemy, że dwa elementy leżą w tym samym wierszu, jeżeli mają taki sam pierwszy indeks.
- 6.2.17 (r) Napisz funkcję, która dostaje jako argumenty dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i zwraca największą spośród średnich wartości elementów poszczególnych wierszy. Przyjmujemy, że dwa elementy leżą w tym samym wierszu, jeżeli mają taki sam pierwszy indeks.
- 6.2.18 Napisz funkcję, która dostaje jako argument dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i wypisuje jej zawartość na standardowym wyjściu w taki sposób, żeby kolejne wiersze tablicy zostały wypisane w oddzielnych wierszach standardowego wyjścia.

Przyjmujemy, że dwa elementy leżą w tym samym wierszu, jeżeli mają taki sam drugi indeks.

- 6.2.19 **(r)** Napisz funkcję, która dostaje jako argumenty dwie dwuwymiarowe tablice tablic o elementach typu `int` oraz ich wymiary, i przepisuje zawartość pierwszej tablicy do drugiej tablicy.
- 6.2.20 Napisz funkcję, która dostaje jako argumenty dwie dwuwymiarowe tablice tablic o elementach typu `int` oraz ich wymiary, i zamienia zawartości obu tablic.
- 6.2.21 **(r)** Napisz funkcję, która dostaje jako argumenty dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i odwraca kolejność elementów we wszystkich wierszach otrzymanej tablicy (przyjmujemy, że dwa elementy tablicy leżą w tym samym wierszu, jeżeli mają taką samą pierwszą współrzędną).
- 6.2.22 **(C, r)** Napisz funkcję, która dostaje jako argumenty tablicę dwuwymiarową o elementach typu `int` oraz jej wymiary, i odwraca kolejność elementów we wszystkich wierszach otrzymanej tablicy (przyjmujemy, że dwa elementy tablicy leżą w tym samym wierszu, jeżeli mają taką samą pierwszą współrzędną).
- 6.2.23 **(r, !)** Napisz funkcję, która dostaje jako argumenty dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i zmienia kolejność wierszy w tablicy w taki sposób, że wiersz pierwszy ma się znaleźć na miejscu drugiego, wiersz drugi ma się znaleźć na miejscu trzeciego itd., natomiast ostatni wiersz ma się znaleźć na miejscu pierwszego (przyjmujemy, że dwa elementy tablicy leżą w tym samym wierszu jeżeli mają taką samą pierwszą współrzędną).
- 6.2.24 Napisz funkcję, która dostaje jako argumenty dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i zmienia kolejność kolumn w tablicy w taki sposób, że kolumna pierwsza ma się znaleźć na miejscu drugiej, kolumna druga ma się znaleźć na miejscu trzeciej itd., natomiast ostatnia kolumna ma się znaleźć na miejscu pierwszej (przyjmujemy, że dwa elementy tablicy leżą w tej samej kolumnie, jeżeli mają taką samą drugą współrzędną).
- 6.2.25 Napisz funkcję, która dostaje jako argumenty dwuwymiarową kwadratową tablicę tablic `tab` o elementach typu `int` oraz jej wymiar, i zmienia kolejność elementów w otrzymanej tablicy w następujący sposób: dla dowolnych `k` i `j` element `tab[k][j]` ma zostać zamieniony miejscami z elementem `tab[j][k]`.
- 6.2.26 Napisz funkcję, która dostaje jako argumenty dwuwymiarową prostokątną tablicę tablic `tab1` o wymiarach $n \times m$ i elementach typu `int` oraz jej wymiary, i zwraca jako wartość wskaźnik do nowo utworzonej dwuwymiarowej tablicy `tab2` o wymiarach $m \times n$ zawierającej

transponowaną macierz przechowywaną w tablicy `tab1` (czyli dla dowolnych k i j zachodzi $\text{tab1}[k][j] = \text{tab2}[j][k]$).

- 6.2.27 **(*,r)** Napisz funkcję, która dostaje jako argumenty dodatnią liczbę całkowitą n oraz trójwymiarową tablicę `tab` elementów typu `int` o wymiarach $n \times n \times n$, i zamienia elementy tablicy w taki sposób, że dla dowolnych i, j, k z zakresu od 0 do $n - 1$, wartość z komórki `tab[i][j][k]` po wykonaniu funkcji ma się znajdować w komórce `tab[k][i][j]`.
- 6.2.28 **(C,*)** Napisz funkcję, która dostaje jako argumenty dodatnią liczbę całkowitą n oraz trójwymiarową tablicę `tab` elementów typu `int` o wymiarach $n \times n \times n$, i zamienia elementy tablicy w taki sposób, że dla dowolnych i, j, k z zakresu od 0 do $n - 1$, takich że $i \leq j \leq k$, wartość z komórki `tab[i][j][k]` po wykonaniu funkcji ma się znajdować w komórce `tab[k][i][j]`.
- 6.2.29 **(*)** Napisz funkcję, która dostaje jako argument trójwymiarową tablicę `tab` elementów typu `int` o wymiarach $100 \times 100 \times 100$, i zamienia elementy tablicy w taki sposób, że dla dowolnych i, j, k z zakresu od 0 do 99, takich że $i \leq j \leq k$, wartość z komórki `tab[i][j][k]` po wykonaniu funkcji ma się znajdować w komórce `tab[k][i][j]`.
- 6.2.30 Napisz funkcję, która otrzymuje w argumentach dwie kwadratowe tablice tablic elementów typu `int` oraz ich wspólny wymiar, i zwraca jako wartość wynik dodawania macierzy przechowywanych w przekazanych argumentach. Wynik powinien zostać zwrócony w nowo utworzonej tablicy tablic.
- 6.2.31 **(C)** Napisz funkcję, która otrzymuje w argumentach dwie kwadratowe tablice dwuwymiarowe elementów typu `int` oraz ich wspólny wymiar, i zwraca jako wartość wynik dodawania macierzy przechowywanych w przekazanych argumentach. Wynik powinien zostać zwrócony w nowo utworzonej tablicy dwuwymiarowej.
- 6.2.32 **(*,r)** Napisz funkcję, która otrzymuje w argumentach dwie kwadratowe tablice tablic elementów typu `int` oraz ich wspólny wymiar, i zwraca jako wartość wynik mnożenia macierzy przechowywanych w przekazanych argumentach. Wynik powinien zostać zwrócony w nowo utworzonej tablicy tablic.
- 6.2.33 Napisz funkcję, która otrzymuje w argumentach trzy kwadratowe tablice tablic elementów typu `int` oraz ich wspólny wymiar, i zapisuje do trzeciej tablicy wynik mnożenia macierzy przechowywanych w dwóch pierwszych tablicach.
- 6.2.34 **(C,*)** Napisz funkcję, która otrzymuje w argumentach dwie kwadratowe tablice dwuwymiarowe elementów typu `int` oraz ich wspólny wymiar, i zwraca jako wartość wynik mnożenia macierzy przechowywają-

nych w przekazanych argumentach. Wynik powinien zostać zwrócony w nowo utworzonej tablicy dwuwymiarowej.

- 6.2.35 (*) Napisz funkcję, która otrzymuje w argumentach dwie prostokątne dwuwymiarowe tablice tablic elementów typu `int` o wymiarach odpowiednio $n \times m$ i $m \times k$ oraz ich wymiary, i zwraca jako wartość wynik mnożenia macierzy przechowywanych w przekazanych argumentach. Wynik powinien zostać zwrócony w nowo utworzonej tablicy tablic.
- 6.2.36 (*,r,!) Napisz funkcję otrzymującą dwa argumenty: dodatnią liczbę całkowitą n i dwuwymiarową kwadratową tablicę tablic elementów typu `int` o wymiarach $n \times n$, i zwraca jako wartość wyznacznik macierzy przechowywanej w otrzymanej w argumencie tablicy.
- 6.2.37 (C,*,r,!) Napisz funkcję otrzymującą dwa argumenty: dodatnią liczbę całkowitą n i kwadratową tablicę dwuwymiarową elementów typu `int` o wymiarach $n \times n$, i zwraca jako wartość wyznacznik macierzy przechowywanej w otrzymanej w argumencie tablicy.

ROZDZIAŁ 7

ZŁOŻONE TYPY DANYCH, LISTY WSKAŹNIKOWE

7.1.	Wprowadzenie	38
7.2.	Złożone typy danych	38
7.3.	Listy jednokierunkowe	41

7.1. Wprowadzenie

W tym rozdziale czytelnik znajdzie zadania związane ze złożonymi typami danych dostępnymi w C, a więc: strukturami, uniami i typami wyliczeniowymi. Duży nacisk został położony na zadania dotyczące list wskaźnikowych, jednego z popularnych zastosowań struktur. Zadania dotyczące list wskaźnikowych to okazja do przećwiczenia operacji na wskaźnikach oraz zarządzania pamięcią.

Istnieje wiele rodzajów list wskaźnikowych. Jednak ze względu na to, że nie jest to zbiór zadań ze struktur danych, w zadaniach pojawiają się wyłącznie listy jednokierunkowe. Zadania dotyczą zarówno list bez głowy, czyli takich, w których na początku listy znajduje się jej pierwszy element, jak i list z głową, czyli takich, w których na początku listy znajduje się sztuczny element nazywany głową. Dzięki dodaniu do listy głowy wiele operacji się upraszcza.

W języku C++ struktury to prawie to samo, co klasy. Różnią się tylko tym, że w przeciwieństwie do klas, pola i metody w strukturach są domyślnie publiczne. W tym rozdziale traktujemy jednak struktury wyłącznie jako tradycyjne złożone typy danych, tak jak jest to w języku C.

W podrozdziale 7.2 czytelnik znajdzie zadania mające na celu przećwiczenie definiowania złożonych typów danych oraz wykonywania prostych operacji na nich. W podrozdziale 7.3 znajduje się urozmaicony zestaw zadań, umożliwiający przećwiczenie używania list jednokierunkowych z głową i bez głowy. Na początku tego rozdziału znajdują się zadania wymagające implementacji podstawowych operacji na listach. Ze względu na problemy, jakie operowanie na listach sprawia początkującym programistom, wszystkie zadania z tej części podrozdziału 7.3 zostały rozwiązane.

7.2. Złożone typy danych

- 7.2.1 (**r,róż**) Zdefiniuj strukturę **trojkat** przechowującą długości boków trójkąta. Napisz funkcję, która otrzymuje jako argument zmienią typu **struct trojkat**, i zwraca jako wartość obwód trójkąta przekazanego w argumencie.
- 7.2.2 (**r,!**) Napisz funkcję, która otrzymuje jako argumenty zmiennej **troj1** typu **struct trojkat** zdefiniowanego w zadaniu 7.2.1 oraz zmiennej **troj2** wskaźnik na zmiennej typu **struct trojkat**, i przepisuje wartość zmiennej **troj1** do zmiennej wskazywanej przez **troj2**.
- 7.2.3 (**r**) Zdefiniuj strukturę **punkt** służącą do przechowywania współrzędnych punktów w trójwymiarowej przestrzeni kartezjańskiej.

Napisz funkcję, która otrzymuje jako argumenty tablicę `tab` o argumentach typu `struct punkt` oraz jej rozmiar, i zwraca jako wartość najmniejszą spośród odległości pomiędzy punktami przechowywanymi w tablicy `tab`. Zakładamy, że otrzymana w argumencie tablica ma co najmniej dwa argumenty.

- 7.2.4 (r,!) Napisz funkcję, która otrzymuje jako argumenty tablice `tab1` i `tab2` o argumentach typu `struct punkt` zdefiniowanego w rozwiązaniu zadania 7.2.3 oraz ich rozmiar, i przepisuje zwartość tablicy `tab1` do tablicy `tab2`.

- 7.2.5 (r,!) Zdefiniuj strukturę `punkt10` służącą do przechowywania współrzędnych punktów w dziesięciowymiarowej przestrzeni kartezjańskiej. Do przechowywania poszczególnych wymiarów wykorzystaj tablicę dziesięcioelementową.

Napisz funkcję, która otrzymuje jako argumenty tablice `tab1` i `tab2` typu `struct punkt10` oraz ich wspólny rozmiar, i przepisuje zwartość tablicy `tab1` do tablicy `tab2`.

- 7.2.6 (r,!) Zdefiniuj strukturę `punktn` służącą do przechowywania współrzędnych punktów w n -wymiarowej przestrzeni kartezjańskiej. Do przechowywania poszczególnych wymiarów wykorzystaj tablicę n -elementową. W strukturze `punktn` przechowuj także ilość wymiarów przestrzeni.

Napisz funkcję, która otrzymuje jako argumenty tablice `tab1` i `tab2` o argumentach typu `struct punktn` oraz ich wspólny rozmiar, i przepisuje zwartość tablicy `tab1` do tablicy `tab2`. Zakładamy, że tablica `tab2` jest pusta (czyli nie musimy się martwić o jej poprzednią zawartość).

- 7.2.7 Zdefiniuj strukturę `zespolone` służącą do przechowywania liczb zespolonych. Zdefiniowana struktura powinna zawierać pola `im` i `re` typu `double` służące do przechowywania odpowiednio części urojonej i rzeczywistej liczby zespolonej.

Napisz funkcję `dodaj`, która dostaje dwa argumenty typu `zespolone` i zwraca jako wartość ich sumę.

- 7.2.8 Zdefiniuj strukturę `student` służącą do przechowywania danych osobowych studenta (struktura powinna zawierać takie pola, jak: `imie`, `nazwisko`, `adres`, `pesel`, `kierunek` i `numer legitymacji`).

Napisz funkcję, która otrzymuje jako argument wskaźnik na strukturę `student` i wczytuje dane ze standardowego wejścia do rekordu wskazywanego przez argument funkcji.

- 7.2.9 (r) Zdefiniuj strukturę `lista` posiadającą dwa pola: jedno typu `int` oraz drugie będące wskaźnikiem do definiowanego typu.

- 7.2.10 (r) Zdefiniuj unię `super_int`, w której będzie można przechowywać zarówno zmienne typu `int`, jak i `unsigned int`.
- 7.2.11 (r) Zdefiniuj unię `Liczba`, która może służyć w zależności od potrzeb do przechowywania liczby wymiernej lub liczby całkowitej. Zdefiniuj strukturę `Dane`, o dwóch polach polu `tp` typu `int` oraz polu `zaw` typu `Liczba`.
Napisz bezargumentową funkcję, która wczytuje ze standardowego wejścia zawartość do struktury `Dane` i zwraca ją jako wartość. Funkcja powinna pytać użytkownika, czy chce wczytać liczbę całkowitą, czy wymierną oraz w zależności od jego wyboru wstawić do pola `tp` wartość 0 lub 1. Następnie funkcja powinna wczytać do pola `zaw` wartość odpowiedniego typu.
- 7.2.12 (*) Zdefiniuj strukturę `zespolone`, która ma służyć do przechowywania liczb zespolonych oraz unię `Liczba` mogącą przechowywać liczby wymierne i całkowite. Części urojona i rzeczywista liczby zespolonej powinny być przechowywane w polach `im` i `re` typu `Liczba`. Struktura `zespolone` powinna mieć dodatkowe pole `tp` przechowujące informację jakiego typu wartości przechowywane są w polach `im` i `re` (zakładamy, że oba są tego samego typu).
Napisz funkcję `dodaj`, która dostaje dwa argumenty typu `zespolone` i zwraca jako wartość ich sumę. Zwróć uwagę na zgodność typów składników i zwracanej wartości (suma dwóch liczb całkowitych jest liczbą całkowitą, natomiast jeżeli którykolwiek ze składników jest wymierny to i suma jest wymierna).
- 7.2.13 (r) Zdefiniuj strukturę `figura` przechowującą wymiary figur geometrycznych niezbędne do obliczenia pola. Struktura powinna mieć możliwość przechowywania wymiarów takich figur, jak: trójkąt, prostokąt, równoległobok i trapez. Rodzaj przechowywanej figury powinien być zakodowany w wartości pola `fig` typu `int`. Definiując strukturę, staraj się zużyć jak najmniej pamięci.
Napisz funkcję `pole`, która dostaje jako argument zmienną `f` typu `struct figura` i zwraca jako wartość pole figury której wymiary przechowuje zmienna `f`.
- 7.2.14 (r) Zdefiniuj typ wyliczeniowy `czworokat`, mogący przyjmować wartości odpowiadające nazwom różnych czworokątów.
- 7.2.15 Zdefiniuj typ wyliczeniowy `zwierzak`, mogący przyjmować wartości odpowiadające nazwom różnych zwierząt domowych.
- 7.2.16 Zdefiniuj typ wyliczeniowy, służący do przechowywania informacji o stanie połączenia internetowego, o trzech wartościach odpowiadających trzem stanom: połączenie nawiązane, połączenie nienawiązane i połączenie w trakcie nawiązywania. Następnie zdefiniuj strukturę `komputer`

przechowującą stan połączenia, IP podłączonego komputera (jako napis) oraz nazwę jego właściciela.

Napisz funkcję, która jako argument otrzymuje strukturę komputer i wyświetla na standardowym wyjściu jej zawartość w sposób przyjazny dla użytkownika.

- 7.2.17 (**r,!róż**) Zdefiniuj strukturę dane osobowe zawierającą pola: imie, nazwisko, plec, stan_cywילny. W zależności od płci pole stan_cywילny powinno móc mieć jedną z dwóch wartości wolny lub zonaty dla mężczyzn i wolna lub mezatka dla kobiet.

Napisz funkcję wczytaj o dwóch argumentach: tablicy tab o elementów typu stan_cywילny i jej rozmiarze. Funkcja powinna wczytywać do komórek tablicy tab wartości podane na standardowym wejściu.

- 7.2.18 (**r,!**) Zdefiniuj złożony typ danych dzięki któremu będzie można odnosić się do kolejnych bajtów zmiennej typu unsigned int jak do kolejnych elementów tablicy.

- 7.2.19 (**r**) Wykorzystując typ danych zdefiniowany w rozwiązaniu zadania 18 napisz funkcję, która dostaje w argumentach dwie nieujemne liczby całkowite typu unsigned int i zwraca jako wartość nieujemną liczbę całkowitą, której kolejne bajty są iloczynami modulo 256 odpowiadających sobie bajtów liczb podanych w argumentach.

7.3. Listy jednokierunkowe

Jednokierunkowe listy bez głowy

Listing 7.1. Definicja struktury element

```
struct element{
    int i;
    struct element * next;
};
```

Struktura element może być wykorzystana w implementacji jednokierunkowej listy wskaźnikowej. Najpierw zostaną przedstawione zadania umożliwiające przećwiczenie operacji na liście jednokierunkowej bez głowy (czyli takiej, która jest reprezentowana przez wskaźnik na pierwszy element). Listę pustą reprezentuje wskaźnik o wartości NULL. Pamiętaj, że pole next ostatniego elementu listy powinno mieć wartość NULL. Dzięki temu będzie możliwe rozpoznanie końca listy.

- 7.3.1 (**r**) Napisz funkcję utworz zwracającą wskaźnik do pustej listy bez głowy o elementach typu element.

- 7.3.2 (r) Napisz funkcję `wyczysc`, która dostaje jako argument wskaźnik do pierwszego elementu listy wskaźnikowej bezgłowy o elementach typu `element` i usuwa wszystkie elementy listy.
- 7.3.3 (r) Napisz funkcję `dodaj` o dwóch argumentach `Lista` typu `element*` oraz `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna dodawać na początek listy reprezentowanej przez zmienną `Lista` nowy element o wartości `a` pola `i` oraz zwracać wskaźnik do pierwszego elementu tak powiększonej listy.
- 7.3.4 (r) Napisz funkcję `dodajk` o dwóch argumentach `Lista` typu `element*` i `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna dodawać na koniec listy reprezentowanej przez zmienną `Lista` nowy element o wartości `a` pola `i` oraz zwracać wskaźnik do pierwszego elementu tak powiększonej listy.
- 7.3.5 (r) Napisz funkcję `dodajw` o trzech argumentach `Lista` i `elem` typu `element*` oraz `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna dodawać element o wartości `a` pola `i` do listy reprezentowanej przez zmienną `Lista` na miejscu tuż za elementem wskazywanym przez `elem`. W przypadku, gdy `elem` jest równy `NULL` funkcja powinna wstawić nowy element na początek listy. Funkcja powinna zwrócić jako wartość wskaźnik do pierwszego elementu powiększonej listy.
- 7.3.6 (r,!) Napisz funkcję `znajdz` o dwóch argumentach `Lista` typu `element*` i `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna sprawdzać, czy na liście reprezentowanej przez zmienną `Lista` znajduje się element o polu `i` równym `a`. Jeżeli tak, to funkcja powinna zwrócić wskaźnik do tego elementu. W przeciwnym wypadku funkcja powinna zwrócić wartość `NULL`.
- 7.3.7 (r) Napisz funkcję `usun` o dwóch argumentach `Lista` typu `element*` i `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna usuwać z listy reprezentowanej przez zmienną `Lista` element o wartości `a` pola `i` (o ile taki element znajduje się na liście) oraz zwracać wskaźnik do pierwszego elementu zmodyfikowanej listy (jeżeli po usunięciu elementu lista będzie pusta, to funkcja powinna zwrócić wartość `NULL`).
- 7.3.8 (r,!) Napisz funkcję `usunw` o dwóch argumentach `Lista` i `elem` typu `element*` i zwracającą wskaźnik do typu `element`. Funkcja powinna usuwać z listy reprezentowanej przez zmienną `Lista` element wskazywany przez `elem` oraz zwracać wskaźnik do pierwszego elementu zmodyfikowanej listy (jeżeli po usunięciu elementu lista będzie pusta, to funkcja powinna zwrócić wartość `NULL`). Dla `elem` równego `NULL` funkcja `usunw` nie powinna nic robić.
- 7.3.9 (r,!) Napisz funkcję `usunw2` o dwóch argumentach `Lista` i `elem` typu

`element*` i zwracającą wskaźnik do typu `element`. Funkcja powinna usuwać z listy reprezentowanej przez zmienną `Lista` element wskażywany przez `elem->next` oraz zwracać wskaźnik do pierwszego elementu zmodyfikowanej listy (jeżeli po usunięciu elementu lista będzie pusta, to funkcja powinna zwrócić wartość `NULL`). Dla `elem` równego `NULL` funkcja powinna usunąć pierwszy element listy (o ile taki istnieje). Dla `elem` różnego od `NULL` i `elem->next` równego `NULL` funkcja `usunw2` nie powinna nic robić.

Jednokierunkowe listy z głową

W zadaniach dotyczących jednokierunkowych list wskaźnikowych z głową (czyli takich, na początku których znajduje się „sztuczny” pusty element, nazywany głową) zostanie wykorzystana struktury `element` zdefiniowaną w Listingu 7.1 . Pamiętaj, że pole `next` ostatniego elementu listy powinno mieć wartość `NULL`, w ten sposób będzie możliwe rozpoznanie końca listy.

- 7.3.10 (r) Napisz funkcję `utworz` tworzącą pustą listę z głową o elementach typu `element` i zwracającą jako wartość wskaźnik do głowy utworzonej listy.
- 7.3.11 (r) Napisz funkcję `wyczysc`, która dostaje jako argument wskaźnik do listy wskaźnikowej z głową o elementach typu `element` i usuwa wszystkie elementy listy (razem z głową).
- 7.3.12 (r) Napisz funkcję `dodaj` o dwóch argumentach `Lista` typu `element*` i `a` typu `int`. Funkcja powinna dodawać na początek listy reprezentowanej przez zmienną `Lista` nowy element o wartości `a` pola `i`.
- 7.3.13 (r) Napisz funkcję `dodajk` o dwóch argumentach `Lista` typu `element*` i `a` typu `int`. Funkcja powinna dodawać na koniec listy reprezentowanej przez zmienną `Lista` nowy element o wartości `a` pola `i`.
- 7.3.14 (r) Napisz funkcję `dodajw` o trzech argumentach `Lista` oraz `elem` typu `element*` i `a` typu `int`. Funkcja powinna dodawać element o wartości `a` pola `i` do listy reprezentowanej przez zmienną `Lista` na miejscu tuż za elementem wskażowanym przez `elem`.
- 7.3.15 (r) Napisz funkcję `znajdz` o dwóch argumentach `Lista` typu `element*` i `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna sprawdzać, czy na liście reprezentowanej przez zmienną `Lista`, znajduje się element o polu `i` równym `a`. Jeżeli tak, to funkcja powinna zwrócić wskaźnik do tego elementu. W przeciwnym wypadku funkcja powinna zwrócić wartość `NULL`.
- 7.3.16 (r) Napisz funkcję `znajdzp` o dwóch argumentach `Lista` typu `element*` i `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna sprawdzać, czy na liście reprezentowanej przez zmienną `Lista`, znajduje się element o polu `i` równym `a`. Jeżeli tak, to funkcja powinna zwrócić wskaźnik do elementu go poprzedzającego. W przeciwnym

wypadku funkcja powinna zwrócić wskaźnik do ostatniego elementu listy.

- 7.3.17 (r) Napisz funkcję **usun** o dwóch argumentach **Lista** typu **element*** i **a** typu **int**. Funkcja powinna usuwać z listy reprezentowanej przez zmienną **Lista** element o wartości **a** pola **i** (o ile taki element znajduje się na liście).
- 7.3.18 (r) Napisz funkcję **usunw** o dwóch argumentach **Lista** i **elem** typu **element***. Funkcja powinna usuwać z listy reprezentowanej przez zmienną **Lista** element wskazywany przez zmienną **elem**.
- 7.3.19 (r) Napisz funkcję **usunw2** o dwóch argumentach **Lista** i **elem** typu **element***. Funkcja powinna usuwać z listy reprezentowanej przez zmienną **Lista** element wskazywany przez **elem->next**.

Pozostałe zadania z list jednokierunkowych

- 7.3.20 (r) Napisz funkcję **zeruj**, która dostaje jako argument listę wskaźnikową o elementach typu **element** i nadaje wartość 0 polom **i** we wszystkich elementach listy. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.21 Napisz funkcję **bezwzględna**, która dostaje jako argument listę wskaźnikową o elementach typu **element** i zapisuje do pól **i** wszystkich elementów listy wartość bezwzględną ich pierwotnej wartości. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.22 Zdefiniuj strukturę **trojkat** mogącą służyć jako typ elementów listy jednokierunkowej. Struktura **trojkat** powinna posiadać pola służące do przechowywania wszystkich boków trójkąta oraz jego pola.
Napisz funkcję **pole**, która otrzymuje w argumencie listę wskaźnikową o elementach typu **trojkat** i we wszystkich elementach listy do odpowiedniego pola wstawia wartość pola trójkąta o bokach, których długość przechowuje dana struktura. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.23 (r) Zdefiniuj strukturę **trojka** mającą służyć jako typ elementu jednokierunkowej listy wskaźnikowej przechowującej trójkę dodatnich liczb całkowitych **a, b, c**.
Napisz funkcję **pitagoras**, która dostaje w argumencie listę wskaźnikową o elementach typu **trojka** i usuwa z otrzymanej listy wszystkie elementy nieprzechowujące trójkę pitagorejskich (czyli takich, że $a^2 + b^2 = c^2$). Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy. W wersji dla list bez głowy funkcja powinna zwracać wskaźnik do pierwszego elementu przekształconej listy. Jeżeli wynikowa lista bez głowy będzie pusta, funkcja powinna zwrócić **NULL**.
- 7.3.24 (r) Napisz funkcję **suma**, która dostaje jako argument listę wskaźnikową o elementach typu **element** i zwraca jako wartość sumę pól **i**

ze wszystkich elementów listy. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.

- 7.3.25 (r) Napisz funkcję `minimum`, która dostaje jako argument listę wskaźnikową o elementach typu `element` i zwraca jako wartość najmniejszą spośród wartości pól `i` elementów listy. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.26 (r) Napisz funkcję `minimum`, która dostaje jako argument `Lista` listę wskaźnikową o elementach typu `element` i zwraca jako wartość wskaźnik do elementu listy o najmniejszej wartości pola `i`. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.27 (r) Napisz funkcję `minimum`, która dostaje jako argument listę wskaźnikową o elementach typu `element` i zwraca jako wartość wskaźnik do bezpośredniego poprzednika elementu listy o najmniejszej wartości pola `i`. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy. W przypadku listy bez głowy, jeżeli najmniejszą wartość pola `i` ma pierwszy element listy lub gdy otrzymana w argumencie lista jest pusta, funkcja `minimum` powinna zwrócić wartość `NULL`.
- 7.3.28 (r) Napisz funkcję, która dostaje jako argument listę o elementach typu `element` i zwraca jako wartość największą na wartość bezwzględną spośród różnic pomiędzy polami `i` w różnych elementach listy otrzymanej w argumencie. Zakładamy, że otrzymana w argumencie funkcji lista jest co najmniej dwuelementowa. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.29 (r) Napisz funkcję `kopiuj`, która jako argument otrzymuje jednokierunkową listę wskaźnikową o elementach typu `element`, tworzy kopię otrzymanej w argumencie listy i zwraca jako wartość wskaźnik do kopii. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.30 (r) Napisz funkcję `doklej` o dwóch argumentach `Lista1` i `Lista2` typu `element *` wskazujących na dwie jednokierunkowe listy wskaźnikowe bez głowy, która wstawia na koniec listy `Lista1` elementy listy `Lista2` (funkcja nie powinna alokować w pamięci nowych zmiennych typu `element`, ale odpowiednio podpisać już istniejące). Funkcja powinna zwracać wskaźnik do pierwszego elementu połączonej listy.
- 7.3.31 (r,!) Napisz funkcję, która dostaje w argumentach wskaźnik do listy wskaźnikowej o elementach typu `element` i odwraca kolejność elementów listy. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy. W wersji dla list bez głowy funkcja powinna zwracać wskaźnik do pierwszego elementu odwróconej listy.
- 7.3.32 (r) Napisz funkcję która dostaje w argumentach wskaźniki do dwóch list wskaźnikowych bez głowy o elementach typu `element` i równej długości, tworzy listę złożoną z elementów obu list ułożonych naprzemienne (pierwszy ma być pierwszy element pierwszej listy, następ-

nie pierwszy element drugiej listy, drugie element pierwszej listy itd.) i zwraca jako wartość wskaźnik do pierwszego elementu nowo utworzonej listy.

- 7.3.33 (**r,!**) Napisz funkcję, która dostaje jako argument listę wskaźnikową o elementach typu **element** i przesuwa jej elementy w taki sposób, że pierwszy element będzie drugi, drugi element będzie trzeci etc. a na pierwszym miejscu będzie ostatni element pierwotnej listy. Dla listy o długości nie większej niż 1 funkcja nie powinna nic robić. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy. W wersji dla list bez głowy funkcja powinna zwracać wskaźnik do pierwszego elementu przekształconej listy.
- 7.3.34 (**r**) Napisz funkcję, która dostaje jako argumenty dwie listy wskaźnikowe **Lista1** i **Lista2** o elementach typu **element** i tworzy nową listę o elementach tego samego typu przechowującą wartości występujące w polach **i** elementów zarówno listy **Lista1** jak i **Lista2**. W stworzonej liście nie powinny powtarzać się przechowywane w elementach wartości. Funkcja powinna zwracać wskaźnik do nowo utworzonej listy. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.35 (***,r,!**) Napisz funkcję sortującą rosnąco podaną w argumencie listę o elementach typu **element**. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy. W wersji dla listy bez głowy funkcja powinna zwracać wskaźnik do pierwszego elementu posortowanej listy.

ROZDZIAŁ 8

OPERACJE NA PLIKACH

8.1.	Wprowadzenie	48
8.2.	Zadania	48

8.1. Wprowadzenie

W językach C i C++ pliki powiązane są ze strumieniami i pracuje się na nich podobnie jak na innych strumieniach. W języku C do operacji na plikach służą funkcje z biblioteki `stdio`, a wśród nich między innymi: `fopen`, `fclose`, `fwrite`, `fread`, `fprintf`, `fscanf`, `feof` i `fseek`. W języku C++ do operowania na plikach służą klasy `ifstream`, `ofstream` i `fstream` znajdujące się w bibliotece `fstream`. Szczegóły czytelnik znajdzie w literaturze.

Aby zacząć operować na pliku należy go otworzyć, zaś po zakończeniu pracy należy plik zamknąć. Otwierając plik należy określić w jakim celu plik jest otwierany (do czytania, do pisania etc.) oraz czy plik ma być otwarty w trybie binarnym (czyli jako ciąg bajtów) czy tekstowym. Aby zrozumieć różnicę pomiędzy plikiem binarnym a tekstowym można zapisać do obu rodzajów plików jednobajtową bezznakową liczbę całkowitą o wartości 100. W pliku binarnym znajdzie się jeden bajt zawierający binarnie zapisaną liczbę 100, natomiast w pliku tekstowym znajdą się trzy bajty, z których pierwszy będzie zawierał kod znaku '1', a dwa kolejne bajty będą zawierały kod znaku '0'.

Częstym błędem u początkujących programistów C jest niewłaściwe użycie funkcji `feof`. Używając jej należy pamiętać, że ta funkcja zwraca `true` dopiero wtedy, gdy nie powiedzie się próba czytania. Czyli to, że `feof` ma wartość `false`, nie oznacza, że w pliku pozostało jeszcze coś do przeczytania. Podobny problem dotyczy metody `eof` klasy `fstream`.

Inną rzeczą, o której należy pamiętać, to fakt, że strumieni w języku C++ nie można kopiować. Można przekazywać je co najwyżej przez referencję lub wskaźnik.

W treściach zadań wielokrotnie pojawia się pojęcie deskryptora. W systemie UNIX jest to liczba całkowita jednoznacznie identyfikująca otworzony plik. Chcąc pisać do lub czytać z otworzonego pliku należy podać jego deskryptor. W treści zadań dla uproszczenia pojęcie deskryptora używane jest w odniesieniu do wartości typu `FILE *`.

8.2. Zadania

8.2.1 (**r,róż**) Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku, otwiera plik do tekstowego czytania i zwraca jako wartość deskryptor świeżo otwartego pliku (w wersji dla języka C++ funkcja powinna zwrócić wskaźnik do obiektu klasy `fstream`).

8.2.2 (**r,!**) Napisz funkcję, która dostaje jako argument deskryptor do pliku tekstowego otwartego do czytania (w wersji dla języka C++ referencję

- do obiektu klasy `fstream`), wypisuje zawartość pliku na standardowe wyjście i zamyka plik.
- 8.2.3 **(r,!)** Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku tekstowego i wypisuje na standardowym wyjściu zawartość pliku z pominięciem białych znaków.
Napisz dwie wersje funkcji dla znaków typów `char` i `wchar_t`.
- 8.2.4 **(r)** Napisz funkcję, która dostaje jako argumenty ścieżkę dostępu do pliku tekstowego oraz znak `c` i zwraca jako wartość liczbę wystąpień znaku `c` w podanym w argumencie pliku.
Napisz dwie wersje funkcji dla znaków typów `char` i `wchar_t`.
- 8.2.5 Napisz funkcję, która dostaje w argumencie ścieżkę dostępu do pliku tekstowego i wypisuje na standardowym wyjściu statystyki występowania w pliku poszczególnych znaków (zakładamy, że znaki są typu `char`).
8.2.6 Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku tekstowego i zwraca jako wartość najczęściej występujący w pliku znak (zakładamy, że znaki są typu `char`).
8.2.7 Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku tekstowego zawierającego liczby całkowite oddzielone białymi znakami i zwraca jako wartość sumę znajdujących się w pliku liczb.
8.2.8 Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku tekstowego zawierającego liczby całkowite oddzielone białymi znakami i zwraca jako wartość najmniejszą spośród znajdujących się w pliku liczb.
8.2.9 **(r,!)** Napisz funkcję, która dostaje jako argumenty ścieżki dostępu do dwóch plików tekstowych i zwraca jako wartość 1, jeżeli podane pliki mają taką samą zawartość oraz 0 w przeciwnym wypadku.
Napisz dwie wersje funkcji dla znaków typu `char` i `wchar_t`.
- 8.2.10 **(r)** Napisz funkcję, która dostaje jako argumenty ścieżki dostępu do dwóch plików tekstowych i zwraca jako wartość 1, jeżeli podane pliki mają taką samą zawartość z dokładnością do białych znaków oraz 0 w przeciwnym wypadku.
Napisz dwie wersje funkcji dla znaków typu `char` i `wchar_t`.
- 8.2.11 **(r)** Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku, otwiera plik do tekstowego pisania z kursorem ustawionym na końcu pliku i zwraca jako wartość deskryptor świeżo otwartego pliku (w wersji dla języka C++ funkcja powinna zwrócić wskaźnik do obiektu klasy `fstream`).
8.2.12 **(r)** Napisz funkcję, która dostaje jako argument deskryptor do pliku tekstowego otwartego do pisania (w wersji dla języka C++ referencję do obiektu klasy `fstream`) oraz liczbę `n`, wczytuje ze standardowego

wejścia n wersów tekstu, zapisuje do pliku wczytany tekst i zamyka plik.

Napisz dwie wersje funkcji dla znaków typów `char` i `wchar_t`.

- 8.2.13 **(r)** Napisz funkcję, która dostaje jako argumenty deskryptory dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego pliku.
- 8.2.14 **(r)** Napisz funkcję, która dostaje jako argumenty ścieżki dostępu do dwóch plików (w wersji dla języka C++ referencje do obiektów klasy `fstream`) i przepisuje zawartość pierwszego pliku do drugiego pliku (stara zawartość drugiego pliku ma zostać skasowana).
- 8.2.15 Napisz funkcję, która dostaje jako argumenty ścieżki dostępu do dwóch plików i dopisuje zawartość pierwszego pliku na koniec drugiego pliku.
- 8.2.16 Napisz funkcję, która dostaje w argumentach jednowymiarową tablicę liczb całkowitych `tab`, jej rozmiar oraz ścieżkę dostępu do pliku tekstowego, i dopisuje w kolejnych wierszach na końcu otrzymanego pliku wartości kolejnych elementów tablicy `tab`.
- 8.2.17 **(*,r,!)** Napisz funkcję, która dostaje jako argumenty nazwę pliku, dwuwymiarową tablicę tablic o elementach typu `int` oraz wymiary tablicy i zapisuje binarnie zawartość tablicy do podanego pliku.
- 8.2.18 **(*,r,!)** Napisz funkcję, która dostaje jako argumenty nazwę pliku, dwuwymiarową tablicę tablic o elementach typu `int` oraz wymiary tablicy i wczytuje binarnie zawartość pliku do tablicy. Napisz funkcję tak, aby była „kompatybilna” z funkcją z zadania 8.2.17.
- 8.2.19 **(*)** Napisz funkcję, która dostaje jako argumenty nazwę pliku, dwuwymiarową tablicę tablic o elementach typu `int` oraz wymiary tablicy i zapisuje binarnie zawartość tablicy oraz jej wymiary do podanego pliku.
- 8.2.20 **(*)** Napisz funkcję, która dostaje jako argumenty nazwę pliku, wczytuje zawartość pliku do nowo utworzonej dwuwymiarowej tablicy tablic. Wymiary tablicy powinny być podane w pliku. Napisz funkcję tak, aby była „kompatybilna” z funkcją z zadania 8.2.19.

ROZDZIAŁ 9

INSTRUKCJE PREPROCESORA, APLIKACJE WIELOPLIKOWE, MAKEFILE.

9.1.	Wprowadzenie	52
9.2.	Makra	52
9.3.	Aplikacje wieloplikowe, makefile	53

9.1. Wprowadzenie

Przed właściwym procesem komplikacji, czyli tłumaczeniem programu napisanego w języku wysokiego poziomu na kod maszynowy, w przypadku języków C i C++ kod programu przetwarzany jest przez preprocesor. W początkach języka C preprocesor odgrywał bardzo dużą rolę. Jednak w wyniku rozwoju tego języka, w tym dodania do niego m. in. takich elementów jak stałe, czy funkcje inline, znaczenia preprocesora zmalało. Nie bez znaczenia był też rozwój debuggerów, które znacznie ułatwiały szukanie błędów w programach, co wcześniej było jedną z dziedzin, w której wykorzystywano dyrektywy preprocesora. Obecnie dyrektywy preprocesora wykorzystuje się niemal wyłącznie dołączania bibliotek i tworzenia aplikacji wieloplikowych.

9.2. Makra

W zadaniach w tym podrozdziale należy napisać makrodefinicje. Makrodefinicje, nazywane krócej makrami, tworzy się przy pomocy dyrektywy `#define`.

- 9.2.1 **(r,!)** Napisz makro, które dostaje trzy argumenty i zwraca ich sumę.
- 9.2.2 Napisz makro, które dla trzech otrzymanych w argumentach liczb zwraca ich średnią.
- 9.2.3 **(r)** Napisz makro, które dostaje jako argumenty dwie liczby całkowite i wypisuje na standardowym wyjściu większą z nich.
- 9.2.4 **(r)** Napisz jednoargumentowe makro, które zwraca wartość 1 jeżeli argumentem jest liczba parzysta i 0 jeżeli argument jest nieparzysty.
- 9.2.5 Napisz makro, które dostaje dwa argumenty i zwraca większy z nich (zakładamy, że otrzymane wartości są porównywalne).
- 9.2.6 **(r,!)** Napisz makro, które dostaje trzy liczby całkowite jako argumenty i wypisuje na standardowym wyjściu największą z otrzymanych wartości.
- 9.2.7 **(r)** Napisz makro, które dostaje trzy argumenty i zwraca największą z otrzymanych wartości (zakładamy, że otrzymane wartości są porównywalne).
- 9.2.8 **(r)** Napisz makro o dwóch argumentach `x` i `n`, które działa jak pętla `for+`, w której zmienna `x` przebiega wartości od 0 do `n-1`.
- 9.2.9 Napisz makro o dwóch argumentach `x` i `n`, które działa jak pętla `for`, w której zmienna `x` przebiega wartości od `n` do 0.
- 9.2.10 **(r)** Napisz makro, które nadaje wartość 0 podanej w argumencie zmiennej.

9.2.11 Napisz makro, które zwiększa o 2 podaną w argumencie zmienną liczbową.

9.3. Aplikacje wieloplikowe, makefile

- 9.3.1 (r) Napisz program, który oblicza miejsca zerowe wielomianu drugiego stopnia o współczynnikach wczytanych ze standardowego wejścia. W programie wykorzystaj samodzielnie napisaną funkcję liczącą pierwiastek z liczby dodatniej. Funkcję liczącą pierwiastek umieść w oddzielnym pliku.
- 9.3.2 Napisz program, który oblicza wartość wielomianu jednej zmiennej o współczynnikach podanych przez użytkownika w punkcie podanym przez niego. W programie wykorzystaj samodzielnie napisaną funkcję liczącą potęgę. Funkcję liczącą potęgę umieść w oddzielnym pliku.
- 9.3.3 (r,!) Napisz program, który oblicza miejsca zerowe wielomianu drugiego stopnia o współczynnikach wczytanych ze standardowego wejścia. W programie wykorzystaj samodzielnie napisaną funkcję liczącą pierwiastek z liczby dodatniej. Program podziel na trzy pliki: plik zawierający główny program, plik zawierający funkcję pierwiastkującą, oraz plik z nagłówkiem funkcji pierwiastkującej.
- 9.3.4 (r,!) Napisz plik makefile pozwalający na komplikację programu z zadania 9.3.3.
- 9.3.5 Napisz program, który oblicza wartość wielomianu jednej zmiennej o współczynnikach podanych przez użytkownika w punkcie podanym przez użytkownika. W programie wykorzystaj samodzielnie napisaną funkcję liczącą potęgę. Program podziel na trzy pliki: plik zawierający główny program, plik zawierający funkcję potęgującą oraz plik z nagłówkiem funkcji potęgującej.
- 9.3.6 Napisz plik makefile pozwalający na komplikację programu z zadania 9.3.5.
- 9.3.7 (r) Napisz program, który wczytuje ze standardowego wejścia ciąg liczb zespolonych i wypisuje na standardowym wyjściu sumę wczytyanych liczb. Program powinien składać się z trzech części:
- programu głównego,
 - biblioteki zawierającej definicje typu `zespolone` oraz funkcji do wczytywania i wypisywania elementów tego typu,
 - biblioteki zawierającej funkcje do dodawania i mnożenia liczb zespolonych.
- Biblioteki powinny się składać z dwóch plików: pliku nagłówkowego oraz pliku z definicjami funkcji.

9.3.8 **(r)** Napisz plik makefile pozwalający na komplikację programu z zadania 9.3.7.

9.3.9 Napisz program, który wczytuje ze standardowego wejścia listę pracowników (imię, nazwisko, wiek), zapisuje ich w tablicy rekordów typu **osoba**, i wypisuje średnią wieku wczytanych pracowników. Program powinien składać się z trzech części:

- programu głównego,
- biblioteki zawierającej definicję typu **osoba** oraz funkcje do wczytywania i wypisywania elementów tego typu,
- biblioteki zawierającej funkcje statystyczne (średnia wieku, minimalny oraz maksymalny wiek) pracujące na tablicach o elementach typu **osoba**. Funkcje zawarte w tej bibliotece powinny otrzymywać w argumentach tablicę elementów typu **osoba** oraz jej rozmiar i zwracać wynik działania jako swoją wartość.

Biblioteki powinny się składać z dwóch plików: pliku nagłówkowego oraz pliku z definicjami funkcji.

9.3.10 Napisz plik makefile pozwalający na komplikację programu z zadania 9.3.9.

9.3.11 **(r, !)** Napisz program, który wczytuje ze standardowego wejścia listę pracowników (imię, nazwisko, wiek), zapisuje ich w tablicy rekordów typu **osoba**, i wypisuje średnią wieku wczytanych pracowników. Program powinien składać się z trzech części:

- programu głównego,
- biblioteki **dane** zawierającej definicję typu **osoba** oraz funkcje do wczytywania i wypisywania elementów tego typu. Wczytane dane osobowe oraz liczba przechowywanych rekordów powinny być przechowywane przez zmienne zadeklarowane w tej bibliotece,
- biblioteki zawierającej funkcje statystyczne (średnia wieku, minimalny oraz maksymalny wiek) pracujące na tablicach o elementach typu **osoba**. Funkcje zawarte w tej bibliotece powinny korzystać z danych przechowywanych przez zmienne zadeklarowane w bibliotece **dane**.

Biblioteki powinny się składać z dwóch plików: pliku nagłówkowego oraz pliku z definicjami funkcji.

9.3.12 **(r)** Napisz plik makefile pozwalający na komplikację programu z zadania 9.3.11.

9.3.13 **(r, !)** Napisz bibliotekę służącą do przechowywania danych osobowych (imię, nazwisko, wiek). Biblioteka powinna zawierać definicję typu **osoba** oraz udostępniać następujące funkcje:

- **wczytaj** – funkcja wczytująca dane jednej osoby ze standardowego wejścia. Dane powinny być wczytywane do tablicy o elementach typu **osoba**.

- **wypisz** – funkcja wypisująca na standardowym wyjściu wszystkie wczytane dane osobowe,
- **ile** – funkcja zwracająca jako wartość liczbę osób, których dane osobowe zostały wczytane.
- **wczytana** – funkcja udostępniająca wczytane dane osobowe. Funkcja dla podanej w argumencie nieujemnej liczby całkowitej **n** powinna zwrócić element typu **osoba** przechowywany pod indeksem **n**.

Dostęp do danych przechowywanych w bibliotece powinien być możliwy wyłącznie za pośrednictwem zdefiniowanych w bibliotece funkcji. Zdefiniowane w bibliotece zmienne powinny być widoczne wyłącznie w tej bibliotece.

9.3.14 Napisz bibliotekę **kalkulator**. Biblioteka ta powinna udostępniać następujące funkcje:

- **wczytaj** – funkcja wczytująca podaną w argumencie wartość do kalkulatora,
- **dodaj** – funkcja dodająca liczbę podaną w argumencie do wartości przechowywanej w kalkulatorze; wynik działania powinien zostać zapisany w kalkulatorze i zwrócony przez funkcję,
- **odejmij** – funkcja odejmująca liczbę podaną w argumencie od wartości przechowywanej w kalkulatorze; wynik działania powinien zostać zapisany w kalkulatorze i zwrócony przez funkcję,
- **pomnoż** – funkcja mnożąca liczbę podaną w argumencie przez wartość przechowywaną w kalkulatorze;; wynik działania powinien zostać zapisany w kalkulatorze i zwrócony przez funkcję,
- **odejmij** – funkcja dzieląca wartość przechowywaną w kalkulatorze przez liczbę podaną w argumencie funkcji; wynik działania powinien zostać zapisany w kalkulatorze i zwrócony przez funkcję.

Zdefiniowane w bibliotece **kalkulator** zmienne powinny być widoczne wyłącznie w tej bibliotece.

9.3.15 (**r**,!**róż**) Napisz bibliotekę **koło** zawierającą:

- jednoargumentową funkcję **pole** zwracającą pole koła o podanym w argumencie promieniu,
- jednoargumentową funkcję **obwód** zwracającą obwód koła o podanym w argumencie promieniu,
- stałą **pi** przechowującą wartość liczby π . Stała ta powinna być dostępna w programach używających biblioteki **koło**.

ROZDZIAŁ 10

ROZWIAZANIA I WSKAZÓWKI

10.1.	Rozwiązania do zadań z rozdziału 1.2	58
10.2.	Rozwiązania do zadań z rozdziału 1.3	62
10.3.	Rozwiązania do zadań z rozdziału 1.4	65
10.4.	Rozwiązania do zadań z rozdziału 2.2	70
10.5.	Rozwiązania do zadań z rozdziału 3.2	77
10.6.	Rozwiązania do zadań z rozdziału 4.2	79
10.7.	Rozwiązania do zadań z rozdziału 5.2	86
10.8.	Rozwiązania do zadań z rozdziału 6.2	97
10.9.	Rozwiązania do zadań z rozdziału 7.2	105
10.10.	Rozwiązania do zadań z rozdziału 7.3	113
10.11.	Rozwiązania do zadań z rozdziału 8.2	129
10.12.	Rozwiązania do zadań z rozdziału 9.2	138
10.13.	Rozwiązania do zadań z rozdziału 9.3	139

10.1. Rozwiązania do zadań z rozdziału 1.2

Zadanie 1.2.1

Listing 10.1. Rozwiązanie zad. 1.2.1 w języku C

```
#include <stdio.h>
2
int main(){
4    int liczba;
    printf("Hello\World");
6    return 0;
}
```

Standardowe wejście i standardowe wyjście to nazwy predefiniowanych w języku C strumieni. To, co one oznaczają, nie zależy od programu napisanego w C, ale od konfiguracji systemu, w którym uruchamiany jest skompilowany program. Zazwyczaj standardowe wejście oznacza klawiaturę, a standardowe wyjście ekran.

Listing 10.2. Rozwiązanie zad. 1.2.1 w języku C++

```
#include <iostream>

int main(){
    int liczba;
    std::cout<<"Hello\World";
    return 0;
}
```

Aby w powyższym rozwiązaniu uniknąć używania przedrostka `std::`, można użyć polecenia `using namespace std;`. Wtedy rozwiązanie wygląda następująco:

Listing 10.3. Rozwiązanie zad. 1.2.1 w języku C++

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hello\World";
    return 0;
}
```

Zadanie 1.2.4

Listing 10.4. Rozwiązanie zad. 1.2.4 w języku C

```
1 #include <stdio.h>
3 int main(){
4     int liczba;
5     scanf("%d", &liczba);
6     printf("%d", liczba);
7     return 0;
8 }
```

W funkcji `scanf` zmienne, do których wczytywane są wartości, są poprzedzone znakiem „&”. Używając funkcji `scanf` należy uważać, aby nie zapominać o wstawianiu „&”, gdyż jest to błąd niewykrywany przez kompilatory, który ujawnia się dopiero podczas działania programu. Wyjątkiem, w którym nie stosuje się znaku „&” przed nazwą wczytywanej zmiennej jest wczytywanie napisów.

Listing 10.5. Rozwiązanie zad. 1.2.4 w języku C++

```
#include <iostream>
using namespace std;

int main(){
    int liczba;
    cin>>liczba;
    cout<<liczba;
    return 0;
}
```

Zadanie 1.2.5

Listing 10.6. Rozwiązanie zad. 1.2.5 w języku C

```
#include <stdio.h>

int main(){
    double liczba;
    scanf("%lf", &liczba);
    printf("%f", liczba);
    return 0;
}
```

Warto zapamiętać, że w funkcjach `scanf` i `printf` innych flag używa się przy wczytywaniu i wypisywaniu wartości typu `double` (odpowiednio „%lf” i „%f”).

Listing 10.7. Rozwiązanie zad. 1.2.5 w języku C++

```
#include <iostream>

int main(){
    double liczba;
    std::cin>>liczba;
    std::cout<<liczba;
    return 0;
}
```

Jak widać powyższe rozwiązanie różni się od rozwiązania zadania 1.2.4 jedynie typem zmiennej `liczba`. Operatory „`<<`” i „`>>`” w pewnym sensie rozpoznają typ zmiennej `liczba` i w zależności od niego wczytują i wypisują wartość zmiennej `liczba` w odpowiedni sposób.

Zadanie 1.2.8

Listing 10.8. Rozwiązanie zad. 1.2.8 w języku C

```
#include <stdio.h>
2
int main(){
4   int l1, l2, l3;
5   scanf("%d", &l1);
6   scanf("%d", &l2);
7   scanf("%d", &l3);
8   printf("%f", (double)(l1 + l2 + l3) / 3);
9   return 0;
10 }
```

Znaczna część studentów na kolokwium linię 8 powyższego programu napisałaby w następujący sposób `printf("%f", (l1+l2+l3)/3);` co byłoby błędem, gdyż dla $l1=0$, $l2=1$ i $l3=1$ na ekranie wyświetlona zostałaby liczba 0, a nie 0.666667. Operatory arytmetyczne w językach C i C++ zwracają wynik takiego samego typu jak argumenty, a gdy argumenty są różnego typu, to typ wyniku jest „najpojemniejszym” spośród typów argumentów. W powyższym programie wartość pierwszego argumentu dzielenia została zrzutowana na typ —`double`—. Dzięki temu argumentami dzielenia nie są dwie wartości typu `int`, a jedna wartość typu `double` i druga wartość typu `int`. Co za tym idzie, wynik dzielenia wykonanego w programie jest typu `double`, a więc może on wyrazić liczbę 0.6(6) (a raczej najbliższa jej wartość możliwa do zapisania w typie `double`). Innym poprawnym, choć mniej eleganckim, rozwiązaniem jest napisanie linii 8 programu 10.8 w następujący sposób `printf("%f", (l1*1.0+l2+l3)/3);`.

Rozwiązań w języku C++ jest analogiczne.

Zadanie 1.2.9

Listing 10.9. Rozwiązań zad. 1.2.9 w języku C

```
1 #include <stdio.h>
2 #include <math.h>

4 int main(){
    double x;
6    scanf("%lf", &x);
    printf("%f", sqrt(x));
8    return 0;
}
```

Aby powyższy program skompilować przy użyciu gcc należy użyć parametru „-lm”. Jest tak, gdyż w programie użyta została funkcja `sqrt` z biblioteki `math`. Podobnie należy postępować także przy komplikacji za pomocą gcc programów używających innych operacji z biblioteki `math`. Problem taki nie występuje w przypadku g++, a więc poniższe rozwiązanie dla C++ kompliuje się poprawnie bez dodatkowego parametru:

Listing 10.10. Rozwiązań zad. 1.2.9 w języku C

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;

5 int main(){
    double x;
7    cin>>x;
    cout<<sqrt(x);
9    return 0;
}
```

Zadanie 1.2.11 Rozwiązań dla języka C:

Listing 10.11. Rozwiązań zad. 1.2.8 w języku C

```
1 #include <stdio.h>
2
3 int main(){
4     float f;
5     scanf("%f", &f);
6     printf("%.2f", f);
8 }
```

Rozwiązanie dla języka C++:

Listing 10.12. Rozwiązanie zad. 1.2.8 w języku C

```

#include<iostream>
using namespace std;

int main(){
    float f;
    cin>>f;
    cout.precision(2);
    cout.setf( ios::fixed, ios::floatfield );
    cout<<fixed<<f;
}

```

10.2. Rozwiązania do zadań z rozdziału 1.3

Zadanie 1.3.1

Listing 10.13. Rozwiązanie zadania 1.3.1 w języku C

```

#include <stdio.h>

int main(){
    int liczba;
    printf("Podaj liczbę całkowitą:");
    scanf("%d", &liczba);
    if (liczba < 0)
        liczba *= -1;
    printf(" | liczba | = %d", liczba);
    return 0;
}

```

To samo zadanie można rozwiązać używając operatora warunkowego:

Listing 10.14. Rozwiązanie zadania 1.3.1 w języku C

```

#include <stdio.h>

int main(){
    int liczba;
    printf("Podaj liczbę całkowitą:");
    scanf("%d", &liczba);
    printf(" | liczba | = %d", (liczba>=0)?liczba:(-1)*liczba);
    return 0;
}

```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 1.3.5

Listing 10.15. Rozwiązanie zadania 1.3.5 w języku C

```
#include <stdio.h>
#include <math.h>

int main(){
    int w;
    double bok1, bok2, bok3, h, p, s;
    printf("Witam. Obliczam pole trojkata. Wpisz:\n");
    printf("1-jesli chcesz podac dl. podstawy i wys.\n");
    printf("2-jesli chcesz podac dl. trzech bokow\n");
    scanf("%d", &w);
    if (w == 1){
        printf("Podaj dlugosc podstawy trojkata.");
        scanf("%lf", &bok1);
        printf("Podaj wysokosc trojkata.");
        scanf("%lf", &h);
        p=bok1*h/2;
    }
    else {
        printf("Podaj dlugosc pierwszego boku trojkata:");
        scanf("%lf", &bok1);
        printf("Podaj dlugosc drugiego boku trojkata:");
        scanf("%lf", &bok2);
        printf("Podaj dlugosc trzeciego boku trojkata:");
        scanf("%lf", &bok3);
        s=(bok1 + bok2 + bok3)/2;
        p=sqrt(s * (s - bok1) * (s - bok2) * (s - bok3));
    }
    printf("Pole trojkata o podanych wymiarach wynosi %f", p);
    return 0;
}
```

Rozwiązanie w języku C++:

Listing 10.16. Rozwiązanie zadania 1.3.5 w języku C++

```
#include <iostream>
#include <cmath>
using namespace std;

int main(){
    int w;
    double bok1, bok2, bok3, h, p, s;
    cout<<"Witam. Obliczam pole trojkata. Wpisz:"<<endl;
    cout<<"1-jesli chcesz podac dl. podstawy i wys."<<endl;
    cout<<"2-jesli chcesz podac dl. i trzech bokow"<<endl;
```

```

cin>>w;
if (w == 1){
    cout<<"Podaj_dlugosc_podstawy_trojkata:_";
    cin>>bok1;
    cout<<"Podaj_wysokosc_trojkata:_";
    cin>>h;
    p=bok1*h/2;
}
else {
    cout<<"Podaj_dlugosc_pierwszego_boku_trojkata:_";
    cin>>bok1;
    cout<<"Podaj_dlugosc_drugiego_boku_trojkata:_";
    cin>>bok2;
    cout<<"Podaj_dlugosc_trzeciego_boku_trojkata:_";
    cin>>bok3;
    s=(bok1 + bok2 + bok3)/2;
    p=sqrt(s * (s - bok1) * (s - bok2) * (s - bok3));
}
cout<<"Pole_trojkata_o_podanych_wymiarach_wynosi_";
return 0;
}

```

Zadanie 1.3.8

Listing 10.17. Rozwiązanie zadania 1.3.8 w języku C

```

#include <stdio.h>

int main(){
    int i;
    double a,b,h,p;
    printf("Pole_jakiej_figury_chcesz_policzyc?\n");
    printf("1_kwadrat\n");
    printf("2_prostokat\n");
    printf("3_trojkat\n");
    scanf("%d",&i);
    switch (i){
        case 1: printf("Podaj_dl_boku_kwadratu");
        scanf("%lf",&a);
        p=a*a;
        break;
        case 2: printf("Podaj_dl_bokow_prostokata");
        scanf("%lf %lf",&a,&b);
        p=a*b;
        break;
        case 3: printf("Podaj_dl_podstawy_i_wys_trojkata");
        scanf("%lf %lf",&a,&h);
        p=0.5*a*h;
    }
    printf("Pole_figury_o_podanych_wymiarach_wynosi_%f\n",p);
    return 0;
}

```

```
}
```

W powyższym programie zamiast instrukcji `switch` można użyć kilku instrukcji `if`, jednak dzięki użyciu instrukcji `switch` program jest bardziej czytelny.

Wersja dla języka C++ jest analogiczna.

10.3. Rozwiązania do zadań z rozdziału 1.4

Zadanie 1.4.1

Listing 10.18. Rozwiązanie zadania 1.4.1 w języku C

```
#include <stdio.h>

int main(){
    int n,m;
    printf("Podaj liczbę całkowitą n: ");
    scanf("%d", &n);
    printf("Podaj liczbę całkowitą m: ");
    scanf("%d", &m);
    for(int i = n ; i < m ; i += n)
        printf("%d\n", i);
    return 0;
}
```

W powyższym programie zmienna `i` została zadeklarowana jednocześnie ze swoją inicjacją w pętli `for`. Taka możliwość w języku C pojawiła się w standardzie C99. W powyższym programie zmiennej `i` nie można używać poza pętlą `for`. W trakcie komplikacji program 10.18 może wymagać dodatkowego parametru mówiącego kompilatorowi, że program jest zgodny ze standardem C99 języka C (w gcc jest to parametr „`-std=c99`”).

Rozwiązanie w języku C++:

Listing 10.19. Rozwiązanie zadania 1.4.1 w języku C++

```
#include <iostream>
using namespace std;

int main(){
    int n,m;
    cout<<"Podaj liczbę całkowitą n: ";
    cin>>n;
    cout<<"Podaj liczbę całkowitą m: ";
    cin>>m;
    for(int i = n ; i < m ; i += n)
        cout<<i<<endl;
```

```
    return 0;
}
```

Zadanie 1.4.4

Listing 10.20. Rozwiązanie zadania 1.4.4 w języku C

```
#include <stdio.h>

int main(){
    int n, i, silnia=1;
    printf("Podaj liczbę całkowitą n:");
    scanf("%d", &n);
    for(i=2; i<=n; i++)
        silnia *= i;
    printf("Silnia z %d wynosi %d\n", n, silnia);
    return 0;
}
```

W programie 10.20 należy zwrócić uwagę na kilka rzeczy:

- Zmienna **silnia** jest inicjowana od razu przy swojej deklaracji.
- Dla $n = 0$ oraz $n = 1$ pętla **for** nie wykona się ani razu i program zwróci początkową wartość zmiennej **silnia**, a więc 1. Jest to oczywiście poprawna odpowiedź dla tych przypadków.
- Warto przyjrzeć się sposobowi wyliczania silni dla $n > 1$. Kluczowa jest tu linia **silnia *= i;**, którą inaczej można zapisać jako **silnia = silnia * i;**. Jest to typowa programistyczna sztuczka. W i -tym obrocie pętli obliczana jest wartość $i!$ (program zapisuje ją do zmiennej **silnia**), korzystając ze wzoru $i! = (i - 1)! * i$, gdzie $(i - 1)!$ to stara wartość zmiennej **silnia**. W ten sposób po n obrotach pętli **for** w zmiennej **silnia** zapisana jest wartość $n!$.

Czytelnik, któremu zaprezentowane powyżej rozwiązanie wydaje się niejasne powinien spróbować rozwiązać kilka kolejnych zadań. Można je rozwiązać w bardzo podobny sposób.

Rozwiązań zadania w języku C++ jest podobne.

Zadanie 1.4.8

Listing 10.21. Rozwiązanie zadania 1.4.8 w języku C

```
#include <stdio.h>

int main(){
    int n, i, fib1=1, fib2=1, pom;
    printf("Podaj liczbę całkowitą n:");
    scanf("%d", &n);
```

```

for( i=2; i<=n; i++){
    pom= fib1;
    fib1 = fib2 + fib1;
    fib2 = pom;
}
printf("Elem. ciagu Fib. o indeksie %d to %d\n", n, fib1);
return 0;
}

```

W powyższym rozwiążaniu założono, że indeks pierwszego elementu ciągu Fibonacciego to 0.

Rozwiążanie w języku C++ jest podobne.

Zadanie 1.4.9 Tym razem przedstawiono rozwiązanie zadania w języku C++. Rozwiążanie w języku C różni się od zaprezentowanego jedynie operacjami wejścia/wyjścia.

Zadanie można rozwiązać poprzez przeszukanie wszystkich liczb dodatnich mniejszych równych zarówno od n , jak i od m :

Listing 10.22. Rozwiążanie zadania 1.4.9 w języku C++

```

#include <iostream>
using namespace std;

int main(){
    int n, m, nwd=1, max;
    cout<<"Podaj liczbę całkowitą n:" ;
    cin>>n;
    cout<<"Podaj liczbę całkowitą m:" ;
    cin>>m;
    max=(n>m)?n:m;
    for (int i=2; i<=max; i++)
        if ((n % i == 0) && (m % i == 0))
            nwd=i;
    cout<<"NWD liczb "<<n<<" i "<<m<<" wynosi "<<nwd<<endl;
    return 0;
}

```

Ponieważ powyższy program przeszukuje zbiór potencjalnych rozwiązań od dołu, to ostatni znaleziony wspólny dzielnik n i m będzie tym największym. W programie użyto operatora „%” zwracającego resztę z dzielenia pierwszego argumentu przez drugi.

Innym sposobem rozwiązania zadania jest zaimplementowanie algorytmu Euklidesa szukania NWD:

Listing 10.23. Rozwiążanie zadania 1.4.9 w języku C++

```
#include <iostream>
using namespace std;

int main(){
    int n, m, pom1, pom2;
    cout<<"Podaj liczbę całkowitą n:" ;
    cin>>n;
    cout<<"Podaj liczbę całkowitą m:" ;
    cin>>m;
    pom1=n;
    pom2=m;
    while(pom1*pom2!=0)
        if (pom1>pom2)
            pom1=pom1%pom2;
        else
            pom2=pom2%pom1;
    cout<<"NWD liczby "<<n<<" i "<<m<<" wynosi " ;
    if (pom1!=0)
        cout<<pom1<<endl;
    else
        cout<<pom2<<endl;
    return 0;
}
```

Drugi z zaprezentowanych programów dla dużych liczb n i m działa znacznie szybciej niż pierwszy.

Zadanie 1.4.10 Także to zadanie można rozwiązać poprzez przeszukiwanie wszystkich potencjalnych rozwiązań.

Listing 10.24. Rozwiązanie zadania 1.4.10 w języku C

```
#include <stdio.h>

int main(){
    int x, i, pierw=0;
    printf("Podaj liczbę całkowitą x:");
    scanf("%d", &x);
    for(i=1; i<=x; i++)
        if (i * i <= x)
            pierw=i;
    printf("Pierwszy z %d to w przybliżeniu %d\n", x, pierw);
    return 0;
}
```

Zadanie 1.4.10 można rozwiązać efektywniej. Jedną z możliwości jest zastosowanie algorytmu bisekcji:

Listing 10.25. Rozwiązanie zadania 1.4.10 w języku C

```
#include <stdio.h>

int main(){
    int x, pocz, kon, sr;
    printf("Podaj liczbę całkowitą x:");
    scanf("%d", &x);
    pocz=0;
    kon=x;
    while (kon - pocz > 1){
        sr=(pocz + kon) / 2;
        if (sr * sr <= x)
            pocz=sr;
        else
            kon=sr;
    }
    printf("Pierwiastek z %d to %w przybliżeniu ", x);
    if (x<=1)
        printf("%d\n", kon);
    else
        printf("%d\n", pocz);
    return 0;
}
```

W powyższym programie w każdym obrocie pętli `while` zbiór potencjalnych rozwiązań zmniejsza się o połowę. Przydaje się tu fakt, iż dzielenie liczb całkowitych daje w wyniku liczbę całkowitą.

W języku C++ to zadanie rozwiązuje się w analogiczny sposób.

Zadanie 1.4.12 To zadanie można rozwiązać za pomocą zagnieżdzonych pętli:

Listing 10.26. Rozwiązanie zadania 1.4.12 w języku C

```
#include <stdio.h>

int main(){
    int n, m, pom1, pom2, suma=0, i;
    printf("Podaj liczbę całkowitą n:");
    scanf("%d", &n);
    for( i = 2 ; i < n ; i++ ){
        pom1=n;
        pom2=i;
        while(pom1*pom2!=0)
            if (pom1<pom2)
                pom2=pom2%pom1;
            else
                pom1=pom1%pom2;
        if (( pom1 == 1 )||( pom2 == 1 ))
```

```

        suma += i;
    }
    printf("Wynik obliczenia to: %d\n", suma);
    return 0;
}

```

W języku C++ to zadanie rozwiązuje się w analogiczny sposób.

Zadanie 1.4.13 Pozornie zadanie wymaga zastosowania pętli w pętli. W rzeczywistości jednak tak nie jest:

Listing 10.27. Rozwiązań zadania 1.4.13 w języku C++

```

#include <iostream>
using namespace std;

int main(){
    int n, silnia=1,suma=1;
    cout<<"Podaj liczbę całkowitą n:";
    cin>>n;
    for(int i=1; i<=n; i++){
        silnia *= i;
        suma+=silnia;
    }
    cout<<"0! + 1! + ... + "<<n<<"! = "<<suma<<endl;
    return 0;
}

```

W języku C to zadanie rozwiązuje się w analogiczny sposób.

10.4. Rozwiązań do zadań z rozdziału 2.2

Zadanie 2.2.1

Listing 10.28. Rozwiązań zadania 2.2.1 w języku C

```

#include <stdio.h>

int bezwzgledna(int liczba){
    if (liczba < 0)
        return liczba*(-1);
    else
        return liczba;
}

int main(){

```

```

int n;
printf("Podaj liczbę całkowitą:");
scanf("%d", &n);
printf("|%d|= %d\n", n, bezwzględna(n));
return 0;
}

```

Rozwiązanie w języku C++ jest podobne.

Zadanie 2.2.2

Listing 10.29. Rozwiązanie zadania 2.2.2 w języku C

```

#include <stdio.h>

int silnia(unsigned int liczba){
    int i, sil=1;
    for(i=2;i <= liczba; i++)
        sil*=i;
    return sil;
}

int main(){
    int n;
    printf("Podaj liczbę całkowitą:");
    scanf("%d", &n);
    printf("silnia z %d = %d\n", n, silnia(n));
    return 0;
}

```

W języku C++ to zadanie rozwiązuje się w analogiczny sposób.

Zadanie 2.2.10

Listing 10.30. Rozwiązanie zadania 2.2.10 w języku C

```

#include <stdio.h>

unsigned int NWD(unsigned int p, unsigned int d){
    while(p != d)
        if (p > d)
            p=p-d;
        else
            d=d-p;
    return p;
}

unsigned int suma(unsigned int n){
    int i, sum=0;

```

```

for( i = 1 ; i < n ; i++ ){
    if ( NWD(i,n) == 1 )
        sum += i;
}
return sum;
}

int main(){
    int n;
    printf("Podaj liczbę całkowitą: ");
    scanf("%d",&n);
    printf("Wynik obliczenia: %d\n" ,suma(n));
    return 0;
}

```

Dzięki używaniu funkcji programy zyskują na czytelności. Ponadto, raz zapelmatowane i przetestowane funkcje mogą być traktowane jak „czarne skrzynki”, do których kodu się nie wraca. Dzięki takiemu podejściu programista może skupiać się naraz na niewielkich fragmentach kodu zawartych w pojedynczych funkcjach. Znacznie upraszcza to pisanie programów. Przykładowo, dzięki wydzieleniu funkcji NWD w programie 10.30 napisanie funkcji suma stało się proste.

W językach C i C++ argumenty funkcji przekazywane są przez wartość. Oznacza to, że wewnątrz funkcji pracuje się na „kopiąch” wartości przekazanych w argumentach. Przykładowo, w programie 10.30, zmiana wartości zmiennych p i d wewnątrz funkcji NWD nie pociąga za sobą zmiany wartości zmiennych i i n z funkcji suma.

Rozwiążanie w języku C++ jest analogiczne.

Zadanie 2.2.13 a)

Listing 10.31. Rozwiązanie zadania 2.2.13 a) w języku C

```

int pierw(unsigned int x){
    int pocz=0, kon=x, sr;
    while (kon - pocz > 1){
        sr=(pocz + kon) / 2;
        if (sr * sr <= x)
            pocz=sr ;
        else
            kon=sr ;
    }
    if (x<=1)
        return kon;
    else
        return pocz;
}

```

```

void wypisz(unsigned int n){
    int i ,p;
    for( i = 1 ; i <=pierw(n) ; i++ ){
        p=pierw(n-i*i);
        if ((p!=0)&&(i*i+p*p==n))
            printf("%d%d%d%d\n",i,i,p,p,n);
    }
}

```

W języku C++ to zadanie rozwiązuje się w analogiczny sposób.

Zadanie 2.2.13 b)

Listing 10.32. Rozwiązanie zadania 2.2.13 b) w języku C

```

int pierw(unsigned int x){
    int pocz=0, kon=x, sr;
    while (kon - pocz > 1){
        sr=(pocz + kon) / 2;
        if (sr * sr <= x)
            pocz=sr ;
        else
            kon=sr ;
    }
    if (x<=1)
        return kon;
    else
        return pocz;
}

void wypisz(unsigned int n){
    int i ,p;
    for( i = 1 ; i <= pierw(n); i++ ){
        p=pierw(n-i*i);
        if ((i*i+p*p==n)&&(i<p))
            printf("%d%d%d%d\n",i,i,p,p,n);
    }
}

```

W tym wariantie zadania, aby nie powtarzać tych samych rozkładów, program sprawdza, czy i jest mniejsze od p .

W języku C++ to zadanie rozwiązuje się w analogiczny sposób.

Zadanie 2.2.17

Listing 10.33. Rozwiązań zadania 2.2.17 w języku C

```
void zlicz () {
    static unsigned int liczba=0;
    liczba++;
    printf("Funkcja zostala wywolana %d razy \n", liczba );
}
```

W powyższym rozwiązań istotne jest słowo kluczowe **static**.

Rozwiązań dla języka C++ jest analogiczne.

Zadanie 2.2.20 Rozwiązań tego zadania dla języków C i C++ niczym się nie różni.

Listing 10.34. Rozwiązań zadania 2.2.20 w języku C/C++

```
unsigned int silnia (unsigned int n) {
    if (n <=1 )
        return 1;
    else
        return silnia (n-1)*n;
}
```

Stosując funkcje rekurencyjne w wielu przypadkach można uzyskać elegancki i prosty zapis algorytmu. Niestety szukanie błędów w funkcjach rekurencyjnych nie należy do rzeczy prostych. Dlatego przy pisaniu takich funkcji trzeba szczególnie uważać. W szczególności trzeba pamiętać o tym, że funkcja musi posiadać **warunek stopu**, czyli warunek przy którym oblicza swoją wartość wprost (już się dalej rekurencyjnie nie wywołuje) oraz o tym, żeby kolejne rekurencyjne wywołania funkcji prowadziły do spełnienia warunku stopu. Kolejnych kilka zadań ma sprawdzić umiejętność implementowania prostych funkcji rekurencyjnych.

Zadanie 2.2.23 Ciąg Fibonacciego wydaje się być stworzony do tego, by jego elementy generować przy pomocy funkcji rekurencyjnej:

Listing 10.35. Rozwiązań zadania 2.2.23 w języku C/C++

```
unsigned int fib (unsigned int n) {
    if (n <=1 )
        return 1;
    else
        return fib (n-1) + fib (n-2);
}
```

Po analizie złożoności obliczeniowej powyższej funkcji okazuje się jednak, że dużo bardziej efektywna jest iteracyjne generowanie elementów ciągu Fibonacciego. Aby się o tym przekonać, wystarczy uruchomić powyższą funkcję oraz program 10.21 dla odpowiednio dużych danych. Słaba efektywność rekurencyjnej implementacji wynika z tego, że wielokrotnie liczy ona wartości tych samych elementów ciągu Fibonacciego.

Zadanie 2.2.26

Listing 10.36. Rozwiążanie zadania 2.2.26 w języku C/C++

```
unsigned int ciag(unsigned int n){  
    if (n <= 2 )  
        return 1;  
    else  
        switch (n%3){  
            case 0: return ciag(n-1) + ciag(n-2);  
            case 1: return 5 * ciag(n-1) + 4;  
            case 2: return ciag(n-1);  
        }  
}
```

W funkcji `ciag` przedstawionej w Listingu 10.36 nie użyto instrukcji `break` w bloku instrukcji `switch`. Wynika to z tego, że instrukcja `return` kończy wykonywanie funkcji (a więc także instrukcji wyboru `switch`) i instrukcja `break` nie jest już potrzebna.

Zadanie 2.2.27

Listing 10.37. Rozwiążanie zadania 2.2.27 w języku C/C++

```
unsigned int f(unsigned int n, unsigned int m){  
    if (n == 0)  
        return m;  
    if (m == 0)  
        return n;  
    return f(n-1,m) + f(n,m-1) + f(n-1, m-1);  
}
```

W funkcji `f` przedstawionej w Listingu 10.37 nie użyto instrukcji `else`. Wynika to z tego, że instrukcja `return` kończy wykonywanie funkcji. Jeżeli więc `return` znajduje się w bloku instrukcji polecenia `if`, to wszystko, co jest po tym bloku instrukcji, zostanie wykonane tylko w przypadku nie spełnienia warunku z `if-a` (a więc tak, jakby po `if` była instrukcja `else`). Pominięcie instrukcji `else` skraca kod funkcji, ale może także zmniejszyć jej czytelność.

Zadanie 2.2.29Listing 10.38. Rozwiązanie zadania 2.2.29 w języku C/C++

```
unsigned int NWD(unsigned int n, unsigned int m){
    if (n == m)
        return m;
    if (n > m)
        return NWD(n%m, m);
    else
        return NWD(m%n, n);
}
```

W powyższej funkcji usunięcie instrukcji `else` nie zmieni działania funkcji.

Zadanie 2.2.30Listing 10.39. Rozwiązanie zadania 2.2.30 w języku C++

```
unsigned int pot(unsigned int n, unsigned int m = 2){
    int p=1;
    if (n == 0)
        return 0;
    for (int i=1; i <= m; i++)
        p*=n;
    return p;
}
```

Zadanie 2.2.36Listing 10.40. Rozwiązanie zadania 2.2.36 w języku C++

```
double pot(double n, unsigned int m){
    double p=1;
    if (n == 0)
        return 0;
    for (int i=1; i <= m; i++)
        p*=n;
    return p;
}

int pot(int n, unsigned int m){
    int p=1;
    if (n == 0)
        return 0;
    for (int i=1; i <= m; i++)
        p*=n;
    return p;
}
```

```
unsigned int pot(unsigned int n, unsigned int m){  
    unsigned int p=1;  
    if (n == 0)  
        return 0;  
    for (int i=1; i <= m; i++)  
        p*=n;  
    return p;  
}
```

10.5. Rozwiązania do zadań z rozdziału 3.2

Zadanie 3.2.1

Listing 10.41. Rozwiązanie zadania 3.2.1 w języku C/C++

```
int mniejsza(int * a, int * b){  
    if (*a<*b)  
        return *a;  
    else  
        return *b;  
}
```

Zadanie to posiada także krótsze rozwiązanie:

Listing 10.42. Rozwiązanie zadania 3.2.1 w języku C/C++

```
int mniejsza(int * a, int * b){  
    return (*a<*b)?*a:*b;  
}
```

Zadanie 3.2.2

Listing 10.43. Rozwiązanie zadania 3.2.2 w języku C/C++

```
int* mniejsza(int * a, int * b){  
    if (*a<*b)  
        return a;  
    else  
        return b;  
}
```

Zadanie 3.2.3

Listing 10.44. Rozwiązanie zadania 3.2.3 w języku C/C++

```
void zamien(int * a, int * b){
```

```

int pom;
pom=*a;
*a=*b;
*b=pom;
}

```

Zadanie 3.2.7Listing 10.45. Rozwiązanie zadania 3.2.7 w języku C++

```

void zamien(int & a, int & b){
    int pom;
    pom=a;
    a=b;
    b=pom;
}

```

Zadanie 3.2.9 Rozwiązanie w języku C:Listing 10.46. Rozwiązanie zadania 3.2.9 w języku C

```

int * alokuj(){
    return malloc(sizeof(int));
}

```

Używając funkcji `malloc` należy pamiętać o dołączenia pliku nagłówkowego `<stdlib.h>`. Jeżeli się tego nie zrobi, kompilator zgłosi ostrzeżenie o próbie przypisania wartości całkowitoliczbowej do zmiennej wskaźnikowej bez rzutowania.

Rozwiązanie w języku C++:

Listing 10.47. Rozwiązanie zadania 3.2.9 w języku C

```

int * alokuj(){
    return new int;
}

```

Zadanie 3.2.11 Rozwiązanie w języku C:Listing 10.48. Rozwiązanie zadania 3.2.11 w języku C

```

int * alokuj(unsigned int n){
    return malloc(n*sizeof(int));
}

```

Rozwiązanie w języku C++:

Listing 10.49. Rozwiązanie zadania 3.2.11 w języku C

```
int * alokuj(unsigned int n){  
    return new int[n];  
}
```

W obu językach alokowanie w pamięci jednowymiarowych tablic dynamicznych odbywa się w identyczny sposób jak w powyższych rozwiązaniach.

Zadanie 3.2.13

Listing 10.50. Rozwiązanie zadania 3.2.13 w języku C/C++

```
double wywolaj(double (* fun)(int arg), int a){  
    return fun(a);  
}
```

Zadanie 3.2.15

Listing 10.51. Rozwiązanie zadania 3.2.15 w języku C/C++

```
void przepisz(int const * a, int * b){  
    *b=*a;  
}
```

Zadanie 3.2.16

Listing 10.52. Rozwiązanie zadania 3.2.15 w języku C/C++

```
void przepisz(int const * a, int * const b){  
    *b=*a;  
}
```

10.6. Rozwiązania do zadań z rozdziału 4.2

Zadanie 4.2.1a

Listing 10.53. Rozwiązanie zadania 4.2.1a w języku C/C++

```
void zeruj (unsigned int n, int * tab){  
    int i;  
    for(i=0;i<n;i++)  
        tab[i]=0;  
}
```

Warto zapamiętać, że w przypadku tablic przekazanych do funkcji w jej argumentach, wewnątrz funkcji pracujemy na „oryginałach” tablic, a nie ich kopiąch. Oznacza to, że zmiana wewnątrz funkcji wartości elementów takich tablic ma konsekwencje także na zewnątrz funkcji. Jest to zachowanie odmienne od argumentów funkcji o typach prostych, w przypadku których pracujemy na kopiąch argumentów. Takie a nie inne zachowanie tablic wynika z faktu, że w C i C++ są one wskaźnikami na bloki pamięci. Skopiowany wskaźnik cały czas wskazuje na ten sam blok pamięci.

Zadanie 4.2.1b

Listing 10.54. Rozwiązań zadania 4.2.1b w języku C/C++

```
void inicjuj (unsigned int n, int * tab){
    int i;
    for(i=0;i<n;i++)
        tab[ i ]=i;
}
```

Zadanie 4.2.2a

Listing 10.55. Rozwiązań zadania 4.2.2a w języku C/C++

```
double srednia (unsigned int n, int * tab){
    int i;
    double sred=0;
    for(i=0;i<n;i++)
        sred+=tab[ i ];
    sred/=n;
    return sred;
}
```

Zastosowane powyżej rozwiązanie jest rozwinięciem metody zastosowanej w rozwiązań zadania 1.4.4.

Zadanie 4.2.3

Listing 10.56. Rozwiązań zadania 4.2.3 w języku C/C++

```
double srednia (unsigned int n,const int * tab){
    int i;
    double sred=0;
    for(i=0;i<n;i++)
        sred+=tab[ i ];
    sred/=n;
    return sred;
}
```

Zadanie 4.2.5

Listing 10.57. Rozwiązanie zadania 4.2.5 w języku C

```
int pierwsza (unsigned int n){
    int i, j, pom;
    bool sito[n];
    for(i=0;i<n;i++)
        sito[i]=true;
    for(i=2;i<n;i++)
        if (sito[i]){
            pom=i;
            for(j=2*i;j<n;j+=i)
                sito[j]=false;
        }
    return pom;
}
```

W pierwszych standardach języka C nie było typu `bool`. Został on wprowadzony dopiero w standardzie C99. Aby móc go użyć, należy dołączyć plik nagłówkowy `stdbool.h`.

Rozwiążanie w języku C++ nie różni się bardzo od powyższego. Istnieje jedna istotna różnica, która uzasadnia prezentację rozwiązań dla obu języków.

Listing 10.58. Rozwiązanie zadania 4.2.5 w języku C++

```
int pierwsza (unsigned int n){
    int pom;
    bool * sito = new bool[n];
    for(int i=0;i<n;i++)
        sito[i]=true;
    for(int i=2;i<n;i++)
        if (sito[i]){
            pom=i;
            for(int j=2*i;j<n;j+=i)
                sito[j]=false;
        }
    delete [] sito;
    return pom;
}
```

Język C++, inaczej niż język C, nie pozwala na tworzenie tablic automatycznych o rozmiarze zadanym przez zmienną. Stąd w tym przypadku pamięć dla tablicy `sito` ręcznie rezerwowana operatorem `new` i ręcznie zwalniana przy użyciu operatora `delete`. Usuwając tablicę należy dodać „`[]`” po operatorze `delete`, a przed wskaźnikiem na tablicę. W języku C++ zamiast

tablicy można użyć szablonu klasy `vector`, jednak w niniejszym zbiorze zadań przedstawiona jest tylko strukturalna części C++.

Także w rozwiązyaniu w języku C można ręcznie zarezerwować pamięć dla tablicy `sito`:

Listing 10.59. Rozwiązań zadania 4.2.5 w języku C

```
int pierwsza (unsigned int n){
    int i, j, pom;
    bool * sito=malloc(n*sizeof(bool));
    for(i=0;i<n; i++)
        sito[ i]=true;
    for(i=2;i<n; i++)
        if (sito[ i]){
            pom=i ;
            for(j=2*i ;j<n; j+=i )
                sito[ j]=false ;
        }
    free( sito );
    return pom;
}
```

Jak można zauważyć w Listingu 10.59 w języku C, inaczej niż ma to miejsce w C++, pamięć zajmowaną przez tablicę zwalnia się dokładnie w taki sam sposób, jak w przypadku typów prostych.

Zadanie 4.2.6a

Listing 10.60. Rozwiązań zadania 4.2.6a w języku C/C++

```
void przepisz (unsigned int n, int * tab1 , int * tab2){
    int i;
    for(i=0;i<n; i++)
        tab2[ i]=tab1[ i ];
}
```

Zadanie 4.2.10a

Listing 10.61. Rozwiązań zadania 4.2.10a w języku C/C++

```
int maksimum(unsigned int n, int * tab){
    int i, max=tab[ 0 ];
    for(i=1;i<n; i++)
        if (tab[ i]>max)
            max=tab[ i ];
    return max;
}
```

Zadanie 4.2.10c

Listing 10.62. Rozwiązanie zadania 4.2.10c w języku C/C++

```
int maksimum(unsigned int n, int * tab){
    int i, max=0;
    for(i=1;i<n;i++)
        if (tab[i]>tab[max])
            max=i ;
    return max;
}
```

Zadanie 4.2.12a

Listing 10.63. Rozwiązanie zadania 4.2.12a w języku C/C++

```
void odwroc (unsigned int n, int * tab){
    int i ,pom;
    for(i=0;i<n/2;i++){
        pom=tab[i];
        tab[i]=tab[n-1-i];
        tab[n-1-i]=pom;
    }
}
```

W powyższym rozwiązaniu ważne jest, że zmienna i przebiega wartości mniejsze od $n/2$. Gdyby i przebiegało wartości od 0 do $n-1$, wtedy kolejność elementów w tablicy zostałaby odwrócona dwukrotnie, a co za tym idzie, po zakończeniu działania funkcji `odwroc` kolejność elementów tablicy `tab` pozostałaby niezmieniona.

Zadanie 4.2.12b

Listing 10.64. Rozwiązanie zadania 4.2.12b w języku C/C++

```
void przesun(unsigned int n, int * tab){
    int i ,pom=tab[0];
    for(i=0;i<n-1;i++)
        tab[i]=tab[i+1];
    tab[n-1]=pom;
}
```

Kolejność, w jakiej zmienna i przebiega wartości ze zbioru $\{0, \dots, n\}$, nie jest obojętna, o czym można się przekonać porównując powyższe rozwiązanie z rozwiązaniem zadania 4.2.12c

Zadanie 4.2.12c

Listing 10.65. Rozwiązanie zadania 4.2.12c w języku C/C++

```
void przesun(unsigned int n, int * tab){
    int i ,pom=tab[n-1];
    for(i=n-2;i>=0;i--)
        tab[i+1]=tab[i];
    tab[0]=pom;
}
```

Gdyby w powyższym programie zmienna *i* przebiegała wartości od 0 do $n - 2$, tak jak ma to miejsce w programie 10.64, to w efekcie do wszystkich komórek tablicy *tab*, za wyjątkiem komórki *tab[0]*, zostałaby wstawiona pierwotna wartość komórki *tab[0]*.

Zadanie 4.2.12d Istnieje wiele algorytmów sortujących tablice. Poniżej zaimplementowany został jeden z prostszych algorytmów. Jego idea jest następująca: wyszukać największy element tablicy i zamienić go miejscami z ostatnim, po czym znaleźć największy element spośród pierwszych $n - 1$ elementów i zamienić go miejscami z elementem o indeksie $n - 1$ etc.

Listing 10.66. Rozwiązanie zadania 4.2.12d w języku C/C++

```
int maksimum(unsigned int n, int * tab){
    int i , max=0;
    for(i=1;i<n;i++)
        if (tab[i]>tab[max])
            max=i ;
    return max;
}

void sortuj (unsigned int n, int * tab){
    int i ,j ,pom;
    for(i=0;i<n-1;i++){
        j=maksimum(n-i ,tab);
        pom=tab[n-i -1];
        tab[n-i -1]=tab[j];
        tab[j]=pom;
    }
}
```

Co prawda niniejszy zbiór zadań nie obejmuje zagadnień z zakresu algorytmiki, ale mimo to, jako ciekawostkę, przedstawiono poniżej implementację algorytmu sortowania przez scalanie. Dla dużych tablic algorytm ten działa znacznie szybciej od tego przedstawionego w programie 10.66

Listing 10.67. Rozwiązanie zadania 4.2.12d w języku C/C++

```
void merge(int* tablica ,int start ,int srodek ,int koniec){
    int tab_pom[koniec-start];
    int i,j,k;
    i = start;
    k = 0;
    j = srodek + 1;

    while ((i <= srodek)&&(j <= koniec)){
        if (tablica[j] < tablica[i]){
            tab_pom[k] = tablica[j];
            j = j + 1;
        }
        else{
            tab_pom[k] = tablica[i];
            i = i + 1;
        }
        k = k + 1;
    }

    if (i <= srodek)
        while (i <= srodek){
            tab_pom[k] = tablica[i];
            i = i + 1;
            k = k + 1;
        }
    else
        while (j <= koniec){
            tab_pom[k] = tablica[j];
            j = j + 1;
            k = k + 1;
        }
    for(i= 0;i<=koniec-start;i++)
        tablica[start + i]= tab_pom[i];
}

void merge_sort(int * tablica , int start , int koniec){
    int srodek;
    if (start != koniec){
        srodek = (start + koniec) / 2;
        merge_sort(tablica , start , srodek);
        merge_sort(tablica , srodek + 1 , koniec);
        merge(tablica , start , srodek , koniec);
    }
}

void sortuj(unsigned int n, int * tab){
    merge_sort(tab ,0 ,n-1);
}
```

Zadanie 4.2.13

Listing 10.68. Rozwiązanie zadania 4.2.13 w języku C

```
int * alokuj (unsigned int n){
    return malloc(n*sizeof(int));
}
```

Należy pamiętać, że w takiej sytuacji nie należy zwracać wskaźnika do tablicy utworzonej automatycznie, gdyż przestaje ona istnieć w momencie zakończenia działania funkcji. Poniżej rozwiązanie dla języka C++

Listing 10.69. Rozwiązanie zadania 4.2.13 w języku C++

```
int * alokuj (unsigned int n){
    return new int[n];
}
```

Zadanie 4.2.15

Listing 10.70. Rozwiązanie zadania 4.2.15 w języku C

```
void zwolnij (int *tab){
    free(tab);
}
```

To samo w języku C++:

Listing 10.71. Rozwiązanie zadania 4.2.15 w języku C++

```
void zwolnij (int * tab){
    delete [] tab;
}
```

10.7. Rozwiązań do zadań z rozdziału 5.2

Zadanie 5.2.1 Najpierw wersja dla typu `char`:

Listing 10.72. Rozwiązanie zadania 5.2.1 w języku C/C++

```
int wyczysc(char *nap){
    nap[0]=0;
}
```

Wersja dla typu `wchar_t` różni się niewiele:

Listing 10.73. Rozwiążanie zadania 5.2.1 w języku C/C++

```
int wyczysc(wchar_t *nap){
    nap[0]=0;
}
```

Zadanie 5.2.2 Najpierw wersja dla typu `char`:

Listing 10.74. Rozwiążanie zadania 5.2.2 w języku C/C++

```
int dlugosc(char *nap){
    int i=0;
    while(nap[i]!=0)
        i++;
    return i;
}
```

Wersja dla typu `wchar_t` różni się niewiele:

Listing 10.75. Rozwiążanie zadania 5.2.2 w języku C/C++

```
int dlugosc(wchar_t *nap){
    int i=0;
    while(nap[i]!=0)
        i++;
    return i;
}
```

Zadanie 5.2.4

Listing 10.76. Rozwiążanie zadania 5.2.4 w języku C/C++

```
int porownaj(char *nap1, char * nap2){
    int i;
    for(i=0;(nap1[i]!=0)&&(nap2[i]!=0);i++)
        if (nap1[i]!=nap2[i])
            return (nap1[i]<nap2[i])?1:0;
    if (nap1[i]==0)
        return 0;
    else
        return 1;
}
```

W powyższym rozwiążaniu wykorzystany został fakt, że w tablicy kodów ASCII małe łacińskie litery są ułożone zgodnie z porządkiem alfabetycznym

Zadanie 5.2.7

Listing 10.77. Rozwiązanie zadania 5.2.7 w języku C/C++

```
void sklej(char *nap1, char * nap2, char * nap3){
    int i, j;
    for(i=0;nap1[i]!=0; i++)
        nap3[i]=nap1[i];
    for(j=0;nap2[j]!=0; i++,j++)
        nap3[i]=nap2[j];
    nap3[i]=0;
}
```

Wersja dla napisów o znakach typu `wchar_t` jest podobna.

Zadanie 5.2.8

Listing 10.78. Rozwiązanie zadania 5.2.8 w języku C/C++

```
void maleduze(char *nap){
    int i;
    for(i=0;nap[i]!=0; i++)
        if ((nap[i]>='a')&&(nap[i]<='z'))
            nap[i]-=(‘a’-‘A’);
}
```

Zadanie 5.2.9

Listing 10.79. Rozwiązanie zadania 5.2.9 w języku C/C++

```
void wytnij(char *nap, int n, int m){
    int i, dl;
    for(dl=0;nap[dl]!=0; dl++);
    if (dl+1>m){
        for(i=0;i+m<dl; i++)
            nap[n+i]=nap[i+m+1];
    }
    else
        if ((n<dl)&&(dl+1<=m) )
            nap[n]=0;
}
```

Wersja dla napisów o znakach typu `wchar_t` jest podobna.

Zadanie 5.2.10

Listing 10.80. Rozwiązanie zadania 5.2.10 w języku C/C++

```

bool porownaj(char *nap1, char* nap2, int n){
    int i;
    for(i=0;(nap1[ i ]!=0)&&(nap2[ i ]!=0) ;i++)
        if (nap1[ n+i ]!=nap2[ i ])
            return false;
    if (nap2[ i ]==0)
        return true;
    else
        return false;
}

void wytnij2(char *nap1, char* nap2){
    int i ,dl;
    for(dl=0;nap2[ dl ]!=0;dl++);
    for(i=0;nap1[ i ]!=0; i++)
        if (porownaj(nap1,nap2,i)){
            wytnij(nap1,i,i+dl-1);
            return;
        }
}

```

Użyta w powyższym rozwiązaniu funkcja `wytnij`, to funkcja z rozwiązania zadnia 5.2.9.

Wersja dla napisów o znakach typu `wchar_t` jest podobna.

Zadanie 5.2.11

Listing 10.81. Rozwiązanie zadania 5.2.11 w języku C/C++

```

void wytnijzw(char *nap1, char* nap2){
    int i ,dl;
    int wyst[256]={};
    for(i=0;nap2[ i ]!=0; i++)
        wyst[nap2[ i ]]=1;
    for(i=0,j=0;nap1[ i ]!=0; i++)
        if (wyst[nap1[ i ]]==0){
            if (j<i)
                nap1[ j ]=nap1[ i ];
            j++;
        }
        nap1[ j ]=0;
}

```

Wersja dla typu `wchar_t` jest trochę inna ze względu na fakt, że tablica `wyst` musiałaby w tym przypadku być bardzo duża:

Listing 10.82. Rozwiązanie zadania 5.2.11 w języku C/C++

```

bool czywyst(wchar_t z, wchar_t * nap){
    int i;
    for(i=0;nap[i]!=0; i++)
        if (nap[i]==z)
            return true;
    return false;
}

void wytnijzw(wchar_t *nap1, wchar_t * nap2){
    int i,j;
    for(i=0,j=0;nap1[i]!=0; i++){
        if (!czywyst(nap1[i],nap2)){
            if (j<i)
                nap1[j]=nap1[i];
            j++;
        }
        nap1[j]=0;
    }
}

```

Zadanie 5.2.13

Listing 10.83. Rozwiązanie zadania 5.2.13 w języku C/C++

```

void wytnijtm(wchar_t *nap1, wchar_t * nap2){
    int i,j;
    for(i=0,j=0;nap1[i]!=0; i++)
        if (nap1[i]!=nap2[i]){
            if (j<i)
                nap1[j]=nap1[i];
            j++;
        }
        nap1[j]=0;
}

```

Wersja dla napisów o znakach typu `char` jest podobna.

Zadanie 5.2.14 Wersja dla napisów o znakach typu `char`:

Listing 10.84. Rozwiązanie zadania 5.2.14 w języku C/C++

```

void wypisz(char* nap){
    printf("%"s",nap);
}

```

Wersja dla napisów o znakach typu `wchar_t`:

Listing 10.85. Rozwiążanie zadania 5.2.14 w języku C/C++

```
void wypisz(wchar_t* nap){
    wprintf(L"%ls", nap);
}
```

Do wyświetlania napisów o znakach typu `wchar_t` można użyć także funkcji `printf`:

Listing 10.86. Rozwiążanie zadania 5.2.14 w języku C/C++

```
void wypisz(wchar_t* nap){
    printf("%ls", nap);
}
```

Rozwiązańia w języku C++:

Listing 10.87. Rozwiążanie zadania 5.2.14 w języku C++

```
void wypisz(char* nap){
    cout<<nap;
}
```

Wersja dla typu `wchar_t` w języku C++:

Listing 10.88. Rozwiążanie zadania 5.2.14 w języku C++

```
void wypisz(wchar_t* nap){
    wcout<<nap;
}
```

W przeciwieństwie do języka C, gdzie funkcji `printf` można używać do wypisywania napisów o znakach typu `wchar_t`, w języku C++, aby operować na takich napisach, trzeba używać strumienia `wcout` zamiast `cout`.

Zadanie 5.2.15 Wersja dla napisów typu `string`:

Listing 10.89. Rozwiążanie zadania 5.2.15 w języku C++

```
void wypisz(string s){
    cout<<s;
}
```

Wersja dla napisów typu `wstring`:

Listing 10.90. Rozwiązanie zadania 5.2.15 w języku C++

```
void wypisz(wstring s){
    wcout<<s;
}
```

Zadanie 5.2.16 Wersja dla napisów o znakach typu `char`:

Listing 10.91. Rozwiązanie zadania 5.2.16 w języku C/C++

```
void wczytaj(char* nap){
    scanf("%s", nap);
}
```

Wersja dla napisów o znakach typu `wchar_t`:

Listing 10.92. Rozwiązanie zadania 5.2.16 w języku C/C++

```
void wczytaj(wchar_t* nap){
    wsscanf(L"%ls", nap);
}
```

Do wczytywania napisów o znakach typu `wchar_t` można użyć także funkcji `scanf`:

Listing 10.93. Rozwiązanie zadania 5.2.16 w języku C/C++

```
void wczytaj(wchar_t* nap){
    scanf("%ls", nap);
}
```

Przy wczytywaniu napisów przy użyciu funkcji `scanf` nie pisze się znaku „`&`” przed zmienną, do której wczytujemy wartość. Jest tak dlatego, że zmienne tablicowe są wskaźnikami na bloki pamięci zawierające właściwą tablicę.

Inną rzeczą, o której należy pamiętać, jest fakt, że wczytując napis do tablicy należy zadbać o to, żeby tablica była wystarczającego rozmiaru. W praktyce jedynym sposobem na zapewnienie tego jest dodanie parametru przed specyfikatorami formatu `s` i `ls` w funkcjach `scanf` i `wsscanf`, który ograniczy rozmiar wczytywanych napisów.

Rozwiązań w języku C++:

Listing 10.94. Rozwiązanie zadania 5.2.16 w języku C++

```
void wczytaj(char* nap){
    cin>>nap;
}
```

Wersja dla typu `wchar_t` w języku C++:

Listing 10.95. Rozwiązanie zadania 5.2.16 w języku C++

```
void wczytaj(wchar_t* nap){
    wcin>>nap;
}
```

W przeciwieństwie do języka C, gdzie funkcji `scanf` można używać do wczytywania napisów o znakach typu `wchar_t`, w języku C++, aby operować na takich napisach, trzeba używać strumienia `wcin` zamiast `cin`.

Zadanie 5.2.17 Wersja dla napisów typu `string`:

Listing 10.96. Rozwiązanie zadania 5.2.17 w języku C++

```
void wczytaj(string& s){
    cin>>s;
}
```

Wersja dla napisów typu `wstring`:

Listing 10.97. Rozwiązanie zadania 5.2.16 w języku C++

```
void wczytaj(wstring& s){
    wcin>>s;
}
```

Zadanie 5.2.18 Wersja dla napisów o znakach typu `char`:

Listing 10.98. Rozwiązanie zadania 5.2.18 w języku C

```
char * pierwszy(char** tnap, int n){
    int i, min=0;
    char * wyn;
    for(i=1;i<n;i++)
        if (stromo(tnap[min],tnap[i])>0)
            min=i ;
    wyn=malloc ((strlen(tnap[min])+1)*sizeof(char));
    strcpy(wyn,tnap[min]);
    return wyn;
}
```

Wersja dla napisów o znakach typu `wchar_t`:

Listing 10.99. Rozwiązanie zadania 5.2.18 w języku C

```
wchar_t * pierwszy(wchar_t** tnap, int n){
    int i, min=0;
```

```
wchar_t * wyn;
for(i=1;i<n;i++)
    if (wcscmp(tnap[min],tnap[i])>0)
        min=i;
wyn=malloc((wcslen(tnap[min])+1)*sizeof(wchar_t));
wcscpy(wyn,tnap[min]);
return wyn;
}
```

Rozwiązanie w języku C++ jest analogiczne. Wystarczy skorzystać z faktu, że funkcje biblioteczne języka C są dostępne również w C++.

Zadanie 5.2.19 Wersja dla napisów typu `string`:

Listing 10.100. Rozwiązanie zadania 5.2.19 w języku C

```
string pierwszy(string* nap,int n){
    int i, min=0;
    for(i=1;i<n;i++)
        if (nap[min]>nap[i])
            min=i;
    return nap[min];
}
```

Wersja dla napisów typu `wstring` jest niemal identyczna.

Zadanie 5.2.20 Wersja dla napisów o znakach typu `char`:

Listing 10.101. Rozwiązanie zadania 5.2.20 w języku C

```
char * godzina(int godz, int min, int sek){
    char * wyn=malloc(9*sizeof(char));
    sprintf(wyn,"%02d:%02d:%02d",godz,min,sek);
    return wyn;
}
```

Wersja dla napisów o znakach typu `wchar_t`:

Listing 10.102. Rozwiązanie zadania 5.2.20 w języku C

```
wchar_t * godzina(int godz, int min, int sek){
    wchar_t * wyn=malloc(9*sizeof(wchar_t));
    swprintf(wyn,9,L"%02d:%02d:%02d",godz,min,sek);
    return wyn;
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 5.2.21 Wersja zwracająca napis o typie `string`:

Listing 10.103. Rozwiązanie zadania 5.2.21 w języku C++

```
string godzina(int godz, int min, int sek){
    stringstream wyn;
    wyn<<((godz<10)?"0":"")<<godz;
    wyn<<": "<<((min<10)?"0":"")<<min;
    wyn<<": "<<((sek<10)?"0":"")<<sek;
    return wyn.str();
}
```

Wersja zwracająca napis o typie `wstring`:

Listing 10.104. Rozwiązanie zadania 5.2.21 w języku C++

```
wstring godzina(int godz, int min, int sek){
    wstringstream wyn;
    wyn<<((godz<10)?L"0":L"")<<godz;
    wyn<<L": "<<((min<10)?L"0":L"")<<min;
    wyn<<L": "<<((sek<10)?L"0":L"")<<sek;
    return wyn.str();
}
```

Zadanie 5.2.22 Wersja dla napisów o znakach typu `char`:

Listing 10.105. Rozwiązanie zadania 5.2.22 w języku C

```
char * sklej(char * nap1, char * nap2, char * nap3){
    char * wyn=malloc((strlen(nap1)+strlen(nap2)
                       +strlen(nap3)+1)*sizeof(char));
    strcpy(wyn,nap1);
    strcat(wyn,nap2);
    strcat(wyn,nap3);
    return wyn;
}
```

Wersja dla napisów o znakach typu `wchar_t`:

Listing 10.106. Rozwiązanie zadania 5.2.22 w języku C

```
wchar_t * sklej(wchar_t * nap1, wchar_t * nap2, wchar_t * nap3){
    wchar_t * wyn=malloc((wcslen(nap1)+wcslen(nap2)
                          +wcslen(nap3)+1)*sizeof(wchar_t));
    wcscpy(wyn,nap1);
    wcscat(wyn,nap2);
    wcscat(wyn,nap3);
    return wyn;
```

 }

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 5.2.23 Wersja zwracająca napis o typie `string`:

Listing 10.107. Rozwiązanie zadania 5.2.23 w języku C++

```
string sklej(string nap1, string nap2, string nap3){
    return nap1+nap2+nap3;
}
```

Rozwiązanie dla napisów typu `wstring` jest podobne.

Zadanie 5.2.26 Wersja dla napisów o znakach typu `char`:

Listing 10.108. Rozwiązanie zadania 5.2.26 w języku C/C++

```
void maleduze(char * nap){
    int i;
    for(i=0;nap[i]!=0;i++)
        nap[i]=toupper(nap[i]);
}
```

Wersja dla napisów o znakach typu `wchar_t`: jest bardzo podobna:

Listing 10.109. Rozwiązanie zadania 5.2.26 w języku C/C++

```
void maleduze(wchar_t * nap){
    int i;
    for(i=0;nap[i]!=0;i++)
        nap[i]=towupper(nap[i]);
}
```

Zadanie 5.2.27 Wersja dla napisów typu `string`:

Listing 10.110. Rozwiązanie zadania 5.2.27 w języku C++

```
void maleduze(string& nap){
    for(int i=0;i<nap.length();i++)
        nap[i]=toupper(nap[i]);
}
```

Wersja dla napisów typu `wstring` jest bardzo podobna.

Listing 10.111. Rozwiązanie zadania 5.2.27 w języku C++

```
void maleduze(wstring & nap){
```

```
for (int i=0;i<nap.length();i++)
    nap[i]=towupper(nap[i]);
}
```

10.8. Rozwiązania do zadań z rozdziału 6.2

Zadanie 6.2.1

Listing 10.112. Rozwiązanie zadania 6.2.1 w języku C

```
int** alokuj(unsigned int n, unsigned int m){
    int ** t=malloc(n*sizeof(int *));
    int i;
    for(i=0;i<n;i++)
        t[i]=malloc(m*sizeof(int));
    return t;
}
```

W języku C++ rozwiązanie powyższego zadania wygląda następująco:

Listing 10.113. Rozwiązanie zadania 6.2.1 w języku C++

```
int** alokuj(unsigned int n, unsigned int m){
    int ** t= new int*[n];
    for(int i=0;i<n;i++)
        t[i]= new int[m];
    return t;
}
```

Zadanie 6.2.2

Listing 10.114. Rozwiązanie zadania 6.2.2 w języku C

```
int (* alokuj(unsigned int n, unsigned int m))[]{
    return malloc(n*sizeof(int [m]));
}
```

W języku C++ nie da się wprost rozwiązać powyższego zadania ze względu na brak możliwości utworzenia w nim wielowymiarowych, w pełni dynamicznych tablic, takich jak te w programie 10.114. Ten problem można rozwiązać tworząc klasę zawierającą jednowymiarową tablicę dynamiczną i przeciążając odpowiednie operatory. Nie mieści się to jednak w zakresie materiału obejmowanego przez niniejszy skrypt.

Zadanie 6.2.3

Listing 10.115. Rozwiązanie zadania 6.2.3 w języku C

```
void zwolnij(unsigned int n, unsigned int m, int ** t){
    int i;
    for(i=0;i<n; i++)
        free(t[i]);
    free(t);
}
```

W języku C++ rozwiązanie powyższego zadania wygląda następująco:

Listing 10.116. Rozwiązanie zadania 6.2.3 w języku C++

```
void zwolnij(unsigned int n, unsigned int m, int ** t){
    for(int i=0;i<n; i++)
        delete [] t[i];
    delete [] t;
}
```

Zadanie 6.2.4

Listing 10.117. Rozwiązanie zadania 6.2.4 w języku C

```
void zwolnij(unsigned int n, int t[ ][n]){
    free(t);
}
```

Zadanie 6.2.7

Listing 10.118. Rozwiązanie zadania 6.2.7 w języku C

```
int** alokuj(unsigned int n){
    int ** t=malloc(n*sizeof(int*));
    int i ;
    for(i=0;i<n; i++)
        t[i]=malloc((i+1)*sizeof(int));
    return t ;
}
```

W języku C++ rozwiązanie powyższego zadania wygląda następująco:

Listing 10.119. Rozwiązanie zadania 6.2.7 w języku C++

```
int** alokuj(unsigned int n){
    int ** t= new int *[n];
    for(int i=0;i<n; i++)
        t[i]= new int [i+1];
    return t ;
}
```

 }

Zadanie 6.2.8

Listing 10.120. Rozwiązanie zadania 6.2.8 w języku C/C++

```
void zeruj(int t [][100], unsigned int n){
    int i ,j ;
    for(i=0;i<n ;i++)
        for(j=0;j<100;j++)
            t [ i ][ j ]=0;
}
```

Zadanie 6.2.9

Listing 10.121. Rozwiązanie zadania 6.2.9 w języku C/C++

```
void zeruj(int** t , unsigned int n, unsigned int m){
    int i ,j ;
    for(i=0;i<n ;i++)
        for(j=0;j<m; j++)
            t [ i ][ j ]=0;
}
```

Zadanie 6.2.10

Listing 10.122. Rozwiązanie zadania 6.2.10 w języku C

```
void zeruj(unsigned int n, unsigned int m,int t [][m]){
    int i ,j ;
    for(i=0;i<n ;i++)
        for(j=0;j<m; j++)
            t [ i ][ j ]=0;
}
```

W powyższym rozwiążaniu ważne jest, żeby deklaracja argumentu **m** wystąpiła przed deklaracją argumentu **t** (innym przypadku kompilator zgłosi błąd użycia niezadeklarowanej zmiennej **m** w deklaracji **t**).

Zadanie 6.2.14

Listing 10.123. Rozwiązanie zadania 6.2.14 w języku C/C++

```
int suma(int t [][100][100]){
    int i ,j ,k,sum=0;
    for(i=0;i<100;i++)
        for(j=0;j<100;j++)
            for(k=0;k<100;k++)
```

```
    sum+=t[i][j][k];
  return sum;
}
```

Zadanie 6.2.16

Listing 10.124. Rozwiązanie zadania 6.2.16 w języku C/C++

```
int max_sred(int **t, unsigned int n, unsigned int m){
  int i,j,sum,max;
  double wart;
  for(i=0;i<n;i++){
    sum=0;
    for(j=0;j<m;j++)
      sum+=t[i][j];
    if (((double)sum/m>wart) || (i==0)){
      max=i;
      wart=(double)sum/m;
    }
  }
  return max;
}
```

Zadanie 6.2.17

Listing 10.125. Rozwiązanie zadania 6.2.17 w języku C/C++

```
double max_sred(int **t, unsigned int n, unsigned int m){
  int i,j,sum;
  double wart;
  for(i=0;i<n;i++){
    sum=0;
    for(j=0;j<m;j++)
      sum+=t[i][j];
    if (((double)sum/m>wart) || (i==0))
      wart=(double)sum/m;
  }
  return wart;
}
```

Zadanie 6.2.19

Listing 10.126. Rozwiązanie zadania 6.2.19 w języku C/C++

```
void przepisz(int **t1, int **t2, unsigned int n,
                unsigned int m){
  int i,j;
  for(i=0;i<n;i++)
    for(j=0;j<m;j++)
      t2[i][j]=t1[i][j];
```

 }

Zadanie 6.2.21

Listing 10.127. Rozwiązanie zadania 6.2.21 w języku C/C++

```
void pom (unsigned int n, int * tab){
    int i ,p;
    for(i=0;i<n/2;i++){
        p=tab[i];
        tab[i]=tab[n-1-i];
        tab[n-1-i]=p;
    }
}

void odwroc(int **t, unsigned int n, unsigned int m){
    int i ,j;
    for(i=0;i<n;i++)
        pom(n,t[i]);
}
```

W powyższym rozwiązaniu wykorzystano strukturę wielowymiarowych tablic tablic oraz rozwiązanie zadania 4.2.12a.

Zadanie 6.2.22

Listing 10.128. Rozwiązanie zadania 6.2.22 w języku C

```
void pom (unsigned int n, unsigned int j, int tab [][]n) {
    int i ,p;
    for(i=0;i<n/2;i++){
        p=tab[j][i];
        tab[j][i]=tab[j][n-1-i];
        tab[j][n-1-i]=p;
    }
}

void odwroc(unsigned int n, unsigned int m, int t [][]m) {
    unsigned int i ,j;
    for(i=0;i<n;i++)
        pom(n,i,t);
}
```

Zadanie 6.2.23 Zadanie to można rozwiązać na kilka sposobów. Korzystając z faktu, że funkcja dostaje jako argument tablicę tablic można po prostu przepiąć wskaźniki do wierszy:

Listing 10.129. Rozwiązanie zadania 6.2.23 w języku C/C++

```
void przepnij( unsigned int n, unsigned int m, int ** t){
    int i;
    int * pom=t[n-1];
    for(i=n-1;i>0;i--)
        t[i]=t[i-1];
    t[0]=pom;
}
```

Powyzszego rozwiązania nie można zastosować w przypadku, gdy istnieje możliwość, że gdzieś w programie przechowywany jest wskaźnik do pojedynczego wiersza przekazanej w argumencie tablicy. W takim przypadku trzeba przepisywać wartości poszczególnych elementów tablicy:

Listing 10.130. Rozwiązań zadania 6.2.23 w języku C/C++

```
void przepisz( unsigned int n, unsigned int m, int ** t){
    int i,j,pom;
    for(i=0;i<m;i++){
        pom=t[n-1][i];
        for(j=n-1;j>0;j--)
            t[j][i]=t[j-1][i];
        t[0][i]=pom;
    }
}
```

Drugie z zaprezentowanych rozwiązań ma tę zaletę, że jest możliwe do zastosowania także w przypadku tablic dwuwymiarowych oraz łatwo je zaadoptować do zamiany miejscami kolumn.

Zadanie 6.2.27

Listing 10.131. Rozwiązań zadania 6.2.27 w języku C/C++

```
void zamien( unsigned int n, int *** tab){
    int i,j,k,pom;
    for(i=0;i<n;i++){
        for(j=i;j<n;j++){
            if (i==j)
                k=i;
            else
                k=i+1;
            for (;k<n;k++){
                pom=tab[i][j][k];
                tab[i][j][k]=tab[j][k][i];
                tab[j][k][i]=tab[k][i][j];
                tab[k][i][j]=pom;
            }
        }
    }
}
```

Zadanie 6.2.32

Listing 10.132. Rozwiązanie zadania 6.2.32 w języku C/C++

```

int ** pomnoz(unsigned int n, int **tab1, int **tab2){
    int i,j,k;
    int ** tab3=malloc(n*sizeof(int *));
    for(i=0;i<n;i++)
        tab3[i]=malloc(n*sizeof(int));
    for(i=0;i<n;i++)
        for(j=0;j<n;j++){
            tab3[i][j]=0;
            for(k=0;k<n;k++)
                tab3[i][j]+=tab1[i][k]*tab2[k][j];
        }
    return tab3;
}

```

Zadanie 6.2.36

Listing 10.133. Rozwiązanie zadania 6.2.36 w języku C

```

int ** utworz(int n){
    int i, ** t=malloc(n*sizeof(int *));
    for(i=0;i<n;i++)
        t[i]=malloc(n*sizeof(int));
    return t;
}

void usun(int n, int ** t){
    int i;
    for(i=0;i<n;i++)
        free(t[i]);
    free(t);
}

void przepisz(int n, int m, int ** t1, int ** t2){
    int i1,i2,j1,j2;
    for(i1=0,i2=0;i1<n;i1++)
        if (i1!=m){
            for(j1=1,j2=0;j1<n;j1++,j2++)
                t2[i2][j2]=t1[i1][j1];
            i2++;
        }
}

int wyznacznik (unsigned int n, int **tab){
    int i,j, wyz;

```

```

int ** t;
if (n==1)
    return tab[0][0];
wyz=0;
t=utworz(n-1);
for(i=0;i<n;i++){
    przepisz(n,i,tab,t);
    if (i%2==0)
        wyz+=tab[i][0]*wyznacznik(n-1,t);
    else
        wyz-=tab[i][0]*wyznacznik(n-1,t);
}
usun(n,t);
return wyz;
}

```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 6.2.37

Listing 10.134. Rozwiązanie zadania 6.2.37 w języku C/C++

```

void przepisz(int n, int m, int t1[][n], int t2[][n-1]){
    int i1,i2,j1,j2;
    for(i1=0,i2=0;i1<n;i1++)
        if (i1!=m){
            for(j1=1,j2=0;j1<n;j1++,j2++)
                t2[i2][j2]=t1[i1][j1];
            i2++;
        }
}

int wyznacznik (unsigned int n, int tab[][n]){
    int i,j, wyz;
    int t[n-1][n-1];
    if (n==1)
        return tab[0][0];
    wyz=0;
    for(i=0;i<n;i++){
        przepisz(n,i,tab,t);
        if (i%2==0)
            wyz+=tab[i][0]*wyznacznik(n-1,t);
        else
            wyz-=tab[i][0]*wyznacznik(n-1,t);
    }
    return wyz;
}

```

10.9. Rozwiązania do zadań z rozdziału 7.2

Zadanie 7.2.1

Listing 10.135. Rozwiązanie zadania 7.2.1 w języku C/C++

```
struct trojkat {
    double a,b,c;
};

double obwod(struct trojkat t){
    return t.a+t.b+t.c;
}
```

Ważną rzeczą jest, żeby nie zapominać o średniku po definicji typów złożonych. Komunikat kompilatora o błędzie w przypadku pominięcia średnika może mówić o błędzie w zupełnie innym miejscu programu.

W języku C++, nie trzeba powtarzać słowa kluczowego **struct** przy deklaracji zmiennej mającej przechowywać wcześniej zdefiniowaną strukturę, a więc rozwiązanie w nim zadania mogłoby wyglądać następująco:

Listing 10.136. Rozwiązanie zadania 7.2.1 w języku C++

```
struct trojkat {
    double a,b,c;
};

double obwod(trojkat t){
    return t.a+t.b+t.c;
}
```

W języku C, aby uniknąć powtarzania przed nazwą typu słowa kluczowego **struct** (a także słów kluczowych **union** i **enum**), można użyć polecenia **typedef**:

Listing 10.137. Rozwiązanie zadania 7.2.1 w języku C/C++

```
struct trojkat {
    double a,b,c;
};

typedef struct trojkat troj;

double obwod(troj t){
    return t.a+t.b+t.c;
}
```

Polecenia `typedef` można używać także w języku C++.

Zadanie 7.2.2

Listing 10.138. Rozwiązanie zadania 7.2.2 w języku C/C++

```
void przepisz(struct trojkat troj1, struct trojkat *troj2){
    *troj2=troj1;
}
```

Warto zauważyć, że wszystkie pola struktury są przepisywane za jednym zamachem.

Zadanie 7.2.3

Listing 10.139. Rozwiązanie zadania 7.2.3 w języku C/C++

```
struct punkt{
    double x,y,z;
};

double minimum(struct punkt tab[], int n){
    int i,j;
    double pom,min=sqrt(pow(tab[1].x-tab[0].x,2)
        +pow(tab[1].y-tab[0].y,2)
        +pow(tab[1].z-tab[0].z,2));
    for(i=0;i<n-1;i++)
        for(j=i+1;j<n; j++){
            pom=sqrt(pow(tab[j].x-tab[i].x,2)
                +pow(tab[j].y-tab[i].y,2)
                +pow(tab[j].z-tab[i].z,2));
            if (pom<min) min=pom;
        }
    return min;
}
```

Zadanie 7.2.4

Listing 10.140. Rozwiązanie zadania 7.2.4 w języku C/C++

```
void przepisz(struct punkt tab1[], struct punkt tab2[],
                unsigned int n){
    int i;
    for(i=0;i<n; i++)
        tab2[i]=tab1[i];
}
```

Zadanie 7.2.5

Listing 10.141. Rozwiązanie zadania 7.2.5 w języku C/C++

```
struct punkt10{
    double t[10];
};

void przepisz(struct punkt10 tab1[], struct punkt10 tab2[],  
              unsigned int n){
    int i;
    for(i=0;i<n;i++)
        tab2[i]=tab1[i];
}
```

W zdefiniowanej powyżej funkcji **przepisz** cała struktura, a więc także cała zawartość pola **t** jest przepisywana przy użyciu pojedynczego operatora przypisania.

Zadanie 7.2.6

Listing 10.142. Rozwiązanie zadania 7.2.6 w języku C

```
struct punktn{
    double *t;
    int w;
};

void przepisz(struct punktn tab1[], struct punktn tab2[],  
              unsigned int n){
    int i,j;
    for(i=0;i<n;i++){
        tab2[i].t=malloc(tab1[i].w*sizeof(double));
        for(j=0;j<tab1[i].w;j++)
            tab2[i].t[j]=tab1[i].t[j];
    }
}
```

W tym przypadku należy tworzyć i przepisywać tablice ręcznie (operator przypisania dla **struct punktn** przepisałby wskaźnik, ale nie stworzyłby nowej tablicy).

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.2.9

Listing 10.143. Rozwiązanie zadania 7.2.9 w języku C/C++

```
struct lista{
    int i;
```

```
struct lista * wsk;
};
```

Zadanie 7.2.10

Listing 10.144. Rozwiązanie zadania 7.2.10 w języku C/C++

```
union super_int{
    int i;
    unsigned int u;
};
```

Zadanie 7.2.11

Listing 10.145. Rozwiązanie zadania 7.2.11 w języku C

```
union Liczba{
    int i;
    double d;
};

struct Dane{
    int tp;
    union Liczba zaw;
};

struct Dane wczytaj(){
    struct Dane dane;
    printf("Jezeli chcesz podac liczbe calkowita wpisz 0\n");
    printf("jeżeli chcesz podac liczbe wymierna wpisz 1\n");
    scanf("%d", &dane.tp);
    if (dane.tp==0){
        printf("Wpisz liczbe calkowita");
        scanf("%d", &dane.zaw.i);
    }
    else{
        printf("Wpisz liczbe wymierna");
        scanf("%lf", &dane.zaw.d);
    }
    return dane;
}
```

W językach C i C++ rozróżniane są duże i małe litery. Dlatego w funkcji `wczytaj` możliwe było zadeklarowanie zmiennej `dane` typu `struct Dane`. W praktyce nie należy używać w programie nazw, które różnią się tylko wielkością liter. Używanie w programie podobnych nazw zmiennych, funkcji etc. sprzyja robieniu przypadkowych i trudnych do wykrycia błędów.

Rozwiązań w języku C++ wygląda analogicznie.

Zadanie 7.2.13

Listing 10.146. Rozwiązań zadania 7.2.13 w języku C/C++

```

struct trojkat{
    double a,h;
};

struct prostokat{
    double a,b;
};

struct trapez{
    double a,b,h;
};

union wymiary{
    struct trojkat troj, rown;
    struct prostokat prost;
    struct trapez trap;
};

struct figura{
    union wymiary wym;
    int fig;
};

double pole(struct figura f){
    switch (f.fig){
        case 0: return f.wym.troj.a*f.wym.troj.h/2;
        case 1: return f.wym.prost.a*f.wym.prost.b;
        case 2: return f.wym.rown.a*f.wym.rown.h;
        case 3: return f.wym.trap.h
                  *(f.wym.trap.a+f.wym.trap.b)/2;
    }
}

```

W powyższym rozwiązaniu pola `rown` i `troj` unii `wymiary` są tego samego typu. Jest to możliwe gdyż do liczenia pól trójkątów i równoległoboków potrzebne są te same wymiary (podstawa i wysokość).

Zadanie 7.2.14

Listing 10.147. Rozwiązań zadania 7.2.14 w języku C/C++

```

enum czworokat{
    kwadrat, prostokat, romb, rownoleglobok, trapez
};

```

Zadanie 7.2.17

Listing 10.148. Rozwiązanie zadania 7.2.17 w języku C

```
enum Plec{
    mezczyzna, kobieta
};

enum mezczyzna{
    wolny, zonaty
};

enum kobieta{
    wolna, mezatka
};

union czlowiek{
    enum mezczyzna m;
    enum kobieta k;
};

struct dane_osobowe{
    char imie[30];
    char nazwisko[30];
    enum Plec plec;
    union czlowiek stan_cywilny;
};

void wczytaj(struct dane_osobowe tab[], int n){
    int i, d;

    for(i=0;i<n;i++){
        printf("Czy teraz wczytujemy dane kobiety (1) ");
        printf("czy mezczyzny (2)?");
        scanf("%d",&d);
        if (d==1)
            tab[i].plec=kobieta;
        else
            tab[i].plec=mezczyzna;

        printf("podaj imie");
        scanf("%s",tab[i].imie);

        printf("podaj nazwisko");
        scanf("%s",tab[i].nazwisko);

        printf("podaj stan cywilny");
        if (tab[i].plec==kobieta)
            printf("(wolna - 0, mezatka - 1)");
    }
}
```

```

    else
        printf("(wolny=0,zonaty=1)");
        scanf("%d",&tab[ i ].stan_cywilny.k);
    }
}

```

Jak widać w powyższym rozwiążaniu, typ wyliczeniowy można wczytywać jak zmienne typu `int`. Ponadto niezależnie, czy wczytywane są dane kobiety czy mężczyzny, wczytywane są dane do pola `k` w unii `stan_cywilny` typu `union człowiek`. Można tak zrobić, gdyż pola `k` i `m` znajdują się w tym samym miejscu w pamięci, a więc wczytując dane do jednego pola, zmienia się jednocześnie wartość drugiego. Ponadto oba wspomniane pola są typów wyliczeniowych o tej samej liczbie zdefiniowanych wartości.

Funkcja `wczytaj` w języku C++ może wyglądać następująco:

Listing 10.149. Rozwiążanie zadania 7.2.17 w języku C++

```

void wczytaj(struct dane_osobowe tab[], int n){
    int i,d;

    for(i=0;i<n;i++){
        cout<<"Czy teraz wczytujemy dane kobiety (1) ";
        cout<<"czy mezczyzny (2) ?" <<endl;
        cin>>d;
        if (d==1)
            tab[ i ].plec=kobieta;
        else
            tab[ i ].plec=mezczyzna;

        cout<<"Podaj imię " <<endl;
        cin>>tab[ i ].imie;

        cout<<"Podaj nazwisko " <<endl;
        cin>>tab[ i ].nazwisko;

        cout<<"Podaj stan cywilny " <<endl;;
        if (tab[ i ].plec==kobieta){
            cout<<"wolna=0,mezatka=1" <<endl;
            cin>>d;
            if (d==0)
                tab[ i ].stan_cywilny.k=wolna;
            else
                tab[ i ].stan_cywilny.k=mezatka;
        }
        else{
            cout<<"wolny=0,zonaty=1" <<endl;
            cin>>d;
            if (d==0)
                tab[ i ].stan_cywilny.m=wolny;
            else
                tab[ i ].stan_cywilny.m=zonaty;
        }
    }
}

```

```

        else
            tab[ i ].stan_cywilny.m=zonaty ;
        }
    }
}

```

Podstawową różnicą w stosunku do rozwiązania w języku C jest niemożność wczytania wprost wartości do zmiennej typu wyliczeniowego. Aby to zrobić, trzeba by przeciążyć operator „`>>`”, ale to nie mieści się w zakresie materiału niniejszego zbioru zadań. Innym rozwiązaniem mogłoby być wczytanie wartości typu `int` i zrzutowanie jej do typu wyliczeniowego.

Podobnie jak w C, w języku C++ typy wyliczeniowe są domyślnie wyświetlane jako wartości typu `int`.

Zadanie 7.2.18

Listing 10.150. Rozwiązanie zadania 7.2.18 w języku C/C++

```

union sup_int{
    unsigned int l;
    unsigned char t[4];
};

```

Poprawność powyższego rozwiązania jest zależna od używanego kompilatora, gdyż standard języka C nie określa, z ilu bajtów ma się składać zmienna typu `int`. Typowy rozmiar typu `int` w systemach 32-bitowych to 4 bajty, stąd taki rozmiar tablicy `t`.

Zadanie 7.2.19

Listing 10.151. Rozwiązanie zadania 7.2.19 w języku C/C++

```

unsigned int funkcja(unsigned int a, unsigned int b){
    union sup_int poma, pomb, pomwyn;
    poma.l=a;
    pomb.l=b;
    pomwyn.t[0]=poma.t[0]*pomb.t[0];
    pomwyn.t[1]=poma.t[1]*pomb.t[1];
    pomwyn.t[2]=poma.t[2]*pomb.t[2];
    pomwyn.t[3]=poma.t[3]*pomb.t[3];
    return pomwyn.l;
}

```

10.10. Rozwiązania do zadań z rozdziału 7.3

Zadanie 7.3.1

Listing 10.152. Rozwiązanie zadania 7.3.1 w języku C/C++

```
struct element * utworz(){
    return NULL;
}
```

Zadanie 7.3.2

Listing 10.153. Rozwiązanie zadania 7.3.2 w języku C

```
void wyczysc(struct element* Lista){
    struct element * wsk=Lista;
    while(Lista!=NULL){
        Lista=Lista->next;
        free(wsk);
        wsk=Lista;
    }
}
```

Rozwiązanie w języku C++:

Listing 10.154. Rozwiązanie zadania 7.3.2 w języku C++

```
void wyczysc(element* Lista){
    element * wsk=Lista;
    while(Lista!=NULL){
        Lista=Lista->next;
        delete wsk;
        wsk=Lista;
    }
}
```

Zadanie 7.3.3

Listing 10.155. Rozwiązanie zadania 7.3.3 w języku C

```
struct element * dodaj(struct element* Lista, int a){
    struct element * wsk=malloc(sizeof(struct element));
    wsk->i=a;
    wsk->next=Lista;
    return wsk;
}
```

Rozwiązanie w języku C++:

Listing 10.156. Rozwiązanie zadania 7.3.3 w języku C++

```
element * dodaj(element* Lista , int a){
    element * wsk = new element;
    wsk->i=a;
    wsk->next=Lista ;
    return wsk;
}
```

Zadanie 7.3.4Listing 10.157. Rozwiązanie zadania 7.3.4 w języku C

```
struct element * dodajk(struct element* Lista , int a){
    struct element * wsk;
    if (Lista==NULL){
        Lista=wsk=malloc(sizeof(struct element));
    }
    else{
        wsk=Lista ;
        while(wsk->next!=NULL)
            wsk=wsk->next;
        wsk->next=malloc(sizeof(struct element));
        wsk=wsk->next;
    }
    wsk->i=a;
    wsk->next=NULL;
    return Lista;
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.5Listing 10.158. Rozwiązanie zadania 7.3.5 w języku C

```
struct element* dodajw(struct element* Lista ,
                        struct element* elem,int a){
    struct element * wsk=malloc(sizeof(struct element));
    wsk->i=a;
    if (elem==NULL){
        wsk->next=Lista ;
        Lista=wsk;
    }
    else{
        wsk->next=elem->next ;
        elem->next=wsk;
    }
    return Lista;
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.6

Listing 10.159. Rozwiązanie zadania 7.3.6 w języku C/C++

```
struct element * znajdz(struct element* Lista , int a){
    while((Lista!=NULL)&&(Lista->i!=a))
        Lista=Lista->next;
    return Lista;
}
```

W rozwiązaniu wykorzystany został fakt, że w językach C i C++, jeżeli pierwszy argument koniunkcji jest fałszywy, to drugi nie jest już sprawdzany (dzięki temu w drugim argumencie koniunkcji można założyć, że `Lista->i` istnieje).

Zadanie 7.3.7

Listing 10.160. Rozwiązanie zadania 7.3.7 w języku C

```
struct element * usun(struct element* Lista , int a){
    struct element * wsk,*wsk2;
    if (Lista==NULL)
        return Lista;
    wsk=Lista;
    if (Lista->i==a){
        Lista=Lista->next;
        free(wsk);
    }
    else {
        while((wsk->next!=NULL)&&(wsk->next->i!=a))
            wsk=wsk->next;
        if (wsk->next!=NULL){
            wsk2=wsk->next;
            wsk->next=wsk2->next;
            free(wsk2);
        }
    }
    return Lista;
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.8

Listing 10.161. Rozwiązanie zadania 7.3.8 w języku C

```
struct element * usunw(struct element* Lista ,
                      struct element * elem){
    struct element * wsk,*wsk2;
    if (Lista==NULL)
        return Lista;

    wsk=Lista;

    if (Lista==elem){
        Lista=Lista->next;
        free(wsk);
        return Lista;
    }

    while(( wsk->next!=NULL)&&(wsk->next!=elem ))
        wsk=wsk->next;
    if (wsk->next!=NULL){
        wsk2=wsk->next;
        wsk->next=wsk2->next;
        free(wsk2);
    }
    return Lista;
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.9

Listing 10.162. Rozwiązanie zadania 7.3.9 w języku C

```
struct element * usunw2(struct element* Lista ,
                        struct element * elem){
    struct element * wsk;

    if(Lista==NULL)
        return Lista;

    if (elem==NULL){
        wsk=Lista;
        Lista=Lista->next;
    }
    else if (elem->next==NULL)
        return Lista;
    else{
        wsk=elem->next;
        elem->next=wsk->next;
    }
}
```

```
    free(wsk);
    return Lista;
}
```

Złożoność obliczeniowa powyższej funkcji jest mniejsza niż funkcji z Listingu 10.161. Z tego powodu operacje na listach w argumentach często wymagają podania, zamiast wskaźnika na jakiś element, wskaźnika do elementu poprzedniego.

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.10

Listing 10.163. Rozwiązanie zadania 7.3.10 w języku C

```
struct element * utworz(){
    struct element *wsk=malloc(sizeof(struct element));
    wsk->next=NULL;
    return wsk;
}
```

Inaczej niż ma to miejsce w przypadku listy bez głowy, tworzenie pustej listy z głową wymaga wykonania pewnych prostych operacji.

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.11

Listing 10.164. Rozwiązanie zadania 7.3.11 w języku C

```
void wyczysc(struct element* Lista){
    struct element * wsk=Lista->next;
    Lista=wsk;
    while(Lista!=NULL){
        Lista=Lista->next;
        free(wsk);
        wsk=Lista;
    }
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.12

Listing 10.165. Rozwiązanie zadania 7.3.12 w języku C

```
void dodaj(struct element *Lista , int a){
    struct element *wsk=malloc(sizeof(struct element));
    wsk->i=a;
```

```
wsk->next=Lista->next;
Lista->next=wsk;
}
```

W przeciwnieństwie do podobnego zadania dla list bez głowy, w przypadku list z głową funkcja nie musi zwracać wskaźnika do początku listy. Wynika to z tego, że w przypadku list z głową przed pierwszym elementem listy znajduje się głowa, do której wskaźnik się nie zmienia.

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.13

Listing 10.166. Rozwiązanie zadania 7.3.13 w języku C

```
void dodajk(struct element* Lista , int a){
    struct element * wsk=Lista ;
    while(wsk->next!=NULL)
        wsk=wsk->next ;
    wsk->next=malloc(sizeof(struct element));
    wsk=wsk->next ;
    wsk->i=a ;
    wsk->next=NULL;
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.14

Listing 10.167. Rozwiązanie zadania 7.3.14 w języku C

```
void dodajw(struct element *Lista , struct element * elem ,
            int a){
    struct element *wsk=malloc(sizeof(struct element));
    wsk->i=a ;
    wsk->next=elem->next ;
    elem->next=wsk ;
}
```

Należy zauważyć, że w powyższym rozwiązaniu pierwszy argument (wskaźnik na głowę listy) nie jest wykorzystywany..

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.15

Listing 10.168. Rozwiązanie zadania 7.3.15 w języku C/C++

```
struct element * znajdz(struct element* Lista , int a){
```

```

Lista=Lista->next;
while(( Lista!=NULL)&&(Lista->i!=a))
    Lista=Lista->next;
return Lista;
}

```

Zadanie 7.3.16

Listing 10.169. Rozwiązanie zadania 7.3.16 w języku C/C++

```

struct element * znajdzp(struct element* Lista , int a){
    while(( Lista->next!=NULL)&&(Lista->next->i!=a))
        Lista=Lista->next;
    return Lista;
}

```

Zadanie 7.3.17

Listing 10.170. Rozwiązanie zadania 7.3.17 w języku C

```

void usun(struct element* Lista , int a){
    struct element * wsk;
    while(( Lista->next!=NULL)&&(Lista->next->i!=a))
        Lista=Lista->next;
    if (Lista->next!=NULL){
        wsk=Lista->next;
        Lista->next=wsk->next;
        free(wsk);
    }
}

```

Porównując powyższe rozwiązanie z rozwiązaniem zadania 7.3.7, widać, o ile prostsze jest operowanie na listach z głową w stosunku do list bez głowy.

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.18

Listing 10.171. Rozwiązanie zadania 7.3.18 w języku C

```

void usunw(struct element* Lista , struct element * elem){
    struct element * wsk;
    while(( Lista->next!=NULL)&&(Lista->next!=elem))
        Lista=Lista->next;
    wsk=Lista->next;
    Lista->next=wsk->next;
    free(wsk);
}

```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.19

Listing 10.172. Rozwiązanie zadania 7.3.19 w języku C

```
void usunw2(struct element* Lista, struct element * elem){
    struct element * wsk=elem->next;
    elem->next=wsk->next;
    free(wsk);
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.20 Wersja dla listy bez głowy:

Listing 10.173. Rozwiązanie zadania 7.3.20 w języku C/C++

```
void zeruj(struct element* Lista){
    while(Lista!=NULL){
        Lista->i=0;
        Lista=Lista->next;
    }
}
```

Wersja dla listy z głową:

Listing 10.174. Rozwiązanie zadania 7.3.20 w języku C/C++

```
void zeruj(struct element* Lista){
    while(Lista->next!=NULL){
        Lista=Lista->next;
        Lista->i=0;
    }
}
```

Prostym sposobem uzyskania rozwiązania zadania dla listy wskaźnikowej z głową jest dodanie jednego wiersza do rozwiązania dedykowanego dla listy wskaźnikowej bez głowy:

Listing 10.175. Rozwiązanie zadania 7.3.20 w języku C/C++

```
void zeruj(struct element* Lista){
    Lista=Lista->next;
    while(Lista!=NULL){
        Lista->i=0;
        Lista=Lista->next;
    }
}
```

Zadanie 7.3.23 Wersja dla listy bez głowy:

Listing 10.176. Rozwiązanie zadania 7.3.23 w języku C/C++

```

struct trojka{
    unsigned int a,b,c;
    struct trojka * next;
};

bool tr(struct trojka * e){
    if (e->a*e->a + e->b*e->b == e->c*e->c)
        return true;
    else
        return false;
}

struct trojka * pitagoras(struct trojka* Lista){
    struct trojka * pom, *pom2;
    while ((Lista!=NULL)&&(!tr(Lista))){
        pom=Lista;
        Lista=Lista->next;
        free(pom);
    }
    if (Lista==NULL)
        return NULL;
    pom=Lista;
    while(pom->next!=NULL){
        if (tr(pom->next))
            pom=pom->next;
        else{
            pom2=pom->next;
            pom->next=pom2->next;
            free(pom2);
        }
    }
    return Lista;
}

```

Rozwiązanie dla listy z głową różni się tylko funkcją pitagoras:

Listing 10.177. Rozwiązanie zadania 7.3.23 w języku C/C++

```

void pitagoras(struct trojka* Lista){
    struct trojka * pom;
    while(Lista->next!=NULL){
        if (tr(Lista->next))
            Lista=Lista->next;
        else{
            pom=Lista->next;
            Lista->next=pom->next;
            free(pom);
        }
    }
}

```

```
    }
}
}
```

Zadanie 7.3.24 Wersja dla listy bez głowy:

Listing 10.178. Rozwiązanie zadania 7.3.24 w języku C/C++

```
int suma (struct element* Lista){
    int sum=0;
    while(Lista!=NULL){
        sum+=Lista->i ;
        Lista=Lista->next ;
    }
    return sum;
}
```

Wersja dla listy z głową jest podobna:

Listing 10.179. Rozwiązanie zadania 7.3.24 w języku C/C++

```
int suma (struct element* Lista){
    int sum=0;
    Lista=Lista->next ;
    while(Lista!=NULL){
        sum+=Lista->i ;
        Lista=Lista->next ;
    }
    return sum;
}
```

Zadanie 7.3.25 Wersja dla listy bez głowy:

Listing 10.180. Rozwiązanie zadania 7.3.25 w języku C/C++

```
int minimum (struct element* Lista){
    int min=Lista->i ;
    while(Lista!=NULL){
        if (Lista->i<min)
            min=Lista->i ;
        Lista=Lista->next ;
    }
    return min;
}
```

Wersja dla listy z głową jest podobna:

Listing 10.181. Rozwiązanie zadania 7.3.25 w języku C/C++

```
int minimum (struct element* Lista){
    int min=Lista->next->i ;
    while(Lista->next!=NULL){
        Lista=Lista->next ;
        if (Lista->i<min)
            min=Lista->i ;
    }
    return min;
}
```

Zadanie 7.3.26 Wersja dla listy bez głowy:

Listing 10.182. Rozwiązań zadania 7.3.26 w języku C/C++

```
struct element * minimum (struct element* Lista){
    struct element * min=Lista ;
    while(Lista!=NULL){
        if (Lista->i<min->i)
            min=Lista ;
        Lista=Lista->next ;
    }
    return min;
}
```

Wersja dla listy z głową jest podobna.

Zadanie 7.3.27 Wersja dla listy bez głowy:

Listing 10.183. Rozwiązań zadania 7.3.27 w języku C/C++

```
struct element * minimum (struct element* Lista){
    struct element * min=NULL, *pom;
    if ((Lista==NULL) || (Lista->next==NULL))
        return NULL;
    pom=Lista ;
    while(pom->next!=NULL){
        if (((min==NULL)&&(pom->next->i<Lista->i)) ||
            ((min!=NULL)&&(pom->next->i<min->next->i)))
            min=pom;
        pom=pom->next ;
    }
    return min;
}
```

Wersja dla listy z głową jest prostsza:

Listing 10.184. Rozwiązań zadania 7.3.27 w języku C/C++

```
struct element * minimum (struct element* Lista){
```

```

struct element * min=Lista;
while(Lista->next!=NULL){
    if (Lista->next->i<min->next->i )
        min=Lista ;
    Lista=Lista->next ;
}
return min;
}

```

Zadanie 7.3.28 Wersja dla listy bez głowy:

Listing 10.185. Rozwiązanie zadania 7.3.28 w języku C/C++

```

int maks_rozn (struct element* Lista){
    struct element * pom;
    int maks=abs(Lista->i-Lista->next->i);
    for (;Lista->next!=NULL;Lista=Lista->next)
        for(pom=Lista->next ;pom!=NULL;pom=pom->next)
            if ( abs(pom->i-Lista->i)<maks)
                maks=abs(pom->i-Lista->i );
    return maks;
}

```

Powyzsze rozwiązanie polega na przeszukaniu wszystkich par elementów listy. Efektywniejsze obliczeniowo byłoby znalezienie elementów o najmniejszej i największej wartości pola **i**, i zwrócenie ich różnicy.

Wersja dla listy z głową jest podobna.

Zadanie 7.3.29 Wersja dla listy bez głowy:

Listing 10.186. Rozwiązanie zadania 7.3.29 w języku C

```

struct element * kopijuj(struct element* Lista){
    struct element * kopia ,*pom;
    if (Lista==NULL)
        return NULL;
    kopia=malloc(sizeof(struct element));
    pom=kopia ;
    pom->i=Lista->i ;
    Lista=Lista->next ;
    while(Lista!=NULL){
        pom->next=malloc(sizeof(struct element));
        pom=pom->next ;
        pom->i=Lista->i ;
        Lista=Lista->next ;
    }
    pom->next=NULL;
    return kopia ;
}

```

Uważny czytelnik łatwo przerobi powyższe rozwiązanie na rozwiązanie w języku C++ oraz na rozwiązanie dla listy wskaźnikowej z głową.

Zadanie 7.3.30

Listing 10.187. Rozwiązanie zadania 7.3.30 w języku C/C++

```
struct element * sklej(struct element* Lista1 ,
                      struct element * Lista2){
    struct element * pom;
    if (Lista1==NULL)
        return Lista2;
    pom=Lista1;
    while(pom->next!=NULL)
        pom=pom->next;
    pom->next=Lista2;
    return Lista1;
}
```

Zadanie 7.3.31 Wersja dla listy bez głowy:

Listing 10.188. Rozwiązanie zadania 7.3.31 w języku C/C++

```
struct element * odwroc(struct element* Lista){
    struct element * pom1, *pom2;
    if ((Lista==NULL) || (Lista->next==NULL))
        return Lista;
    pom1=Lista->next;
    pom2=pom1->next;
    Lista->next=NULL;
    pom1->next=Lista;
    while(pom2!=NULL){
        Lista=pom1;
        pom1=pom2;
        pom2=pom2->next;
        pom1->next=Lista;
    }
    return pom1;
}
```

Wersja dla listy z głową jest podobna.

Zadanie 7.3.32

Listing 10.189. Rozwiązanie zadania 7.3.32 w języku C/C++

```
struct element * polacz(struct element* Lista1 ,
                        struct element* Lista2){
    struct element * pom, *pom2;
    if (Lista1==NULL)
```

```

return NULL;
pom=pom2=Lista1 ;
Lista1=Lista1->next ;
pom2->next=Lista2 ;
pom2=Lista2 ;
Lista2=Lista2->next ;
while(Lista1!=NULL){
    pom2->next=Lista1 ;
    pom2=Lista1 ;
    Lista1=Lista1->next ;
    pom2->next=Lista2 ;
    pom2=Lista2 ;
    Lista2=Lista2->next ;
}
return pom;
}

```

Zadanie 7.3.33 Wersja dla listy bez głowy:

Listing 10.190. Rozwiązanie zadania 7.3.33 w języku C/C++

```

struct element * przesun(struct element* Lista){
    struct element * pom;
    if ((Lista==NULL) || (Lista->next==NULL))
        return Lista;
    pom=Lista;
    while(Lista->next->next!=NULL)
        Lista=Lista->next;
        Lista->next->next=pom;
        pom=Lista->next;
        Lista->next=NULL;
    return pom;
}

```

Wersja dla listy z głową jest podobna.

Zadanie 7.3.34 Wersja dla listy bez głowy:

Listing 10.191. Rozwiązanie zadania 7.3.34 w języku C

```

bool wystepuje(struct element * Lista , int i){
    while(Lista!=NULL){
        if (Lista->i==i)
            return true;
        Lista=Lista->next;
    }
    return false;
}

struct element * powtorzone(struct element* Lista1 ,

```

```

struct element* Lista2){
struct element * Lista3=NULL, *pom,*pom2;
if ((Lista1==NULL)||(Lista2==NULL))
    return NULL;
pom=Lista1;
while(pom!=NULL){
    if ((wystepuje(Lista2 , pom->i))
        &&(!wystepuje(Lista3 , pom->i))){
        if (Lista3==NULL)
            pom2=Lista3=malloc (sizeof (struct element));
        else{
            pom2->next=malloc (sizeof (struct element));
            pom2=pom2->next ;
        }
        pom2->i=pom->i ;
    }
    pom=pom->next ;
}
pom2->next=NULL;
return Lista3;
}

```

Uważny czytelnik szybko stworzy wersję dla listy z głową. Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.35 Tym razem zaprezentowany zostanie prosty algorytm sortujący dla listy z głową:

Listing 10.192. Rozwiązań zadania 7.3.35 w języku C/C++

```

struct element * minimum (struct element* Lista){
    struct element * min=Lista;
    while(Lista->next!=NULL){
        if (Lista->next->i<min->next->i )
            min=Lista ;
        Lista=Lista->next ;
    }
    return min;
}

void sort (struct element* Lista){
    struct element * pom,*pom2;
    while(Lista->next!=NULL){
        pom=minimum(Lista );
        if (pom!=Lista ){
            pom2=pom->next;
            pom->next=pom2->next;
            pom2->next=Lista->next ;
            Lista->next=pom2;
        }
    }
}

```

```

        Lista=Lista->next;
    }
}

```

Podobnie jak miało to miejsce w przypadku rozwiązania zadania 4.212d dotyczącego sortowania elementów tablic jednowymiarowych, także tym razem zostanie przedstawione nieco bardziej skomplikowany w implementacji ale też bardziej efektywny algorytm sortujący (w wersji dla list bez głowy):

Listing 10.193. Rozwiązanie zadania 7.3.35 w języku C/C++

```

struct element * merge(struct element* Lista1 ,
                      struct element *Lista2){
    struct element * pom, *pom2;
    if (Lista1->i<Lista2->i){
        pom=pom2=Lista1;
        Lista1=Lista1->next;
    }
    else{
        pom=pom2=Lista2;
        Lista2=Lista2->next;
    }
    while((Lista1!=NULL)&&(Lista2!=NULL))
        if (Lista1->i<Lista2->i){
            pom2->next=Lista1;
            Lista1=Lista1->next;
            pom2=pom2->next;
        }
        else{
            pom2->next=Lista2;
            Lista2=Lista2->next;
            pom2=pom2->next;
        }
    if (Lista1!=NULL)
        pom2->next=Lista1;
    else
        pom2->next=Lista2;
    return pom;
}

struct element * mergesort(struct element* Lista){
    struct element * pom1, *pom2, *l1, *l2;
    unsigned int i=0;
    if ((Lista==NULL) || (Lista->next==NULL))
        return Lista;
    l1=pom1=Lista;
    l2=pom2=Lista->next;
    Lista=Lista->next->next;
    while(Lista!=NULL){

```

```

if ( i%2){
    pom1->next=Lista ;
    pom1=pom1->next ;
}
else{
    pom2->next=Lista ;
    pom2=pom2->next ;
}
i++;
Lista=Lista->next ;
}
pom1->next=NULL;
pom2->next=NULL;
l1=mergesort( l1 );
l2=mergesort( l2 );
l1=merge( l1 , l2 );
return l1 ;
}

```

10.11. Rozwiązania do zadań z rozdziału 8.2

Zadanie 8.2.1 Program w wersji dla języka C:

Listing 10.194. Rozwiązanie zadania 8.2.1 w języku C

```

FILE * otworz(char * plik){
    return fopen( plik , "r" );
}

```

Program w wersji dla języka C++:

Listing 10.195. Rozwiązanie zadania 8.2.1 w języku C++

```

fstream* otworz(char * plik){
    return new fstream( plik , ios :: in );
}

```

Warto zapamiętać, że w powyższym rozwiążaniu funkcja **otworz** nie mogłaby zwrócić wartości strumienia **fstream** ze względu na to, że nie posiada on konstruktora kopiującego, ani nie można zdefiniować dla niego operatora przypisania. Funkcja **otworz** mogłaby zwrócić referencję do strumienia, ale w ten sposób program straciłby wskaźnik do niego i nie mógłby go w przyszłości usunąć z pamięci.

Zadanie 8.2.2 Rozwiązanie w wersji dla języka C:

Listing 10.196. Rozwiązań zadania 8.2.2 w języku C

```
void wypisz(FILE * plik){
    char c;
    while(fscanf(plik , "%c",&c)==1)
        printf("%c",c);
    fclose(plik);
}
```

Rozwiązań w wersji dla języka C++:

Listing 10.197. Rozwiązań zadania 8.2.2 w języku C++

```
void wypisz(fstream& plik){
    char c;
    while(! plik .eof()){
        plik .get(c);
        if (! plik .eof())
            cout<<c;
    }
    plik .close();
}
```

Zadanie 8.2.3 Rozwiązań w wersji dla języka C i znaków typu **char**:

Listing 10.198. Rozwiązań zadania 8.2.3 w języku C

```
void wypisz(char * sciezka){
    FILE * plik=fopen(sciezka , "r");
    char c;
    while(fscanf(plik , "%c",&c)==1)
        if (!isspace(c))
            printf("%c",c);
    fclose(plik);
}
```

Rozwiązań w wersji dla języka C i znaków typu **wchar_t**

Listing 10.199. Rozwiązań zadania 8.2.3 w języku C

```
void wypisz(char * sciezka){
    FILE * plik=fopen(sciezka , "r");
    wchar_t c;
    while(fscanf(plik , "%lc",&c)==1)
        if (!iswspace(c))
            printf("%lc",c);
    fclose(plik);
}
```

Rozwiązań w wersji dla języka C++ i znaków typu **char**:

Listing 10.200. Rozwiązanie zadania 8.2.3 w języku C++

```
void wypisz(char * sciezka){
    fstream plik(sciezka, ios::in);
    char c;
    while(! plik.eof()){
        plik>>c;
        if (! plik.eof())
            cout<<c;
    }
    plik.close();
}
```

W programie 10.200 nie ma konieczności zamykania pliku ręcznie, tak jak jest to zrobione. Zrobiłby to destruktor obiektu `plik` przy wyjściu z funkcji `wypisz`. Ręczne zamykanie pliku w takiej sytuacji ma sens wtedy, kiedy do zakończenia działania funkcji zostało jeszcze dużo czasu lub gdy program ma śledzić, czy przy zamykaniu pliku nie wystąpił błąd.

Rozwiązanie w wersji dla języka C++ i znaków typu `wchar_t`:

Listing 10.201. Rozwiązanie zadania 8.2.3 w języku C++

```
void wypisz(char * sciezka){
    wfstream plik(sciezka, ios::in);
    wchar_t c;
    while(! plik.eof()){
        plik>>c;
        if (! plik.eof())
            wcout<<c;
    }
}
```

Warto zapamiętać różnicę pomiędzy metodą `get` a operatorem `>>`. Metoda `get` wyłuskuje kolejne znaki, natomiast operator `>>` pomija białe znaki.

Zadanie 8.2.4 Rozwiązanie w wersji dla języka C i znaków typu `char`:

Listing 10.202. Rozwiązanie zadania 8.2.4 w języku C

```
int wystapien(char * sciezka, char c){
    char pom;
    int licz=0;
    FILE * plik=fopen(sciezka, "r");
    while(fscanf(plik, "%c",&pom)==1)
        if (c==pom)
            licz++;
    fclose(plik);
    return licz;
}
```

Rozwiązanie w wersji dla języka C++ i znaków typu `char`:

Listing 10.203. Rozwiązanie zadania 8.2.4 w języku C++

```
int wystapien(const char * sciezka , char c ){
    char pom;
    int licz=0;
    fstream plik(sciezka , ios::in);
    while(!plik.eof()){
        plik.get(pom);
        if ((!plik.eof())&&(c==pom))
            licz++;
    }
    return licz ;
}
```

Rozwiązanie dla znaków typu `wchar_t` jest analogiczne.

Zadanie 8.2.9 Rozwiązanie w wersji dla języka C i znaków typu `char`:

Listing 10.204. Rozwiązanie zadania 8.2.9 w języku C

```
int porownaj(char * sciezka1 , char * sciezka2){
    char c1,c2;
    FILE * plik1=fopen(sciezka1 , "r");
    FILE * plik2=fopen(sciezka2 , "r");
    fscanf(plik1 , "%c",&c1);
    fscanf(plik2 , "%c",&c2);
    while ((!feof(plik1))&&(!feof(plik2))){
        if (c1!=c2)
            return 0;
        fscanf(plik1 , "%c",&c1);
        fscanf(plik2 , "%c",&c2);
    }
    if ((!feof(plik1))||(!feof(plik2))){
        fclose( plik1 );
        fclose( plik2 );
        return 0;
    } else{
        fclose( plik1 );
        fclose( plik2 );
        return 1;
    }
}
```

Rozwiązań w wersji dla języka C++ i znaków typu char:

Listing 10.205. Rozwiązań zadania 8.2.9 w języku C++

```

int porownaj(char * sciezka1, char * sciezka2){
    char c1='0',c2='0';
    fstream plik1(sciezka1, ios::in);
    fstream plik2(sciezka2, ios::in);
    while((!plik1.eof())&&(!plik2.eof())){
        plik1.get(c1);
        plik2.get(c2);
        if (c1!=c2)
            return 0;
    }
    if ((!plik1.eof())||(!plik2.eof()))
        return 0;
    else
        return 1;
}

```

Rozwiązań dla znaków typu wchar_t jest analogiczne.

Zadanie 8.2.10 Rozwiązań w wersji dla języka C i znaków typu char:

Listing 10.206. Rozwiązań zadania 8.2.10 w języku C

```

int porownaj(char * sciezka1, char * sciezka2){
    char c1,c2;
    FILE * plik1=fopen(sciezka1, "r");
    FILE * plik2=fopen(sciezka2, "r");
    while ((fscanf(plik1, "%c",&c1)==1)&&(isspace(c1)));
    while ((fscanf(plik2, "%c",&c2)==1)&&(isspace(c2)));
    while ((!feof(plik1))&&(!feof(plik2))){
        if (c1!=c2)
            return 0;
        while ((fscanf(plik1, "%c",&c1)==1)&&(isspace(c1)));
        while ((fscanf(plik2, "%c",&c2)==1)&&(isspace(c2)));
    }
    if ((!feof(plik1))||(!feof(plik2))){
        fclose(plik1);
        fclose(plik2);
        return 0;
    } else{
        fclose(plik1);
        fclose(plik2);
        return 1;
    }
}

```

Rozwiązań w wersji dla języka C++ i znaków typu char:

Listing 10.207. Rozwiązanie zadania 8.2.10 w języku C++

```
int porownaj(char * sciezka1, char * sciezka2){
    char c1='0',c2='0';
    fstream plik1(sciezka1, ios::in);
    fstream plik2(sciezka2, ios::in);
    while((!plik1.eof())&&(!plik2.eof())){
        plik1>>c1;
        plik2>>c2;
        if (c1!=c2)
            return 0;
    }
    if ((!plik1.eof())||(!plik2.eof()))
        return 0;
    else
        return 1;
}
```

Rozwiązanie dla typu znaków **wchar_t** jest analogiczne.

Zadanie 8.2.11 Rozwiązanie dla języka C:

Listing 10.208. Rozwiązanie zadania 8.2.11 w języku C

```
FILE * otworz(char * plik){
    return fopen(plik, "a");
}
```

Rozwiązanie dla języka C++:

Listing 10.209. Rozwiązanie zadania 8.2.11 w języku C++

```
fstream* otworz(char * plik){
    return new fstream(plik, ios::out|ios::app);
}
```

Zadanie 8.2.12 Rozwiązania w wersji dla języka C i znaków typu **char**:

Listing 10.210. Rozwiązanie zadania 8.2.12 w języku C

```
void wczytaj(FILE * plik, int n){
    int i;
    char tab[100];
    for(i=0;i<n;i++){
        fgets(tab,100, stdin);
        fprintf(plik, "%s", tab);
    }
    fclose(plik);
}
```

Wadą powyższego rozwiązania jest fakt, że linie mogą mieć co najwyżej 99 znaków. Poniższe rozwiązanie jest pozbawione tej wady:

Listing 10.211. Rozwiązanie zadania 8.2.12 w języku C

```
void wczytaj(FILE * plik, int n){
    int i=0;
    char c;
    while(i<n){
        scanf("%c",&c);
        if (c=='\n')
            i++;
        if (i<n)
            fprintf(plik,"%c",c);
    }
    fclose(plik);
}
```

Rozwiązanie w wersji dla języka C++ i znaków typu **char**:

Listing 10.212. Rozwiązanie zadania 8.2.12 w języku C++

```
void wczytaj(fstream& plik, int n){
    int i;
    string s;
    for(i=0;i<n;i++){
        getline(cin,s);
        plik<<s<<endl;
    }
    plik.close();
}
```

W programie 10.212 trzeba zamknąć plik ręcznie, gdyż zmienna **plik** nie jest lokalnym obiektem funkcji **wczytaj**, ale referencją do obiektu utworzonego poza tą funkcją. W takiej sytuacji zakończenie działania funkcji **wczytaj** nie spowoduje zamknięcia pliku skojarzonego ze zmienną **plik**.

Rozwiązanie dla typu **wchar_t** jest analogiczne.

Zadanie 8.2.13 Program w wersji dla języka C:

Listing 10.213. Rozwiązanie zadania 8.2.13 w języku C

```
void przepisz(FILE * plik1, FILE * plik2){
    char c;
    while(fscanf(plik1,"%c",&c)==1)
        fprintf(plik2,"%c",c);
}
```

Program w wersji dla języka C++:

Listing 10.214. Rozwiązanie zadania 8.2.13 w języku C++

```
void przepisz(fstream& plik1 , fstream& plik2){
    char c;
    while(! plik1.eof()){
        plik1.get(c);
        if (! plik1.eof())
            plik2<<c;
    }
}
```

Zadanie 8.2.14 Ponieważ w teści nie jest zaznaczone, że przepisywany plik jest tekstowy, to należy go przepisać w trybie binarnym:

Najpierw wersja dla języka C:

Listing 10.215. Rozwiązanie zadania 8.2.14 w języku C

```
void przepisz(char * sciezka1 , char * sciezka2){
    FILE * plik1=fopen(sciezka1 , "rb");
    FILE * plik2=fopen(sciezka2 , "wb");
    char c[100];
    int n=100;
    while(n==100){
        n=fread(c,1,100 ,plik1 );
        fwrite(c,1 ,n ,plik2 )
    }
    fclose(plik1 );
    fclose(plik2 );
}
```

Wersja dla języka C++:

Listing 10.216. Rozwiązanie zadania 8.2.14 w języku C++

```
void przepisz(char * sciezka1 , char * sciezka2){
    fstream plik1(sciezka1 , ios::in|ios::bin);
    fstream plik2(sciezka2 , ios::out|ios::bin);
    char c[100];
    int n=100;
    while(n==100){
        plik1.read(c,100 );
        n=plik1.gcount();
        plik2.write(c,n);
    }
}
```

Zadanie 8.2.17 Wersja w języku C

Listing 10.217. Rozwiązanie zadania 8.2.17 w języku C

```
void zapisz(char * sciezka , int ** tab , int n , int m){
    FILE * plik=fopen(sciezka , "wb");
    int i;
    for(i=0;i<n;i++)
        fwrite(tab[i],sizeof(int) ,m, plik);
    fclose(plik);
}
```

Wersja w języku C++:

Listing 10.218. Rozwiązanie zadania 8.2.17 w języku C++

```
void zapisz(const char * sciezka , int ** tab , int n , int m){
    fstream plik(sciezka , ios::out | ios::binary);
    int i;
    for(i=0;i<n;i++)
        plik.write(reinterpret_cast<char *>(tab[i]) ,
                    sizeof(int)*m);
}
```

W powyższym rozwiązaniu konieczne było rzutowanie typu **int*** na typ **char*** gdyż pierwszy argument metody **write** klasy **fstream** musi być wskaźnikiem na typ **char**.

Zadanie 8.2.18 Wersja w języku C

Listing 10.219. Rozwiązanie zadania 8.2.18 w języku C

```
void wczytaj(char * sciezka , int ** tab , int n , int m){
    FILE * plik=fopen(sciezka , "rb");
    int i;
    for(i=0;i<n;i++)
        fread(tab[i],sizeof(int) ,m, plik);
    fclose(plik);
}
```

Rozwiązanie w języku C++

Listing 10.220. Rozwiązanie zadania 8.2.18 w języku C

```
void wczytaj(const char* sciezka ,int** tab ,int n ,int m){
    fstream plik(sciezka , ios::in | ios::binary);
    int i;
    for(i=0;i<n;i++)
        plik.read(reinterpret_cast<char *>(tab[i]) ,
```

```
    }  
    sizeof(int)*m);
```

10.12. Rozwiązania do zadań z rozdziału 9.2

Zadanie 9.2.1

Listing 10.221. Rozwiązanie zadania 9.2.1 w języku C/C++

```
#define suma(a,b,c) (a+b+c)
```

Dobrym nawykiem jest umieszczanie w nawiasach definicji makr (czyli tak, jak w powyższym rozwiązaniu $(ab+c)+$, a nie $ab+c+$).

Zadanie 9.2.3

Listing 10.222. Rozwiązanie zadania 9.2.3 w języku C/C++

```
#define wiekszy(a,b) \  
    if(a>b) printf("%d",a); else printf("%d",b)
```

Dyrektywy preprocesora powinny mieścić się w pojedynczej linii. Jeżeli z jakiegoś powodu chce się podzielić dyrektywę na kilka linii, należy w miejscach przejścia do nowej linii, wstawić lewy ukośnik, tak jak to ma miejsce w programie 10.222.

Zadanie 9.2.4

Listing 10.223. Rozwiązanie zadania 9.2.4 w języku C/C++

```
#define parzysta(a) (a%2==0)?1:0
```

Zadanie 9.2.6

Listing 10.224. Rozwiązanie zadania 9.2.6 w języku C/C++

```
#define najwiersza(a,b,c) if ((a>=b)&&(a>=c))\  
    printf("%d",a); else if (b>=c) printf("%d",b);\  
    else printf("%d",c)
```

Zadanie 9.2.7

Listing 10.225. Rozwiązanie zadania 9.2.7 w języku C/C++

```
#define najwiersza(a,b,c) ((a>=b)&&(a>=c))?a:((b>=c)?b:c)
```

Zadanie 9.2.8

Listing 10.226. Rozwiązań do zadania 9.2.8 w języku C/C++

```
#define petla(x,y) for(x=0;x<y;x++)
```

Zadanie 9.2.10

Listing 10.227. Rozwiązań do zadania 9.2.10 w języku C/C++

```
#define zeruj(x) x=0
```

10.13. Rozwiązań do zadań z rozdziału 9.3

Zadanie 9.3.1 Plik głównego programu:

Listing 10.228. Plik glowny.c

```
#include <stdio.h>
#include "pierwiastek.c"

int main(){
    double a,b,c,delta;
    printf("Podaj współczynniki równania kwadratowego");
    scanf("%lf %lf %lf",&a,&b,&c);
    delta=b*b-4*a*c;
    if (delta>0)
        printf("x1=%f x2=%f",(-b-pierw(delta))/(2*a),
               (-b+pierw(delta))/(2*a));
    else if (delta==0)
        printf("x=%f",(-b)/(2*a));
    else
        printf("Brak rozwiązań");
    return 0;
}
```

Plik z funkcją liczącą pierwiastek:

Listing 10.229. Plik pierwiastek.c

```
double pierw(double n){
    float p,k,sr;
    p=0;
    k=n;
    sr=(p+k)/2;
    while((sr!=p)&&(sr!=k)){
        if ((sr*sr)>n)
            k=sr;
```

```

    else
        p=sr ;
        sr=(p+k)/2;
    }
    return sr;
}

```

Aby skompilować powyższy program, należy oba pliki umieścić w jednym katalogu i skompilować plik głowny.c. Plik pierwiastek.c zostanie w całości wklejony do pliku głownego.c za sprawą dyrektywy `include`.

Rozwiążanie dla języka C++ jest analogiczne

Zadanie 9.3.3 Plik głównego programu:

Listing 10.230. Plik głowny.c

```

#include <stdio.h>
#include "pierwiastek.h"

int main(){
    double a,b,c,delta;
    printf("Podaj współczynniki równania kwadratowego");
    scanf("%lf %lf %lf",&a,&b,&c);
    delta=b*b-4*a*c;
    if (delta>0)
        printf("x1=%f x2=%f",(-b-pierw(delta))/(2*a),
               (-b+pierw(delta))/(2*a));
    else if (delta==0)
        printf("x=%f",(-b)/(2*a));
    else
        printf("Brak rozwiązań");
    return 0;
}

```

Plik nagłówkowy:

Listing 10.231. Plik pierwiastek.h

```

#ifndef _pierw_
#define _pierw_
double pierw(double);
#endif

```

I plik z funkcją liczącą pierwiastek:

Listing 10.232. Plik pierwiastek.c

```
#include "pierwiastek.h"
```

```
double pierw(double n){  
    float p,k,sr;  
    p=0;  
    k=n;  
    sr=(p+k)/2;  
    while((sr!=p)&&(sr!=k)){  
        if ((sr*sr)>n)  
            k=sr;  
        else  
            p=sr;  
        sr=(p+k)/2;  
    }  
    return sr;  
}
```

W pliku `pieriastek.h` umieszczone zostały dyrektywy preprocesora zabezpieczające przed wielokrotnym dołączaniem tego samego nagłówka. Nie jest to konieczne w powyższym przypadku, ale jest to standardowa praktyka przy tworzeniu bibliotek. Podobnie standardową praktyką jest dołączanie do plików ze źródłami bibliotek ich plików nagłówkowych. Dzięki temu ewentualne błędy w plikach nagłówkowych są wykrywane już na etapie komplikacji bibliotek.

Aby skompilować powyższy program, należy najpierw oddziennie skompliować piliki `glowny.c` i `pieriastek.c`, a następnie je połączyć. Przy użyciu kompilatora gcc wygląda to następująco:

```
gcc pieriastek.c -c -o pierwiatek.o  
gcc glowny.c -c -o glowny.o  
gcc glowny.o pieriastek.o -o glowny
```

Rozwiązanie dla języka C++ jest analogiczne.

Zadanie 9.3.4 W pliku `Makefile` zapisuje się potrzebne do wykonania operacje w odwrotnej kolejności (czyli jako pierwszą wymienia się operację, która ma być wykonana jako pierwsza):

Listing 10.233. plik Makefile

```
glowny: glowny.o pieriastek.o  
        gcc glowny.o pieriastek.o -o glowny  
  
glowny.o: glowny.c pieriastek.h  
        gcc glowny.c -c -o glowny.o  
  
pieriastek.o: pieriastek.h pieriastek.c  
        gcc pieriastek.c -c -o pieriastek.o
```

W powyższym rozwiązaniu wśród plików potrzebnych do stworzenia plików `glowny.o` i `pierwiastek.o` wymieniony został także plik `pierwiastek.h`, który nie jest parametrem w komplikacji. Dzięki temu program make, śledzi zmiany także w pliku nagłówkowym.

W pliku `Makefile` nie trzeba wymieniać wszystkich komend potrzebnych do komplikacji programu. Wystarczy wypisać zależności. Poniżej skrócona wersja pliku `Makefile`:

Listing 10.234. plik Makefile, krótsza wersja

```
glowny: glowny.o pierwiastek.o
        gcc glowny.o pierwiastek.o -o glowny

glowny.o: glowny.c pierwiastek.h
pierwiastek.o: pierwiastek.h pierwiastek.c
```

Rozwiązanie dla języka C++ jest analogiczne. Wystarczy podmienić `gcc` na `g++` i zmienić rozszerzenia nazw plików.

Zadanie 9.3.7 Plik głównej części programu:

Listing 10.235. Plik `glowny.c`

```
#include <stdio.h>
#include "zespolone.h"
#include "arytmetyka.h"

int main(){
    zespolone suma, wczyt;
    int i;
    printf("Ile liczb zespolonych chcesz zsumowac?");
    scanf("%d",&i);
    while(i>0){
        wczyt=wczytaj();
        suma=dodaj(suma, wczyt);
        i--;
    }
    wypisz(suma);
    return 0;
}
```

Plik nagłówkowy biblioteki `zespolone`:

Listing 10.236. Plik `zespolone.h`

```
#ifndef _zespolone_
#define _zespolone_

typedef struct zesp{ double r,u;} zespolone;
```

```
zespolone wczytaj();  
void wypisz(zespolone);
```

#endif

Plik główny biblioteki zespolone:

Listing 10.237. Plik zespolone.c

```
#include <stdio.h>
#include "zespolone.h"

zespolone wczytaj(){
    zespolone z;
    printf("Podaj \u0142\u0142\u0144czesc \u0142rzeczywista \u0142wczytywanej \u0142liczby");
    scanf("%lf",&(z.r));
    printf("Podaj \u0142\u0142\u0144czesc \u0142urojona \u0142wczytywanej \u0142liczby");
    scanf("%lf",&(z.u));
    return z;
}

void wypisz(zespolone z){
    printf("Czesc \u0142rzeczywista \u0142ma \u0142wartosc %f \u0142a \u0142czesc \u0142urojona %f",
           z.r,z.u);
}
```

Plik nagłówkowy biblioteki z funkcjami arytmetycznymi:

Listing 10.238. Plik arytmetyka.h

```
#ifndef _pierw_
#define _arytmetyka_

#include "zespolone.h"

zespolone dodaj(zespolone, zespolone);
zespolone pomnoz(zespolone, zespolone);

#endif
```

Plik biblioteki z funkcjami arytmetycznymi:

Listing 10.239. Plik arytmetyka.cs

```
#include "zespolone.h"
#include "arytmetyka.h"

zespolone dodaj(zespolone a, zespolone b){
```

```

zespolone z;
z.r=a.r+b.r;
z.u=a.u+b.u;
return z;
}

zespolone pomnoz(zespolone a, zespolone b){
    zespolone z;
    z.r=a.r*b.r-a.u*b.u;
    z.u=a.r*b.u+a.u*b.r;
    return z;
}

```

W plikach biblioteki z operacjami arytmetycznymi na liczbach zespolonych dyrektywą `include` został dołączony plik `zespolone.h` ze względu na zdefiniowany w nim typ `zespolone`.

Rozwiążanie w języku C++ jest analogiczne.

Zadanie 9.3.8

Listing 10.240. plik Makefile, krótsza wersja

```

glowny: glowny.o zespolone.o arytmetyka.o
        gcc glowny.o pierwiastek.o arytmetyka.o -o glowny

glowny.o: glowny.c zespolone.h arytmetyka.h
arytmetyka.o: arytmetyka.h zespolone.h arytmetyka.c
zespolone.o: zespolone.h zespolone.c

```

Zadanie 9.3.11 Plik głównej części programu:

Listing 10.241. Plik glowny.c

```

#include <stdio.h>
#include "dane.h"
#include "statystyka.h"

int main(){
    int i;
    printf("Dane ilu osob chcesz wczytac?");
    scanf("%d",&i);
    while(i>0){
        wczytaj();
        i--;
    }
    printf("%f",srednia());
    return 0;
}

```

Plik nagłówkowy biblioteki `dane`:

Listing 10.242. Plik dane.h

```
#ifndef _dane_
#define _dane_

typedef struct os {
    char imie[15];
    nazwisko[30];
    int wiek;
} osoba;

void wczytaj();
void wypisz(osoba);

#endif
```

Plik główny biblioteki `dane`:

Listing 10.243. Plik dane.c

```
#include <stdio.h>
#include <stdlib.h>
#include "dane.h"

int liczba=0;
int pojemnosc=0;
osoba * tab=NULL;

void wczytaj(){
    if (liczba==pojemnosc){
        osoba * pom=tab;
        tab=malloc(2*(pojemnosc+1)*sizeof(osoba));
        int i;
        for(i=0;i<pojemnosc; i++)
            tab[i]=pom[i];
        if (pom!=NULL)
            free(pom);
        pojemnosc=2*(pojemnosc+1);
    }
    printf("Podaj imię");
    scanf("%s",&(tab[liczba].imie));
    printf("Podaj nazwisko");
    scanf("%s",&(tab[liczba].nazwisko));
    printf("Podaj wiek");
    scanf("%d",&(tab[liczba].wiek));
    liczba++;
}

void wypisz(osoba o){
```

```

    printf("imie : %s\n", nazwisko : %s\n", wiek %d\n",
           o.imie ,o.nazwisko ,o.wiek);

}

```

Plik nagłówkowy biblioteki z funkcjami statystycznymi:

Listing 10.244. Plik statystyka.h

```

#ifndef _pierw_
#define _statystyka_

double srednia();
int minimalny();
int maksymalny();

#endif

```

Plik biblioteki z funkcjami statystycznymi:

Listing 10.245. Plik statystyka.c

```

#include <stdlib.h>
#include "dane.h"
#include "statystyka.h"

extern osoba *tab;
extern int liczba;

double srednia(){
    if (tab==NULL)
        return 0;
    int suma=0,i;
    for(i=0;i<liczba ; i++)
        suma+=tab[ i ].wiek ;
    return (double)suma/liczba ;
}

int minimalny(){
    if (tab==NULL)
        return 0;
    int min=tab[ 0 ].wiek ,i ;
    for(i=1;i<liczba ; i++)
        if (tab[ i ].wiek<min)
            min=tab[ i ].wiek ;
    return min;
}

int maksymalny(){
    if (tab==NULL)

```

```

return 0;
int maks=tab[0].wiek , i ;
for(i=1;i<liczba ; i++)
    if ( tab[i ].wiek>maks)
        maks=tab[i ].wiek ;
return maks;
}

```

Dostęp do zmiennych zadeklarowanych w bibliotece `dane` biblioteka `statystyka` uzyskuje poprzez zadeklarowanie potrzebnych zmiennych ze specyfikatorem `extern`.

Innym sposobem udostępnienia na zewnątrz zmiennych `tab` i `liczba` bez konieczności ich każdorazowego importu przy pomocy modyfikatora `extern` jest umieszczenie kodu odpowiadającego za import zmiennych w pliku nagłówkowym biblioteki `dane`:

Listing 10.246. Plik dane.h

```

#ifndef _dane_
#define _dane_

typedef struct os{
    char imie[15];
    nazwisko[30];
    int wiek;
} osoba;

extern osoba *tab;
extern int liczba;

void wczytaj();
void wypisz(osoba);

#endif

```

W powyższym przypadku zmienne `tab` i `liczba` byłyby dostępne we wszystkich modułach programu używających biblioteki `dane`, niezależnie od tego czy byłyby potrzebne czy nie.

Rozwiązańe w języku C++ jest analogiczne.

Zadanie 9.3.12

Listing 10.247. plik Makefile, krótsza wersja

```

glowny: glowny.o dane.o statystyka.o
    gcc glowny.o dane.o statystyka.o -o glowny

glowny.o: glowny.c dane.h statystyka.h

```

```
statystyka.o: statystyka.h dane.h statystyka.c
dane.o: dane.h dane.c
```

Zadanie 9.3.13 Plik nagłówkowy biblioteki `dane`:

Listing 10.248. Plik dane.h

```
#ifndef _dane_
#define _dane_

typedef struct os{
    char imie[15];
    nazwisko[30];
    int wiek;
} osoba;

void wczytaj();
void wypisz();
int ile();
osoba wczytana(int);

#endif
```

Plik główny biblioteki `dane`:

Listing 10.249. Plik dane.c

```
#include <stdio.h>
#include <stdlib.h>
#include "dane.h"

static liczba=0;
static pojemnosc=0;
static osoba * tab=NULL;

void wczytaj(){
    if (liczba==pojemnosc){
        osoba * pom=tab;
        tab=malloc(2*(pojemnosc+1)*sizeof(osoba));
        int i;
        for(i=0;i<pojemnosc; i++)
            tab[i]=pom[i];
        if (pom!=NULL)
            free(pom);
        pojemnosc=2*(pojemnosc+1);
    }
    printf("Podaj imie");
    scanf("%s",&(tab[liczba].imie));
    printf("Podaj nazwisko");
    scanf("%s",&(tab[liczba].nazwisko));
```

```

    printf("Podaj wiek");
    scanf("%d",&(tab[liczba].wiek));
    liczba++;
}

void wypisz(){
    int i;
    for(i=0;i<liczba;i++)
        printf("imie: %s\nnazwisko : %s\nwiek %d\n",
               tab[i].imie, tab[i].nazwisko, tab[i].wiek);
}

int ile(){
    return liczba;
}

osoba wczytana(int i){
    return tab[i];
}

```

Dzięki użyciu specyfikatora `static` zmienne zadeklarowane w bibliotece dane nie są dostępne poza tą biblioteką (także przy użyciu specyfikatora `extern`).

Rozwiązanie w języka C++ jest podobne.

Zadanie 9.3.15 Rozwiązanie w języku C: Plik nagłówkowy biblioteki `kolo`:

Listing 10.250. Plik kolo.h

```

ifndef _kolo_
#define _kolo_

static double const pi=3.1415;

double pole(double);
double obwod(double);

#endif

```

Gdyby przy deklaracji stałej `pi` pominąć specyfikator `static` to przy próbie komplikacji programu używającego biblioteki `kolo` kompilator zgłosiłby błąd o wielokrotnej definicji stałej `pi`. Plik główny biblioteki `kolo`:

Listing 10.251. Plik kolo.c

```

#include "kolo.h"

double pole(double r){

```

```
    return pi*r*r;
}

double obwod(double r){
    return 2*pi*r;
}
```

Rozwiązanie w języku C++ różni się tylko drobnym szczegółem w pliku nagłówkowym:

Listing 10.252. Plik kolo.h

```
#ifndef _kolo_
#define _kolo_

double const pi=3.1415;

double pole(double);
double obwod(double);

#endif
```

W przeciwieństwie do języka C, stałe w języku C++ są domyślnie statyczne. Użycie w takiej sytuacji specyfikatora `static` jest dozwolone, ale nie wywoła żadnego efektu.

BIBLIOGRAFIA

- [1] Mike Banahan, Declan Brady, Mark Doran, *The C Book, Second Edition*, Addison-Wesley 1991
- [2] Krzysztof Diks, *Wstęp do programowania w języku C*, <http://wazniak.mimuw.edu.pl/>
- [3] Bruce Eckel, *Thinking in C++*. Edycja polska, Helion, Warszawa 2002
- [4] Brian W. Kernighan, Dennis M. Ritchie, *Język ANSI C*, WNT Warszawa 2000, wyd. VI, ISBN 83-204-2620-0
- [5] Bjarne Stroustrup, *Język C++*, WNT, Warszawa 2002
- [6] Kurc C na Wikibooks, <http://pl.wikibooks.org/wiki/C>
- [7] Standard języka C ISO/IEC 9899:1999
- [8] Standard Języka C++ ISO/IEC 14882:2003

Zbiór zadań z programowania w języku C/C++ cz. 2



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY



Projekt „Programowa i strukturalna reforma systemu kształcenia na Wydziale Mat-Fiz-Inf”.
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Człowiek-najlepsza inwestycja

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
WYDZIAŁ MATEMATYKI, FIZYKI I INFORMATYKI
INSTYTUT INFORMATYKI

Zbiór zadań z programowania w języku C/C++ cz. 2

Jacek Krzaczkowski



LUBLIN 2012

**Instytut Informatyki UMCS
Lublin 2012**

Jacek Krzaczkowski
**ZBIÓR ZADAŃ Z PROGRAMOWANIA W JĘZYKU C/C++
CZ. 2**

Recenzent: Grzegorz Matecki

Opracowanie techniczne: Marcin Denkowski
Projekt okładki: Agnieszka Kuśmierska

Praca współfinansowana ze środków Unii Europejskiej w ramach
Europejskiego Funduszu Społecznego

Publikacja bezpłatna dostępna on-line na stronach
Instytutu Informatyki UMCS: informatyka.umcs.lublin.pl.

Wydawca

Uniwersytet Marii Curie-Skłodowskiej w Lublinie
Instytut Informatyki
pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin
Redaktor serii: prof. dr hab. Paweł Mikolajczak
www: informatyka.umcs.lublin.pl
email: dyrii@hektor.umcs.lublin.pl

Druk

FIGARO Group Sp. z o.o. z siedzibą w Rykach
ul. Warszawska 10
08-500 Ryki
www: www.figaro.pl

ISBN: 978-83-62773-23-7

SPIS TREŚCI

PRZEDMOWA	vii
1 KLASY, OBIEKTY, DZIEDZICZENIE	1
1.1. Klasy, obiekty	2
1.2. Dziedziczenie	10
2 POLIMORFIZM	15
3 ZAAWANSOWANE PROGRAMOWANIE OBIEKTOWE	23
4 PRZECIĄŻANIE OPERATORÓW	31
5 SZABLONY	39
6 STL	49
7 WYJĄTKI	57
8 ROZWIĄZANIA	63
8.1. Rozwiązania zadań z rozdziału 1	64
8.2. Rozwiązania zadań z rozdziału 2	81
8.3. Rozwiązania zadań z rozdziału 3	85
8.4. Rozwiązania zadań z rozdziału 4	93
8.5. Rozwiązania zadań z rozdziału 5	102
8.6. Rozwiązania zadań z rozdziału 6	115
8.7. Rozwiązania zadań z rozdziału 7	121
BIBLIOGRAFIA	131

PRZEDMOWA

Niniejszy skrypt jest drugą częścią zbioru zadań pt. „Zadania z programowania C/C++”. Pierwsza część, zawierająca zadania z programowania strukturalnego, została wydana przez Instytut Informatyki UMCS w 2011r. Po roku autor oddaje w ręce czytelnika drugą część zbioru poświęconą programowaniu obiektowemu i zaawansowanym technikom języka C++, takim jak szablony i wyjątki. W przeciwieństwie do pierwszej części przeznaczonej w równym stopniu dla uczących się języka C jak i C++, druga część zbioru zadań dotyczy niemal w całości języka C++. Wyjątkiem jest kilka zadań w rozdziale 7 „Wyjątki” ilustrujących sposoby radzenia sobie w języku C z błędami pojawiającymi się w trakcie działania programu.

Język C++ jest jednym z najpopularniejszych współczesnych języków programowania. Umożliwia on programowanie zgodne z różnymi paradygmatami programowania np: programowanie strukturalne, obiektowe czy gennyrczne. Ta uniwersalność języka C++ powoduje, że niełatwo jest uwzględnić w książce wszystkie aspekty związane z programowaniem w nim. W wielu rozdziałach autor musiał dokonać selekcji poruszanych zagadnień, tak żeby z jednej strony zbiór nie rozrosł się zbytnio, a z drugiej strony żeby móc dać więcej niż po jednym zadaniu dotyczącym najważniejszych kwestii.

Na końcu książki czytelnik znajdzie rozwiązania części zadań. Znajdują się tam rozwiązania zadań reprezentatywnych dla poszczególnych zagadnień, jak również zadań z różnych powodów ciekawych.

Układając zadania autor starał się, żeby dotyczyły one realnych problemów programistycznych, a ich rozwiązania były na tyle, na ile to możliwe, krótkie. Nie było to jednak proste, szczególnie w przypadku zadań mających za zadanie pomóc w opanowaniu bardziej zaawansowanych tematów. Przy pisaniu tego skryptu autor wielokrotnie stawał przed wyborem, czy umieścić w zbiorze oczywiste w danym kontekście zadanie o długim rozwiązaniu, czy zadanie może nie tak naturalne, ale za to o krótkim i ciekawym rozwiązaniu. Autor ma nadzieję, że w większości przypadków dokonał właściwego wyboru.

W trakcie pisania skryptu został opublikowany od dawna oczekiwany nowy standard języka C++. Autor starał więc przed wyborem, czy skrypt pisać pod kątem starego czy nowego standardu. Ze względu na fakt, że znaczna część nowego standardu nie została jeszcze zaimplementowana w najpopularniejszych kompilatorach, niniejszy zbiór skupia się na starym standardzie. Jedyne odstępstwo zostało zrobione w przypadku rozdziału dotyczącego STL-a, w którym to rozdziale została umieszczona pewna liczbę zadań pozwalających przećwiczyć użycie nowych elementów biblioteki STL. Zadania te zostały specjalnie oznaczone.

Przy niektórych zadaniach znajdują się różne oznaczenia. Poniżej znajdują się wyjaśnienia używanych oznaczeń:

* trudne zadanie,

r zadanie rozwiązane w ostatnim rozdziale,

! zadanie, którego rozwiązanie z różnych powodów jest szczególnie interesujące,

C zadanie, które można rozwiązać także w języku C.

C++11 zadanie pozwalające przećwiczyć użycie elementów wprowadzonych w nowym standardzie języka C++.

ROZDZIAŁ 1

KLASY, OBIEKTY, DZIEDZICZENIE

1.1. Klasy, obiekty	2
1.2. Dziedziczenie	10

1.1. Klasy, obiekty

- 1.1 (r) Napisz klasę **poczta** zawierającą publiczne pola do przechowywania danych wiadomości przesyłanej pocztą elektroniczną: **nadawca**, **odbiorca**, **temat** i **tresc**.
- 1.2 (r) Napisz funkcję **wypisz**, która jako argument otrzymuje obiekt typu **poczta** z zadania 1.1 i wypisuje na standardowym wyjściu wartości pól otrzymanego w argumencie obiektu.
- 1.3 (r) Napisz funkcję **wczytaj**, która jako argument otrzymuje referencję do obiektu typu **poczta** z zadania 1.1 i wczytuje ze standardowego wejściu wartości pól obiektu, do którego referencję otrzymała w argumencie.
- 1.4 (r) Do klasy **poczta** z zadania 1.1 dopisz metody wczytujące i wypisujące przechowywane dane.
- 1.5 (r,!) Zdefiniuj strukturę **poczta2** o takich samych polach publicznych jak klasa **poczta** z zadania 1.1. Napisz rozwiązania zadań od 1.2 do 1.4 w wersji dla struktury **poczta2**.
- 1.6 Napisz klasę **ksiazka** zawierającą publiczne pola **tytul**, **autor**, **wydawca**.
- 1.7 Napisz funkcję **wypisz**, która jako argument otrzymuje obiekt typu **ksiazka** z zadania 1.6 i wypisuje na standardowym wyjściu wartości pól otrzymanego w argumencie obiektu.
- 1.8 Napisz funkcję **wczytaj**, która jako argument otrzymuje referencję do obiektu typu **poczta** z zadania 1.6 i wczytuje ze standardowego wejściu wartości pól obiektu, do którego referencję otrzymała w argumencie.
- 1.9 Do klasy **ksiazka** z zadania 1.6 dopisz metody wczytujące i wypisujące pola obiektu.
- 1.10 Napisz klasę **trojkat** zawierającą:
 - publiczne pola **wysokosc** i **podstawa**,
 - publiczne metody służące do wczytywania ze standardowego wejścia i wypisywania na standardowym wyjściu wartości pól obiektu,
 - publiczną metodę **pole** zwracającą jako wartość pole trójkąta o wymiarach przechowywanych w obiekcie.
- 1.11 Napisz funkcję, która dostaje w argumentach dwa obiekty typu **trojkat** z zadania 1.10 i wypisuje na standardowym wyjściu wymiary tego spośród trójkątów otrzymanych w argumentach, który ma większe pole.
- 1.12 Napisz funkcję, która dostaje w argumentach tablicę obiektów typu **trojkat** z zadania 1.10 oraz jej rozmiar i wypisuje na standardowym wyjściu wymiary tego spośród trójkątów otrzymanych w argumentach, który ma większe pole.
- 1.13 Napisz klasę **funkcja** służącą do operowania na funkcjach liniowych jednej zmiennej. Klasa **funkcja** powinna posiadać publiczne pola **a** i **b**

i publiczną metodę **wartosc**, która dla podanego argumentu **x** zwraca wartość funkcji obliczoną ze wzoru $f(x)=a*x+b$.

- 1.14 Napisz klasę **funkcja_kw** służącą do operowania na funkcjach kwadratowych jednej zmiennej. Klasa **funkcja** powinna posiadać publiczne pola **a**, **b** i **c** oraz publiczne metody:

- **wartosc** zwracającą wartość funkcji w podanym jako argument metody punkcie **x** obliczoną ze wzoru $f(x)=a*x*x+b*x+c$,
- **zero** zwracającą **true**, jeżeli przechowywana funkcja ma rzeczywiste miejsce zerowe oraz **false** w przeciwnym wypadku.

- 1.15 (r) Napisz klasę **liczba** służącą do przechowywania liczb całkowitych.

Klasa powinna udostępniać następujące metody publiczne:

- **wczytaj** wczytującą wartość liczby ze standardowego wejścia,
- **wypisz** wypisującą wartość liczby na standardowe wyjście,
- **nadaj_w** nadającą przechowywanej liczbie wartość podaną w argumencie metody,
- **wartosc** zwracającą wartość przechowywanej liczby,
- **abs** zwracającą wartość bezwzględną przechowywanej liczby.

Napisz klasę **liczba** w taki sposób, żeby dostęp do zawartych w niej danych był możliwy tylko za pośrednictwem metod tej klasy.

- 1.16 Napisz klasę **portfel**, przechowującą kwotę posiadanych pieniędzy. Klasa **portfel** powinna udostępniać następujące publiczne metody:

- **inicjuj** nadającą przechowywanej kwocie wartość 0.
- **zarobki** dodającą do przechowywanej kwoty wartość podaną w argumencie.
- **wydatki** odejmującą od przechowywanej kwoty wartość podaną w argumencie.
- **zawartosc** zwracającą jako wartość przechowywaną kwotę.

Napisz klasę **portfel** w taki sposób, żeby dostęp do zawartych w niej danych był możliwy tylko za pośrednictwem metod tej klasy.

- 1.17 (r) Napisz klasę **punkt** służącą do przechowywania współrzędnych punktu w dwuwymiarowym kartezańskim układzie współrzędnych. Napisz metody do wczytywania i wypisywania współrzędnych. Zadeklaruj wszystkie pola klasy jako prywatne.

- 1.18 (r) Napisz klasę **punkt3** służącą do przechowywania współrzędnych punktu w trójwymiarowym kartezańskim układzie współrzędnych. Napisz metody do wczytywania i wypisywania współrzędnych. Zadeklaruj wszystkie pola klasy jako prywatne.

- 1.19 (r) Napisz funkcję **rzutuj**, która otrzymuje jako argument obiekt typu **punkt3** z zadania 1.18 i zwraca jako wartość obiekt typu **punkt** z zadania 1.17 będący prostopadłym rzutem punktu otrzymanego w argumencie na płaszczyznę wyznaczoną przez dwie pierwsze współrzędne

- 1.20 (r) Do klasy **punkt** z zadania 1.17 dopisz metodę **rzutuj**, która otrzy-

muje jako argument obiekt typu `punkt3` z zadania 1.18 i przypisuje polom obiektu, na rzecz którego została wywołana, współrzędne prostopadłego rzutu punktu otrzymanego w argumencie na płaszczyznę wyznaczoną przez dwie pierwsze współrzędne.

- 1.21 Napisz klasę `zespolona` służącą do przechowywania liczb zespolonych. Udostępnij dostęp do pól tej klasy wyłącznie za pomocą publicznych metod.
- 1.22 Napisz funkcję `suma`, która dostaje jako argumenty dwa obiekty klasy `zespolone` z zadania 1.21 i zwraca jako wartość ich sumę.
- 1.23 Napisz klasę `dane_os` służącą do przechowywania danych osobowych. Klasa `dane_os` powinna posiadać prywatne pola `imie`, `nazwisko` i `adres` dostępne wyłącznie za pośrednictwem publicznych metod.
- 1.24 Do klasy `dane_os` z zadania 1.23 dopisz metodę `wypisz` wypisującą przechowywane dane osobowe.
- 1.25 Napisz klasę `tablica`, służącą do przechowywania 10-elementowej tablicy. Dostęp do poszczególnych elementów tablicy powinien być wyłącznie za pomocą publicznej metody `at`, która dla podanego indeksu zwraca referencję do odpowiedniego elementu tablicy. W przypadku podania indeksu spoza zakresu od 0 do 9 metoda `at` powinna zwrócić referencję do pierwszego elementu tablicy.
- 1.26 (r) Napisz klasę `ukryta_liczba`, która przechowuje liczbę całkowitą w prywatnym polu `liczba` i udostępnia publiczną metodę `zeruj`, przypisującą wartość 0 polu `liczba`.
- 1.27 (r) Napisz funkcję `inkrementuj`, która zwiększa o jeden wartość pola `liczba` obiektu typu `ukryta_liczba` z zadania 1.26, do którego referencję funkcja dostała w argumencie.
- 1.28 (r) Do klasy `ukryta_liczba` z zadania 1.26 dopisz metodę `inkrementuj`, która zwiększa o jeden wartość pola `liczba` otrzymanego w argumencie obiektu typu `ukryta_liczba`
- 1.29 Napisz klasę `wektor` służącą do przechowywania dziesięciowymiarowych wektorów. Klasa `wektor` powinna udostępniać następujące publiczne metody:
 - `wypisz` wypisującą wartość wektora na standardowym wyjściu,
 - `wczytaj` wczytującą wartość wektora ze standardowego wejścia,
 - `dodaj` dodającą do przechowywanego wektora wektor otrzymany w argumencie.Wszystkie pola klasy `wektor` powinny zostać zadeklarowane jako prywatne.
- 1.30 Napisz funkcję `porownaj`, która dostaje w argumentach dwa obiekty typu `wektor` z zadania 1.29 i zwraca jako wartość `true` jeżeli pierwszy z otrzymanych w argumentach wektorów jest dłuższy oraz `false` w przeciwnym wypadku.

- 1.31 Napisz klasę **odcinek** przechowującą współrzędne w dwuwymiarowym kartezjańskim układzie współrzędnych początku i końca odcinka. Klasa **odcinek** powinna udostępniać następujące publiczne metody:
- **wczytaj** wczytującą współrzędne początku i końca odcinka ze standardowego wejścia,
 - **wypisz** wypisującą współrzędne początku i końca odcinka na standardowym wyjściu,
 - **przeciecie** o argumencie typu **odcinek** zwracającą **true**, jeżeli otrzymany w argumencie odcinek przecina się z tym, którego dane są przechowywane w obiekcie.
- 1.32 Napisz funkcję zwracającą jako wartość długość odcinka, którego dane przechowywane są w otrzymanym w argumencie obiekcie typu **odcinek** z zadania 1.31.
- 1.33 (r) Napisz klasę **wskaznik** zawierającą jedno pole prywatne **wsk** typu wskaźnik do zmiennej typu **int**. Klasa **wskaznik** powinna udostępniać następujące publiczne metody:
- **utworz**, która dla otrzymanej w argumencie dodatniej liczby całkowitej **n**, rezerwuje pamięć dla **n**-elementowej tablicy o elementach typu **int** i zapisuje w polu **wsk** wskaźnik do nowo utworzonej tablicy,
 - **zwroc** zwracającą jako wartość wskaźnik przechowywany w polu **wsk**,
 - **zwolnij** zwalniającą obszar pamięci wskazywany przez pole **wsk** i nadająca temu polu wartość **NULL**,
 - **kopiuj**, która otrzymuje jako argument referencję **ref** do zmiennej typu **wskaznik** i dokonuje przypisania **ref.wsk=wsk**.
- 1.34 (r) Napisz funkcję **przepisz**, która dostaje jako argumenty wskaźnik **t** do tablicy o elementach typu **int** oraz referencję **ref** do zmiennej typu **wskaznik** z zadania 1.33 i przypisuje wskaźnik **t** do pola **wsk** obiektu, do którego referencję przechowuje zmienna **ref**.
- 1.35 (r) Do klasy **wskaznik** z zadania 1.33 dopisz:
- bezargumentowy konstruktor, przypisujący do pola **wsk** wartość **NULL**,
 - destruktor zwalniający obszar pamięci wskazywany przez pole **wsk** (o ile ma ono wartość różną od **NULL**).
- 1.36 (r) Napisz klasę **identyfikator**, która udostępnia tylko jedną publiczną bezargumentową metodę **id**. Metoda **id** powinna zwracać, za każdym razem inną, nieujemną liczbę całkowitą.
- 1.37 (r) Napisz klasę **identyfikator2**, która udostępnia tylko jedną publiczną bezargumentową metodę **id**. Metoda **id** powinna zwracać kolejne liczby całkowite począwszy od 0.
- 1.38 (r) Napisz klasę **semafor_bin**, której obiekty w każdym momencie są w jednym z dwóch stanów *wolny* lub *zajęty*. Bezpośrednio po utwo-

rzeniu obiekt typu `semafor_bin` powinien być w stanie *wolny*. Klasa `semafor_bin` powinna udostępniać następujące publiczne metody:

- `rezerwuj`, której wywołanie zmienia stan semafora z *wolny* na *zajęty* (w przypadku, gdy semafor jest już w stanie *zajęty*, metoda nie powinna zmieniać jego stanu),
- `zwolnij`, której wywołanie zmienia stan semafora z *zajęty* na *wolny* (w przypadku, gdy semafor jest już w stanie *wolny*, metoda nie powinna zmieniać jego stanu),
- `stan` zwracającą wartość `true` gdy semafor jest w stanie *wolny* i `false` w przeciwnym wypadku.

Wszystkie pola klasy `semafor_bin` powinny być prywatne.

- 1.39 Napisz klasę `semafor`, której obiekty mogą przyjmować stany ze zbioru $\{0, 1, \dots, n - 1\}$, gdzie n jest argumentem konstruktora. Bezpośrednio po utworzeniu obiekt typu `semafor` powinien być w stanie 0. Klasa `semafor` powinna posiadać następujące publiczne metody:

- `semafor(unsigned int n)` konstruktor, inicjujący obiekt, który może przyjmować stany ze zbioru $\{0, 1, \dots, n - 1\}$,
- bezargumentowy konstruktor, inicjujący obiekt, który może przyjmować stany ze zbioru $\{0, 1\}$,
- `rezerwuj`, której wywołanie zwiększa wartość stanu semafora o jeden (w przypadku, gdy semafor jest w stanie o maksymalnej wartości metoda nie powinna niczego robić),
- `zwolnij`, której wywołanie zmniejsza wartość stanu semafora o jeden (w przypadku, gdy semafor jest w stanie 0, metoda nie powinna niczego robić),
- `stan` zwracającą wartość `true` gdy semafor nie osiągnął jeszcze maksymalnej wartości stanu i `false` w przeciwnym wypadku.

Wszystkie pola klasy `semafor` powinny być prywatne.

- 1.40 Popraw klasę `semafor` z zadania 1.39 w taki sposób, żeby pamiętała identyfikatory procesów rezerwujących zasoby przy pomocy obiektów tej klasy. W tym celu metody `rezerwuj` i `zwolnij` powinny mieć całkowitoliczbowy argument `id`, w którym będą otrzymywać identyfikator procesu rezerwującego/zwalniającego zasób. Poprawiona klasa powinna ponadto posiadać publiczną metodę `wypisz` wypisującą na standardowym wyjściu identyfikatory procesów, które w danym momencie posiadają rezerwację zasobu.

- 1.41 Napisz klasę `macierz`, służącą do przechowywania macierzy kwadratowych liczb wymiernych. Klasa `macierz` powinna zawierać:

- publiczne pole `tab`, zawierające wskaźnik do macierzy,
- publiczne pole `n`, zawierające rozmiar macierzy,
- konstruktor, który dostaje w argumencie dodatnią liczbę całkowitą n i tworzy macierz o wymiarach $n \times n$,

- destruktor, który zwalnia pamięć zarezerwowaną przez obiekt.
- 1.42 Napisz klasę **staly_napis**, służącą do przechowywania napisów. Obiekt klasy **staly_napis** powinien od powstania do usunięcia przechowywać ten sam napis otrzymany w konstruktorze. Klasa **staly_napis** powinna udostępniać:
- konstruktor otrzymujący jako argument napis, który ma być przechowywany w obiekcie,
 - metodę **at**, która zwraca wartość znaku znajdującego się w napisie pod indeksem podanym w argumencie.
- Wszystkie pola w klasie **staly_napis** powinny być prywatne.
- 1.43 (r) Napisz klasę **osoba** o dwóch polach prywatnych **imie** i **nazwisko**. Klasa **osoba** powinna udostępniać następujące publiczne metody:
- konstruktor otrzymujący jako argumenty imię i nazwisko, które mają być przechowywane w obiekcie,
 - **wczytaj** wczytującą ze standardowego wejścia wartości do pól obiektu,
 - **wypisz** wypisującą na standardowym wyjściu wartości pól przechowywanych w obiekcie.
- 1.44 (r,!) Napisz funkcję, która dostaje jako argument całkowitą liczbę dodatnią **n** i zwraca jako wartość wektor **n** obiektów typu **osoba** z zadania 1.43. Elementy zwracanego wektora powinny przechowywać imię i nazwisko **Jan Kowalski**.
- 1.45 (r,!,*) Napisz klasę **tab_osob** przechowującą tablicę **tab** wskaźników do obiektów klasy **osoba** z zadania 1.43. Klasa **tab_osob** powinna udostępniać:
- konstruktor, który dostaje jako argumenty nieujemną liczbę całkowitą **n** oraz napisy **imie** i **nazwisko**, tworzy **n** obiektów klasy **osoba** zainicjowanych wartościami zmiennych **imie** i **nazwisko**, tworzy **n**-elementową tablicę **tab** oraz inicjuje jej komórkami wskaźnikami do nowo utworzonych obiektów klasy **osoba**,
 - metodę **at**, która otrzymuje jako argument nieujemną liczbą całkowitą **ind** i zwraca jako wartość referencję do obiektu wskazywanego przez element tablicy **tab** o indeksie **ind**,
 - destruktor zwalniający pamięć zarezerwowaną przez obiekt.
- 1.46 (r,*) Napisz klasę **kolejka** będącą implementacją klasycznej kolejki przechowującej liczby całkowite. Klasa **kolejka** powinna udostępniać następujące publiczne metody:
- bezargumentowy konstruktor tworzący pustą kolejkę,
 - konstruktor kopiący,
 - destruktor zwalniający pamięć zaalokowaną przez obiekt,
 - **pierwszy** zwracającą jako swoją wartość pierwszy element kolejki,
 - **usun_pierwszy** usuwającą pierwszy element kolejki,

- `dodaj_na_koniec`, dodającą na koniec kolejki liczbę całkowitą otrzymaną w argumencie,
 - `pusta` zwracającą `true` jeżeli kolejka jest pusta i `false` w przeciwnym wypadku.
- 1.47 (*) Napisz klasę `stos` będącą implementacją klasycznego stosu przechowującego liczby całkowite. Klasa `stos` powinna udostępniać następujące publiczne metody:
- bezargumentowy konstruktor tworzący pusty stos,
 - konstruktor kopiący,
 - destruktor zwalniający pamięć zaallokowaną przez obiekt,
 - `z_wierzchu` zwracająca go jako swoją wartość element z wierzchu stosu,
 - `usun_z_wierzchu` usuwający element znajdujący się na wierzchu stosu,
 - `poloz_na_stos` kładący na wierzchu stosu liczbę całkowitą otrzymaną w argumencie,
 - `pusty` zwracającą `true` jeżeli stos jest pusty i `false` w przeciwnym wypadku.
- 1.48 Dopolacz konstruktor kopiący do klasy `macierz` z zadania 1.41.
- 1.49 Napisz funkcję, która dostaje jako argumenty dwa obiekty typu `macierz` z zadania 1.41 i zwraca jako wartość kopię tego z nich, który ma mniej komórek równych 0. Jeżeli w macierzach przechowywanych w otrzymanych w argumentach obiektach jest tyle samo zer, funkcja powinna zwrócić jako wartość kopię pierwszego argumentu.
- 1.50 Zmień funkcję z zadania 1.49 w taki sposób, żeby w argumentach zamiast dwóch obiektów typu `macierz` dostawała stałe referencje do nich.
- 1.51 Napisz klasę `wektorN` służącą do przechowywania wektorów w przestrzeniach wielowymiarowych. Wszystkie pola w klasie `wektorN` powinny być prywatne. Wektor powinien udostępniać następujące metody publiczne:
- konstruktor o jednym całkowitoliczbowym argumencie `n` tworzący `n`-wymiarowy wektor,
 - destruktor zwalniający pamięć zaallokowaną przez obiekt,
 - `at` zwracający referencję do współrzędnej wektora o indeksie podanym w argumencie metody,
 - `wymiar` zwracającą liczbę wymiarów wektora przechowywanego w obiekcie.
- 1.52 Napisz funkcję, która dostaje dwa argumenty typu `wektorN` z zadania 1.51 i zwraca jako wartość sumę otrzymanych w argumentach wektorów. Jeżeli wektory mają różną liczbę wymiarów, funkcja powinna zwrócić wektor równy temu z otrzymanych w argumentach wektorów, który ma większą liczbę wymiarów.

- 1.53 Zmień funkcję z zadania 1.52 w taki sposób, żeby w argumentach zamiast dwóch obiektów typu **wektor** dostawała stałe referencje do nich.
- 1.54 Zaimplementuj klasę **napis** przechowującą napis w prywatnej tablicy znaków. Klasa ta powinna mieć następujące metody publiczne:
- bezparametryczny konstruktor tworzący pusty napis,
 - konstruktor kopiący,
 - konstruktor, którego parametrem jest napis przechowywany w tradycyjny sposób, czyli w tablicy o elementach typu **char**, w której koniec napisu jest zaznaczony przez znak o numerze 0,
 - destruktor usuwający wszystkie dynamiczne struktury danych przechowywane przez obiekt,
 - metodę **dopisz** o jednym parametrze która do istniejącego napisu „dokleja” na końcu napis podany w parametrze (utwórz dwie wersje metody **dopisz** - z parametrem typu **napis** i tablicą znaków),
 - metodę **dlug**, zwracającą długość przechowywanego napisu.
- 1.55 (*) Zaimplementuj klasę **tablica** służącą do przechowywania liczb całkowitych. Klasa **tablica** powinna udostępniać:
- bezparametryczny konstruktor tworzący pustą tablicę,
 - konstruktor kopiący,
 - destruktor usuwający wszystkie dynamiczne struktury danych przechowywane przez obiekt,
 - metodę **wartosc** zwracającą wartość komórki tablicy o indeksie podanym w argumentie tej metody,
 - metodę **przypisz** nadającą komórce tabeli o podanym w pierwszym argumentie indeksie wartość podaną w drugim argumentie.
- Obiekt tej klasy powinien zachowywać się jak tablica, która „rośnie” w miarę potrzeb. W przypadku użycia metody **przypisz** z indeksem i ($i > 0$) spoza zakresu dozwolonych indeksów tablica powinna być automatycznie powiększona do tablicy $i + 1$ elementowej.
- 1.56 (*) Napisz klasę **lista** służącą do przechowywania listy zakupów. Poszczególne pozycje listy mają się składać z dwóch elementów: nazwy towaru i ilości w jakiej planujemy go zakupić. Lista powinna udostępniać następujące publiczne metody:
- bezargumentowy konstruktor tworzący pustą listę,
 - konstruktor kopiący,
 - destruktor,
 - **dodaj** otrzymującą w argumentach nazwę towaru oraz ilość tego towaru jaką chcemy zakupić i dodającą te informacje do przechowywanej listy zakupów,
 - **wypisz** wypisującą na standardowym wyjściu wszystkie elementy listy,
 - **usun** usuwającą z listy towar podany w argumentie,

- **wyczysc** usuwającą wszystkie elementy listy.

1.2. Dziedziczenie

- 1.57 (r) Napisz klasę **figura** posiadającą publiczne pola obwód i pole. Napisz klasy pochodne klasy **figura** służące do przechowywania danych różnych konkretnych figur. Klasy pochodne powinny posiadać publiczne pola służące do przechowywania ich wymiarów (różne w zależności od rodzaju przechowywanych figur).
- 1.58 (r,!) W zadaniu 1.57 zamiast klas zdefiniuj analogiczne struktury.
- 1.59 Napisz klasę **ubranie** posiadającą publiczne pola **material** i **kolor**. Napisz klasy **spodnie**, **koszula** i **czapka** pochodne klasy **ubranie**. Klasy pochodne powinny posiadać następujące pola publiczne:
— klasa **spodnie** pola **dlugosc** i **w_pasie**,
— klasa **koszula** pola **dlugosc** i **w_klatce**,
— klasa **czapka** pole **obwod**.
- 1.60 Napisz klasę **mebel** zawierającą publiczne pola **producent** i **kolekcja** oraz klasy **krzeslo**, **stol**, **szafka** pochodne klasy **mebel**. Klasy pochodne powinny posiadać następujące pola publiczne:
— klasa **krzeslo** pole **obicie**,
— klasa **stol** pola **szerokosc** i **dlugosc**,
— klasa **szafka** pola **szerokosc**, **wysokosc** i **glebokosc**.
- 1.61 Napisz klasę **zwierze** udostępniającą publiczne pola **gatunek** oraz **imie**. Napisz klasy **zmija**, **rys**, **orzel** pochodne klasy **zwierze**, służące do przechowywania informacji o zwierzętach konkretnych gatunków.
— klasa **zmija** powinna posiadać publiczne pole **dlugosc**,
— klasa **rys** powinna posiadać publiczne pola **dlugosc** i **wysokosc**,
— klasa **orzel** powinna posiadać publiczne pole **dlugosc** i **rozpietosc_skrzydeł**
- Konstruktory klas pochodnych powinny nadawać polu **gatunek** odpowiednią wartość.
- 1.62 (r) Napisz klasę **atrakcja** posiadającą chronione pola **cena**, **nazwa** i **opis** oraz publiczne metody zwracające wartości tych pól w taki sposób, by nie można było ich modyfikować z zewnątrz klasy. Napisz klasy **kolejka**, **zamek**, **film** pochodne klasy **atrakcja**. Klasy pochodne powinny posiadać następujące pola prywatne
— **kolejka** pola **godz_odjazdu** i **godz_przyjazdu**,
— **zamek** pole **czas_zwiedzania**,
— **film** pola **czas_trwania** i **tytuł**
- Napisz w klasach pochodnych metody zwracające wartości ich nowych pól w taki sposób, żeby nie można było ich zmieniać. Zdefiniuj w klasach

pochodnych metodę `inicjuj`, nadającą wszystkim polom klasy wartości podane w argumentach metody.

- 1.63 (r) Napisz klasę `lista` służącą do przechowywania listy liczb całkowitych. Klasa `lista` powinna udostępniać następujące metody publiczne:
- bezargumentowy konstruktor tworzący pustą listę,
 - konstruktor kopiący,
 - `dodaj_przod` dodającą na początek listy liczbę całkowitą podaną w argumencie,
 - `dodaj_tyl` dodającą na koniec listy liczbę całkowitą podaną w argumencie,
 - `usun_przod` usuwającą pierwszy element listy,
 - `usun_tyl` usuwającą ostatni element listy,
 - `pierwszy_el` zwracającą wartość pierwszego elementu listy,
 - `ostatni_el` usuwającą wartość ostatniego elementu listy,
 - `pusta_lista` zwracającą `true` jeżeli lista nie zawiera żadnego elementu oraz `false` w przeciwnym wypadku.
- 1.64 (r) Napisz klasę `kolejka` z zadania 1.46 wykorzystującą do przechowywania danych prywatne pole typu `lista` z zadania 1.63.
- 1.65 (r) Napisz klasę `kolejka` z zadania 1.46 jako klasę pochodną klasy `lista` z zadania 1.63.
- 1.66 Napisz klasę `stos` z zadania 1.47 wykorzystującą do przechowywania danych prywatne pole typu `lista` z zadania 1.63.
- 1.67 Napisz klasę `stos` z zadania 1.47 jako klasę pochodną klasy `lista` z zadania 1.63.
- 1.68 (r,!,*) Napisz klasę `lepsza_lista` pochodną klasy `lista` z zadania 1.63. Wewnątrz klasy `lepsza_lista` powinna zostać zaimplementowana klasa `iterator`, której obiekty mają „wskaazywać” na pojedyncze elementy listy. Klasa `iterator` powinna udostępniać następujące publiczne metody:
- konstruktor domyślny inicjujący iterator wskazujący na pierwszy element listy podanej w argumencie konstruktora,
 - `element`, zwracająca referencję do wskazywanego elementu listy (liczby całkowitej),
 - `następny` przesuwającą iterator o jedną pozycję do przodu (to znaczy, że po wywołaniu tej metody obiekt typu `iterator` będzie wskazywał następny w kolejności element listy),
 - `poprzedni` przesuwającą iterator o jedną pozycję do tyłu (to znaczy, że po wywołaniu tej metody obiekt typu `iterator` będzie wskazywał poprzedni w kolejności element listy).
 - `początek` zwracającą `true` jeżeli `iterator` wskazuje na pierwszy element listy i `false` w przeciwnym wypadku,

- koniec zwracającą `true` jeżeli `iterator` wskazuje na ostatni element listy i `false` w przeciwnym wypadku,
- 1.69 (r) Funkcję `zeruj`, która otrzymuje jako argument referencję do obiektu klasy `lepsza_lista` i nadaje wartość 0 wszystkim elementom otrzymanej listy.
- 1.70 (r) Napisz klasę `stala1` posiadającą stałe publiczne pole `i` typu `int` o wartości 5.
- 1.71 (r) Napisz klasę `stala2` posiadającą stałe publiczne pole `d` typu `double`. Wartość pola `d` powinna być podawana przy tworzeniu obiektu klasy `stala2` jako argument konstruktora.
- 1.72 (r) Napisz klasę `stale` pochodną typu `stala2` z zadania 1.71. Klasa `stale` powinna posiadać publiczne pole `liczba` typu `stala2`. Zarówno dziedziczone pole `d` jak i publiczne pole `liczba` powinny być zainicjowane wartościami liczbowymi podanymi jako argumenty konstruktora klasy `stale`.
- 1.73 Napisz klasę `l_stala` zawierającą publiczne pole `liczba` typu `const unsigned int`. Napisz konstruktor klasy `l_stale`, który dostaje jako argument dodatnią liczbę całkowitą `n` i nadaje polu `liczba` losową wartość z zakresu od 1 do `n`.
- 1.74 Napisz klasę `l_stala2` zawierającą pole `liczba1` typu `l_stala` z zadania 1.73 i pole `liczba2` typu `const unsigned int`. Napisz konstruktor klasy `l_stala2`, który otrzymuje jako argument dodatnią liczbę całkowitą `n`, po którego wykonaniu pola `liczba1` i `liczba2` będą przechowywać tą samą losową wartość z zakresu od 1 do `n`.
- 1.75 Napisz klasę `superwektor` pochodną klasy `wektorn` z zadania 1.51, która posiada dodatkowo następujące publiczne metody:
- bezargumentowy konstruktor tworzący wektor dwuwymiarowy,
 - `dlugosc` zwracającą długość przechowywanego wektora.
- 1.76 Napisz klasę `lwektor` dziedziczącą po klasie `l_stala` z zadania 1.73 i zawierającą publiczne pole `wek` typu `wektorn` z zadania 1.51. Klasa `lwektor` powinna posiadać konstruktor, który otrzymuje jako wartość dodatnią liczbę całkowitą `n`, inicjuje wartość dziedziczonego pola `liczba` losową wartością `m` z zakresu 1 do `n` i inicjuje pole `wek` jako wektor o `m` wymiarach.
- 1.77 Napisz klasę `stala_figura` służącą do przechowywania danych figur geometrycznych. Klasa ta powinna udostępniać następujące publiczne metody:
- konstruktor, który jako argumenty otrzymuje pole, obwód i rodzaj przechowywanej figury,
 - `pole`, zwracającą pole przechowywanej figury,
 - `obwod` zwracającą obwód przechowywanej figury,

- **rodzaj** wypisującą na standardowym wyjściu rodzaj przechowywanej figury (np. kwadrat, trójkąt etc.)

Napisz klasy pochodne klasy **stala_figur** służące do przechowywania danych różnych konkretnych rodzajów figur geometrycznych (np. kwadratów, trójkątów). Klasy pochodne powinny posiadać konstruktory, których parametrami są wymiary przechowywanych figur (różne w zależności od rodzaju figury) oraz publiczne metody udostępniające poszczególne wymiary figur. Klasy powinny być napisane w taki sposób, żeby metody **pole** i **obwód** wyświetlały wartości obliczone na podstawie podanych w konstruktorach wymiarów.

- 1.78 Napisz klasę **lepszy_int** zawierającą publiczne pole **liczba** typu **int**. Dla klasy **lepszy_int** zdefiniuj konstruktor, który nadaje polu **liczba** wartość podaną w argumentem. Nie definiuj dla klasy **lepszy_int** konstruktora bezargumentowego.
- 1.79 Napisz klasę **para** przechowującą dwa pola typu **lepszy_int** z zadania 1.78. Napisz konstruktor klasy **para** nadający obu polom obiektu wartości liczb całkowitych podanych w argumentach.
- 1.80 (r) Napisz klasę **liczba** nie zawierającą żadnego pola. Zdefiniuj klasy **calkowita** i **wymierna** dziedziczące publicznie po klasie **liczba** zawierające dodatkowo pole **wartosc** typu odpowiednio **int** i **double**.
- 1.81 (r) Napisz funkcję **kopiuj**, która dostaje jako argument tablicę elementów będących wskaźnikami do typu **liczba** z zadania 1.80 i jego typów pochodnych oraz jej rozmiar, tworzy kopię otrzymanej w argumentem tablicy i zwraca ją jako swoją wartość.
- 1.82 Napisz klasę **list_figur** przechowującą listę klas pochodnych klasy **figura** z zadania 1.57. Klasa **lista_figur** powinna udostępniać następujące publiczne metody:
 - konstruktor otrzymujący jako argument maksymalną liczbę elementów listy,
 - **dodaj** dodająca na koniec listy obiekt, do którego wskaźnik metoda otrzymała w argumentem,
 - **ostatni** zwracający jako wartość wskaźnik do ostatniego spośród przechowywanych w liście obiektów,
 - **usun** usuwająca ostatni element listy,
 - **srednia**, zwracająca jako wartość średnią z obwodów przechowywanych na liście figur.

ROZDZIAŁ 2

POLIMORFIZM

- 2.1 (r) Napisz klasę **bazowa** oraz jej klasy pochodne **pochodna1** i **pochodna2**. Powyższe trzy klasy powinny udostępniać następujące metody publiczne:
- **typ_wskaznika** wypisująca na standardowym wejściu typ wskaźnika, przy pomocy którego wywołana została ta metoda,
 - **typ_obiektu** wypisująca na standardowym wyjściu typ obiektu wskażanego przez wskaźnik, przy pomocy którego wywołana została ta metoda.
- 2.2 (r) Napisz klasę **liczba** służącą do przechowywania liczb wymiernych. Klasa **liczba** powinna posiadać publiczne pole **re** typu **double** oraz następujące metody publiczne:
- wirtualną metodę **modul** zwracającą moduł przechowywanej liczby,
 - **wieksza** otrzymującą w argumencie referencję **ref** do obiektu klasy **liczba** i zwracający jako wartość **true** jeżeli moduł liczby **ref** jest większy od modułu liczby przechowywanej w obiekcie, na rzecz którego wywoływana jest metoda oraz **false** w przeciwnym wypadku.
- 2.3 (r) Napisz klasę **zespolone** pochodną klasy **liczba** z zadania 2.2 posiadającą dodatkowo publiczne pole **im** typu **int**. Przeciąż w klasie **zespolone** metodę **modul**. Czy metodę **wiekszy** też trzeba przeciążyć?
- 2.4 (r) Zaimplementuj klasę **funkcja** posiadającą publiczne pole **x** oraz czysto wirtualną metodę **wartosc**, która w klasach pochodnych będzie zwracała wartość funkcji przechowywanej w obiekcie w punkcie **x**.
- 2.5 (r) Zaimplementuj klasę **funkcja_liniowa** pochodną klasy **funkcja** z zadania 2.4. Klasa **funkcja_liniowa** powinna zawierać publiczne pola **a** i **b** oraz przeciążoną metodę **wartosc** w taki sposób, żeby zwracała wartość funkcji **a*x+b**.
- 2.6 (r) Napisz funkcję **bisekcja**, która otrzymuje jako argumenty wskaźnik do obiektu klasy pochodnej klasy **funkcja** z zadania 2.4, liczby **p**, **k** oraz **d** i szuka miejsca zerowego przekazanej w argumencie funkcji metodą bisekcji w przedziale od **p** do **k**. Funkcja ma zwrócić miejsce zeroowe z dokładnością do **d**. Jeżeli wartości funkcji na końcach zadanego przedziału są tego samego znaku to funkcja może zwrócić cokolwiek.
- 2.7 Napisz klasę **liczba** posiadającą publiczne czysto wirtualne metody **wczytaj** i **wypisz**. Napisz klasy **nint** i **ndouble** dziedziczące publicznie po klasie **liczba** i posiadające publiczne pola **wartosc** odpowiednio typu **int** i **double**. Przeciąż dla klas **nint** i **ndouble** metody **wczytaj** i **wypisz** odpowiednio wczytującą ze standardowego wejścia i wypisującą na standardowym wyjściu zawartość pola **wartosc**.
- 2.8 Napisz funkcję **wypisz_tab** otrzymującą jako argument tablicę o elementach typu **liczba*** z zadania 2.7 oraz jej rozmiar i wypisującą war-

- tości przechowywane przez obiekty wskazywane przez elementy otrzymanej w argumencie tablicy.
- 2.9 Napisz program, który wczytuje ze standardowego wejścia pewną, ustaloną przez użytkownika, liczbę wartości typów `int` i `double`, zapamiętuje je w obiektach typów `ntint` i `ndouble` z zadania 2.7 i wypisuje przy pomocy funkcji `wypisz_tab` z zadania 2.8
- 2.10 Napisz klasę `towar` posiadającą publiczne pola `nazwa`, `cena` oraz `ilosć` i wirtualną metodę `opis` wyświetlającą na standardowym wyjściu wszystkie informacje przechowywane w obiekcie.
- 2.11 Napisz następujące klasy pochodne klasy `towar` z zadania 2.10:
- `gwozdzie` posiadające dodatkowe publiczne pola `długość`, `grubość` i `rodzaj_lepka`,
 - `papier_scierny` posiadające dodatkowe publiczne pola `ziarnistość` i `szerokość`,
 - `meble` posiadające dodatkowe pole `kolekcja`.
- Wszystkie klasy pochodne klasy `towar` powinny mieć metodę `opis` przeciążoną w taki sposób, żeby wykorzystać oryginalny kod tej metody.
- 2.12 Zaimplementuj klasę `szafa` pochodną klasy `meble` z zadania 2.11. Klasa `szafa` powinna posiadać publiczne pola `wysokość`, `szerokość` i `głębokość`. Metoda `opis` powinna zostać przeciążona w taki sposób, żeby wykorzystać kod metody `opis` z klas bazowych.
- 2.13 Napisz funkcję `wypisz`, która dostaje jako argument wektor `vec` wskaźników do obiektów klasy `towar` z zadania 2.10 i wypisuje przy pomocy metody `opis` opisy wszystkich obiektów, do których wskaźniki przechowywane są w wektorze `vec`.
- 2.14 Napisz program, który wczytuje ze standardowego wejścia dane różnych towarów, przechowuje je w obiektach klas pochodnych klasy `towar` zdefiniowanych w zadaniach 2.11 oraz 2.12 i na koniec wypisuje opisy wczytanych towarów za pomocą funkcji `wypisz`.
- 2.15 Zaimplementuj klasę `czworokat` posiadającą pola chronione `a`, `b`, `c` i `d` służące do przechowywania długości boków czworokąta. Klasa `czworokat` powinna posiadać:
- metodę `wypisz` wypisującą na standardowym wyjściu długości wszystkich czterech boków,
 - czysto wirtualną metodę `pole` zwracającą jako wartość pole czworokąta,
 - czteroargumentowy konstruktor nadający polom `a`, `b`, `c` i `d` wartości otrzymane w argumentach.
- 2.16 Zaimplementuj następujące klasy pochodne klasy `czworokat` z zadania 2.15:
- `prostokat` posiadającą następujące publiczne metody:
 - dwuargumentowy konstruktor nadający polom `a` i `c` wartość otrzy-

- maną w pierwszym argumencie, zaś polom **b** i **d** wartość otrzymaną w drugim argumencie,
- **wymiary**, która otrzymuje dwa argumenty **w1** oraz **w2** i nadaje polom **a** i **c** wartość **w1**, zaś polom **b** i **d** wartość **w2**.
 - **kwadrat** posiadającą następujące publiczne metody:
 - jednoargumentowy konstruktor nadający polom **a**, **b**, **c** i **d** wartość otrzymaną w argumencie.
 - **wymiar** nadającą polom **a**, **b**, **c** i **d** jedną wartość otrzymaną w argumencie.

Klasy **prostokat** i **kwadrat** powinny mieć odpowiednio przeciążoną metodę **pole**.

- 2.17 Napisz funkcję **wypisz_pola**, która otrzymuje jako argument tablicę o elementach typu **czworokat** *****, gdzie **czworokat** jest typem zdefiniowanym w zadaniu 2.15 oraz jej rozmiar i wypisuje pola wszystkich przechowywanych w tablicy czworokątów.
- 2.18 Napisz program, który wczytuje ze standardowego wejścia wymiary kwadratów i prostokątów, przechowuje ich wymiar wykorzystując obiekty klas **prostokat** i **kwadrat** z zadania 2.16 i wypisuje ich pola przy wykorzystaniu funkcji **wypisz_pola** z zadania 2.17.
- 2.19 Napisz klasę **wierzcholek** posiadającą stałe publiczne pole **wartosc** typu **int** oraz prywatne pola **ojciec**, **lewy_syn**, **prawy_syn** będące wskaźnikami na typ **wierzcholek**
Napisz konstruktor klasy **wierzcholek**, który nadaje polu **wartosc** wartość otrzymaną jako argument, zaś pozostałym polom przypisuje wartość **NULL**.
- 2.20 (*) Napisz klasę abstrakcyjną **drzewo** zawierającą następujące metody publiczne:
- konstruktor, który dla podanej w argumencie liczby całkowitej **wartosc** tworzy drzewo, którego jedyny wierzchołek przechowuje liczbę **wartosc**
 - destruktor zwalniający zaalokowaną przez obiekt pamięć,
 - metodę **wstaw**, która otrzymuje trzy argumenty: liczbę całkowitą **wartosc**, wskaźnik **wsk** do obiektu typu **wierzcholek** z zadania 2.19 i wartość logiczną **strona**, tworzy nowy obiekt typu **wierzcholek** i dodaje go do drzewa jako syna wierzchołka wskazywanego przez **wsk**. Jeżeli zmienna **strona** jest równa **true** to nowy wierzchołek powinien zostać dodany jako lewy syn **wsk** (jeżeli **wsk** ma już lewego syna, to metoda **wstaw** nie powinna dodawać nowego wierzchołka). Analogicznie, jeżeli **strona** jest równa **false** i **wsk** nie ma prawego syna, to metoda **wstaw** powinna dodać nowy wierzchołek do drzewa jako prawego syna **wsk**.
 - czysto wirtualną metodę **nastepny**, która dla otrzymanego w argu-

mencie wskaźnika do typu `wierzcholek` z zadania 2.19 wskazującego na pewien wierzchołek drzewa zwraca wskaźnik do kolejnego według pewnej kolejności wierzchołka drzewa. Dla ostatniego wierzchołka metoda powinna zwrócić pierwszy wierzchołek według tej samej kolejności.

- 2.21 (*) Napisz klasy `drzewo_inorder`, `drzewo_preorder` `drzewo_postorder` pochodne klasy `drzewo` z zadania 2.19, w których metoda `nastepny` zwraca kolejny wierzchołek odpowiednio w kolejności inorder, preorder i postorder.
- 2.22 (r,!) Napisz abstrakcyjną klasę `kolejka` definiującą interfejs kolejki liczb całkowitych. Klasa `kolejka` powinna posiadać następujące publiczne czysto wirtualne metody:
- `pierwszy` zwracającą wartość pierwszego elementu kolejki,
 - `usun_pierwszy` usuwającą pierwszy element kolejki
 - `dodaj_na_koniec`, dodającą na koniec kolejki liczbę całkowitą otrzymaną w argumencie,
 - `pusta` zwracającą `true`, jeżeli kolejka jest pusta i `false` w przeciwnym wypadku.

Klasa powinna udostępniać także wirtualny destruktor.

- 2.23 (r,*) Napisz klasy `kolejka_listowo` i `kolejka_tablicowo` pochodne klasy `kolejka` z zadania 2.22 zawierające odpowiednio listową i tablicową implementację kolejki. Pamiętaj o zaimplementowaniu destruktorów klas `kolejka_listowo` i `kolejka_tablicowo`.
- 2.24 Napisz funkcję `oproznij`, która dostaje jako argument wskaźnik do obiektu typu `kolejka` z zadania 2.22 i wypisuje w kolejnych liniaach na standardowym wyjściu kolejne liczby przechowywane w kolejce aż do jej opróżnienia.
- 2.25 Napisz abstrakcyjną klasę `stos` definiującą interfejs stosu liczb całkowitych. Klasa `stos` powinna posiadać następujące publiczne czysto wirtualne metody:
- `z_wierzchu` zwracająca wartość elementu leżącego na wierzchu stosu,
 - `usun_z_wierzchu` usuwającą element położony na wierzchu stosu,
 - `dodaj_na_koniec` kładący na stosie liczbę całkowitą otrzymaną w argumencie,
 - `pusty` zwracającą `true`, jeżeli stos jest pusty i `false` w przeciwnym wypadku.
- Klasa powinna udostępniać także wirtualny destruktor.
- 2.26 (*) Napisz klasy `stos_listowo` i `stos_tablicowo` pochodne klasy `stos` z zadania 2.25 zawierające odpowiednio listową i tablicową implementację stosu. Pamiętaj o zaimplementowaniu destruktorów klas `stos_listowo` i `stos_tablicowo`.

2.27 (*) Napisz funkcję **wartosc**, która dostaje w argumencie wskaźnik do klasy **stos** z zadania 2.25 i zwraca wartość wczytanego ze standardowego wejścia wyrażenia arytmetycznego zapisanego w odwrotnej notacji polskiej. Zakładamy, że wczytywane wyrażenie składa się z oddzielonych pojedynczymi spacjami liczb całkowitych i operatorów „+”, „-”, „*”, „/”. Funkcja powinna wykorzystać w obliczeniach otrzymany w argumencie **stos**.

2.28 (*) Napisz program kalkulator. W programie wykorzystaj klasy:

- **dzialanie**, abstrakcyjną klasę posiadającą:
 - publiczne stałe pole **nazwa** przechowujące nazwę działania,
 - publiczne stałe pole **n** przechowujące arność działania,
 - czysto wirtualną publiczną metodę **wynik**, która otrzymuje jako argument wektor **n** liczb typu **double** i zwraca jako wartość **wynik** działania na nich.
 - **kalkulator** posiadającą:
 - prywatne pole **wartosc** typu **double** zawierające obecną wartość przechowywaną w kalkulatorze,
 - wektor **dzial** referencji do obiektów typu **dzialanie**,
 - publiczną metodę **obliczenia** będącą właściwą metodą odpowiadającą za działanie kalkulatora i komunikację z użytkownikiem. Metoda **obliczenia** w kolejnych krokach działania powinna wyświetlać bieżącą wartość pola **wartosc** oraz dawać użytkownikowi następujące możliwości do wyboru:
 - nadanie polu **wartosc** wartości wczytanej ze standardowego wejścia,
 - nadanie polu **wartosc** wartości 0,
 - wykonanie działania reprezentowanego przez jeden z obiektów, do którego referencje przechowuje wektor **dziel** i zapisanie jego wyniku do pola **wartosc**. Pierwszym argumentem działania powinna być dotychczasowa wartość pola **wartosc**, pozostałe argumenty powinny zostać wczytane ze standardowego wejścia.
 - zakończenie działania metody **obliczenia**,
 - **dodaj_dzialanie**, która dostaje w argumencie referencję do obiektu klasy pochodnej klasy **dzialanie** i dodaje obiekt tej klasy do wektora **dzial**,
 - **dodawanie**, **odejmowanie**, **mnozenie**, **dzielenie**, **logarytm_naturalny** klasy pochodne klasy **dzialania**.
- 2.29 (*) Napisz program do gry w kółko i krzyżyk do trzech.
- Napisz klasę **plansza**, służącą do przechowywania stanu gry. Klasa **plansza** powinna udostępniać metodę **wypisz** wypisującą na standardowym wyjściu stan planszy, metodę **stan** zwracającą stan pola

o podanych w argumentach indeksach oraz metodę `wykonaj_ruch` aktualizującą stan planszy po wykonaniu ruchu podanego jako argument.

- Napisz klasę `ruch` służącą do przechowywania pojedynczych ruchów w grze.
 - Napisz klasę abstrakcyjną `gracz` posiadającą czysto wirtualną metodę `wybierz_ruch`, która dla podanych w argumentach: stałej referencji do obiektu klasy `plansza` i informacji, co ma postawić gracza (kółko czy krzyżyk), zwraca jako wartość ruch gracza.
 - Napisz klasę `gracz_człowiek` pochodną klasy `gracz`, której metoda `wybierz_ruch` dla podanej planszy zwraca ruch wczytany ze standardowego wejścia.
 - Napisz klasę `gra`, która posiada prywatny obiekt typu `plansza` oraz publiczną wirtualną metodę `graj`, która dostaje jako argumenty dwa wskaźniki do obiektów `gracz`, przeprowadza grę między nimi wywołując na zmianę metodę `wybierz_ruch` obu obiektów i aktualizując stan planszy, a na koniec zwraca 1, jeżeli wygrał `gracz` podany w pierwszym argumencie, -1 gdy wygrał `gracz` podany w drugim argumencie oraz 0 w przypadku remisu. Jeżeli metoda `wybierz_ruch` któregoś z obiektów zwróci niepoprawny ruch, metoda `graj` powinna ją wywołać jeszcze raz.
- 2.30 (*) Napisz program do gry w kółko i krzyżyk pomiędzy dwojgiem ludzi z wykorzystaniem klas z zadania 2.29.
- 2.31 (*) Napisz klasę `gracz_komputer` pochodną klasy `gracz` z zadania 2.29, w której metoda `wybierz_ruch` zwraca ruch na losowo wybrane wolne pole. Napisz program do gry w kółko i krzyżyk pomiędzy człowiekiem a komputerem wykorzystując klasę `gracz_komputer` oraz klasy z zadania 2.29.
- 2.32 (***) Napisz program do gry w warcaby. Napisz klasy analogiczne do tych z zadania 2.29. Porównaj swój program z rozwiązaniami zadania 2.30.

ROZDZIAŁ 3

ZAAWANSOWANE PROGRAMOWANIE OBIEKTOWE

- 3.1 (r) Napisz klasę **stale** zawierającą publiczne stałe statyczne pola **pi** i **e**.
- 3.2 (r) Napisz klasę **liczba** zawierającą publiczne statyczne pole **licz** typu **int**. Wartość pola **licz** zainicjuj wartością 0.
- 3.3 Napisz klasę **tablica** zawierającą jako pole 5-elementową statyczną tablicę liczb całkowitych. Elementy tablicy zainicjuj wartościami kolejnych liczb pierwszych począwszy od 2.
- 3.4 Napisz klasę **stala_tablica** zawierającą jako pole 5-elementową statyczną tablicę stałych liczb całkowitych. Elementy tablicy zainicjuj wartościami kolejnych liczb pierwszych począwszy od 2.
- 3.5 Napisz klasę **staly_wektor** zawierającą jako pole 5-elementowy statyczny wektor stałych liczb całkowitych. Elementy wektora zainicjuj wartością 0.

Listing 3.1.

```

1 class zespolone{
2   public:
3     double re ,im;
4     zespolone() {}
5     zespolone(double r , double i): re(r),im(i) {}
6   };

```

- 3.6 (r) Napisz klasę **zesp** posiadającą publiczne statyczne metody **dodaj**, **odejmij**, **pomnoz**, **podziel**, które otrzymują jako argumenty dwa obiekty typu **zespolone** z tabelki 3.3.1 i zwracają jako wartość wynik odpowiedniego działania. Klasa **zesp** powinna posiadać także publiczną statyczną stałą **i** typu **zespolone** o wartości $\sqrt{-1}$.
- 3.7 (r) Napisz program, który wyświetla na standardowym wyjściu 100 pierwszych elementów ciągu zdefiniowanego w następujący sposób:

$$a_n = \begin{cases} \sqrt{-1} & n = 1 \\ \frac{(2 \cdot a_{n-1} + 10 \cdot \sqrt{-1})}{a_{n-1}} & n > 1 \end{cases}$$

W programie wykorzystaj klasę **zesp** z zadania 3.6.

- 3.8 (r) Stwórz przestrzeń nazw **zesp**. W tej przestrzeni nazw zdefiniuj funkcje **dodaj**, **odejmij**, **pomnoz**, **podziel**, które otrzymują jako argumenty dwa obiekty typu **zespolone** z zadania 1.21 i zwracają jako wartość wynik odpowiedniego działania. Ponadto zdefiniuj w przestrzeni nazw **zesp** stałą typu **zespolone** o wartości $\sqrt{-1}$.
- 3.9 (r) Rozwiąż zadanie 3.6 wykorzystując elementy przestrzeni nazw **zesp** z zadania 3.6.

-
- 3.10 Napisz klasę **zaokraglij**, zawierającą statyczne publiczne metody służące do zaokrąglania liczb wymiernych do liczb całkowitych. Klasa **zaokraglij** powinna udostępniać metody **najblzsza**, **podloga** i **sufit**.
- 3.11 Napisz program, który wczytuje ze standardowego wejścia liczby wymierne i wypisuje je po zaokrągleniu na standardowym wyjściu. W programie wykorzystaj klasę **zaokraglij** z zadania 3.10. O tym, ile liczb powinno zostać wczytanych i w jaki sposób zaokrąglonych (w górę, w dół czy do najbliższej liczby całkowitej), powinien decydować użytkownik.
- 3.12 Stwórz przestrzeń nazw **zaokraglij**, zawierającą funkcje służące do zaokrąglania liczb wymiernych do liczb całkowitych. Przestrzeń nazw **zaokraglij** powinna zawierać funkcje **najblzsza**, **podloga** i **sufit**.
- 3.13 Rozwiąż zadanie 3.11 z wykorzystaniem przestrzeni nazw **zaokraglij** z zadania 3.12.
- 3.14 (r,!) Napisz klasę **policzona** posiadającą publiczną metodę **ile** zwracającą jako wartość liczbę istniejących w danym momencie obiektów tej klasy.
- 3.15 Napisz klasę **unikalne**, której obiekty posiadają stałe prywatne pole **id** typu **unsigned int**. Klasę **unikalne** zaimplementuj w taki sposób, żeby każdy obiekt tego typu w programie w momencie powstawania otrzymywał inną wartość pola **id**.
- 3.16 (r,!) Klasę **zesp** z zadania 3.6 oraz klasę **zaokraglij** z zadania 3.10 napisz w taki sposób, żeby nie można było stworzyć obiektów tych klas.
- 3.17 (r,!) Napisz klasę **finalna**, po której nie można dziedziczyć.
- 3.18 (r,!) Zaimplementuj klasę **dynamiczna** przechowującą tablicę liczb całkowitych w taki sposób, żeby obiektów tej klasy nie dało się stworzyć jako zmiennych automatycznych.
- 3.19 (r,!) Zdefiniuj klasę **tab_info** służącą do przechowywania parametrów tworzonej tablicy liczb całkowitych. Klasa **tab_info** powinna przechowywać informacje o rozmiarze tworzonej tablicy, początkową wartość elementów tablicy oraz o tym, ile razy mogą być zmieniane wartości poszczególnych elementów tablicy. Obiekt klasy **tab_info** powinien także przechowywać informację, czy wartości poszczególnych parametrów były ustawiane czy nie. Klasa **tab_info** powinna udostępniać następujące publiczne metody:
- **rozmiar**, która nadaje polu przechowującemu rozmiar tablicy wartość podaną w argumencie i zwraca jako wartość referencję do obiektu, na rzecz którego została wywołana metoda,
 - **wartosc**, która otrzymuje jako argument początkową wartość elementów tablicy i zwraca jako wartość referencję do obiektu, na rzecz którego została wywołana metoda,
 - **zmiany**, która otrzymuje jako argument liczbę dozwolonych zmian wartości poszczególnych elementów tablicy. Otrzymanie w argumen-

cie wartości -1 oznacza, że wartości elementów tablicy mogą być zmieniane dowolną liczbę razy. Metoda powinna zwrócić jako wartość referencję do obiektu, na rzecz którego została wywołana.

- 3.20 (r,*) Napisz klasę **tablica** służącą do przechowywania tablicy liczb całkowitych. Klasa ta powinna umożliwiać ograniczenie liczby zmian wartości poszczególnych elementów przechowywanej tablicy. Klasa **tablica** powinna udostępniać następujące publiczne metody:

- konstruktor, który dostaje jako argument stałą referencję do obiektu **param** typu **tab_info** z zadania 3.19, zawierającego parametry tworzącej tablicy. W przypadku niezdefiniowania któregoś z parametrów przyjmujemy jego wartość domyślną. Domyślny rozmiar tablicy, to 100, domyśla wartość początkowa elementów tablicy to 0, zaś domyślona dozwolona liczba zmian wartości każdego z elementów to 10.
- destruktor zwalniający pamięć zajmowaną przez obiekt,
- **podaj_w**, która zwraca wartość elementu tablicy o indeksie podanym w parametrze,
- **nadaj_w**, która otrzymuje jako argumenty indeks **i** oraz liczbę całkowitą **w** i nadaje elementowi tablicy o indeksie **i** wartość **w**. Jeżeli element o indeksie **i** był zmieniany już maksymalną dozwoloną liczbę razy, to funkcja nie powinna nic robić.
- **licznik**, która podaje, ile jeszcze razy można zmieniać wartość elementu o indeksie **i**. Jeżeli nie ma ograniczeń na liczbę zmian wartości elementów przechowywanej tablicy, to funkcja powinna zwrócić wartość -1 .
- **rozmiar** zwracającą jako wartość rozmiar tablicy.

- 3.21 (r,!) Napisz bezargumentową funkcję **alokuj**, która alokuje w pamięci obiekt typu **tablica** z zadania 3.20 przechowujący 50-elementową tablicę. Komórki alokowanej tablicy powinny być wypełnione zerami, zaś wartość każdej z komórek tablicy powinna móc być zmieniona dokładnie jeden raz. Funkcja **alokuj** powinna zwrócić jako wartość wskaźnik do zaalokowanego obiektu.

- 3.22 Zdefiniuj klasę **punkt** przeznaczoną do przechowywania współrzędnych punktu na płaszczyźnie. Napisz klasę **opcje** przechowującą parametry wyświetlania danych punktu na standardowym wyjściu. Uwzględnij możliwość wyświetlania współrzędnych w kartezjańskim i biegunowym układzie współrzędnych oraz wyświetlania dokładnych lub zaokrąglonych wartości współrzędnych. Klasa **opcje** powinna udostępniać następujące publiczne metody:

- **kartezjanski**, która ustawia opcję wyświetlania współrzędnych w kartezjańskim układzie współrzędnych i zwraca jako wartość referencję do obiektu, na rzecz którego została wywołana,

- **biegunowy**, która ustawia opcję wyświetlania współrzędnych w biegunowym układzie współrzędnych i zwraca jako wartość referencję do obiektu na rzecz, którego została wywołana,
 - **dokładność**, która dostaje jako argument liczbę **d** typu **double**, ustawia opcję wyświetlania współrzędnych zaokrąglonych w dół do najbliższej wielokrotności **d** i zwraca jako wartość referencję do obiektu na rzecz, którego została wywołana.
- 3.23 Napisz funkcję, która otrzymuje dwa argumenty obiekty: **p** typu **punkt** oraz **op** typu **opcje** i wyświetla współrzędne punktu **p** w sposób zdefiniowany w **op**. Typy **punkt** i **opcje** zostały zdefiniowane w zadaniu 3.22.
- 3.24 (**r,!**) Napisz klasę **napis** zawierającą publiczne pole typu **string** oraz publiczną bezargumentową metodę **stała** zwracającą wartość **true** gdy jest wywoływana na rzecz stałego obiektu i **false** w przeciwnym razie.
- 3.25 (**r,!**) Napisz klasę **napis2** publicznie dziedziczącą po klasie **string** i posiadającą publiczną bezargumentową metodę **stała** zwracającą wartość **true** gdy jest wywoływana na rzecz stałego obiektu i **false** w przeciwnym razie. Dla klasy **napis2** zdefiniuj następujące konstruktory:
- bezargumentowy, inicjujący obiekt przechowujący pusty napis,
 - otrzymujący w argumencie zmienną typu **const string&** i inicjujący obiekt przechowujący napis otrzymany w argumencie,
 - otrzymujący w argumencie zmienną typu **const char[]** i inicjujący obiekt przechowujący napis otrzymany w argumencie,
- Czym w użyciu różni się klasa **napis2** od klasy **napis** z zadania 3.24.
- 3.26 Napisz klasę **prywatna_liczba**, która zawiera prywatne pole **liczba** typu **int** oraz publiczne metody **wypisz** i **wczytaj**. Metodę **wypisz** zaimplementuj w taki sposób, żeby można ją było wywołać również na rzecz obiektów stałych.
- Napisz funkcję **wypisz_vec**, która dostaje jako argument referencję do wektora o elementach typu **const prywatna_liczba** i wypisuje na standardowym wyjściu liczby przechowywane we wszystkich elementach wektora. Do wypisywania wartości użyj metody **wypisz**.

Listing 3.2.

```

class bazowa{
  public:
    virtual ~bazowa() {
      }
    };
  class pochodna: public bazowa{
    };

```

- 3.27 (r) Napisz funkcję `porownaj`, która otrzymuje jako argumenty dwa wskaźniki typu `const bazowa *`, gdzie typ `bazowa` został zdefiniowany w listingu 3.2. Funkcja powinna zwrócić jako wartość `true`, jeżeli wskazywane przez argumenty obiekty są tego samego typu oraz `false` w przeciwnym wypadku.
- 3.28 (r) Napisz funkcję, która otrzymuje jako argument wskaźnik typu `bazowa *` i zwraca wartość typu `pochodna *`, gdzie typy `bazowa` i `pochodna` zostały zdefiniowane w listingu 3.2. Jeżeli argument funkcji wskazuje na obiekt typu `pochodna`, to funkcja powinna zrzutować go na typ `pochodna *` i zwrócić jako wartość. W przeciwnym wypadku funkcja powinna zwrócić wartość `NULL`.

Listing 3.3.

```

1  class liczba{
2     public:
3         virtual void wypisz ()=0;
4     };
5
6  class rzeczywista: public liczba{
7     public:
8         double wartosc;
9         void wypisz (){
10            cout<<wartosc<<endl;
11        }
12    };
13
14 class zespolona: public liczba{
15     public:
16         double wartR, wartU;
17         void wypisz (){
18             cout<<wartR<<" "


---



```

- 3.29 Napisz funkcję `wypisz`, która dostaje jako argumenty tablicę o elementach typu `liczba*` i jej rozmiar, która wywołuje metodę `wypisz` na rzecz obiektów typu `rzeczywista` wskazywanych przez elementy otrzymanej w argumencie tablicy (elementy tablicy wskazujące na obiekty innych typów powinny zostać zignorowane). Typy `liczba` i `rzeczywista` zostały zdefiniowane w listingu 3.3

Listing 3.4.

```

1  class liczba2{
2     public:
3         virtual ~liczba2 () {}
4     };

```

```
6 class rzeczywista2: public liczba2{  
7     public:  
8         double wartosc;  
9  
10    };  
11  
12 class zespolona2: public liczba2{  
13     public:  
14         double wartR, wartU;  
15    };
```

- 3.30 (r) Napisz funkcję **wypisz**, która dostaje jako argumenty tablicę o elementach typu **liczba2*** i jej rozmiar, która wypisuje na standardowe wyjście wartości przechowywane w obiektach typu **rzeczywista2** wskazywanych przez elementy otrzymanej w argumencie tablicy (elementy tablicy wskazujące na obiekty innych typów powinny zostać zignorowane). Typy **liczba2** i **rzeczywista2** zostały zdefiniowane w listingu 3.4

ROZDZIAŁ 4

PRZECIĄŻANIE OPERATORÓW

- 4.1 (r) Napisz klasę **zespolone** służącą do przechowywania liczb zespolonych. Przeciąż dla niej operatory odpowiadające działaniom arytmetycznym oraz operatory `<< i >>` tak, żeby wartości typu **zespolone** można było wczytywać i wypisywać przy pomocy standardowych strumieni.
- 4.2 Napisz klasę **n_int** posiadającą dwa prywatne pola: **liczba** typu **int** oraz **okr** typu **bool**. Wartość pola **okr** powinna być **true** gdy wartość pola **liczba** jest określona. Klasa **n_int** powinna udostępniać:
- bezargumentowy konstruktor nadający polu **okt** wartość **false**,
 - konstruktor nadający polu **liczba** wartość otrzymaną w argumentie, zaś polu **okr** wartość **true**,
 - przeciążone operatory arytmetyczne. W przypadku gdy wartość któregoś z argumentów operatora nie jest określona (jego pole **okr** ma wartość **false**), wartość wyniku również powinna być nieokreślona.
 - operator `>>` przeciążony w taki sposób, żeby umożliwiał wczytywanie ze standardowego wejścia wartości pola **liczba** (po pomyślnym wczytaniu wartości pola **liczba** pole **okr** powinno otrzymać wartość **true**),
 - operator `<<` przeciążony w taki sposób, żeby umożliwiał wypisywanie na standardowym wyjściu wartości pola **liczba**. W przypadku, gdy pole **okr** ma wartość **false**, nic nie powinno zostać wypisane.
- 4.3 (*) Napisz klasę **d_int** służącą do przechowywania dużych liczb całkowitych (niemożliwych do przechowania w typach standardowych). Obiekty klasy **d_int** powinny przechowywać liczby jako tablice zmiennych typu **unsigned char**. Klasa **d_int** powinna posiadać:
- konstruktor bezargumentowy inicjujący obiekt przechowujący wartość 0,
 - konstruktor inicjujący obiekt przechowujący liczbę podaną w argumentie,
 - konstruktor kopiący,
 - destruktor
 - przeciążone operatory porównania (`==`, `<`),
 - przeciążone operatory arytmetyczne,
 - przeciążony operator przypisania,
 - przeciążone operatory `<< i >>` tak, żeby umożliwić proste wczytywanie i wypisywanie na standardowym wyjściu zmiennych typu **d_int**.
- 4.4 (r,*) Napisz klasę **tablica** służącą do przechowywania dynamicznych tablic liczb całkowitych. Klasa **tablica** powinna posiadać prywatne pole **tab** typu **int *** oraz **rozmiar** typu **unsigned int** i udostępniać:
- konstruktor bezargumentowy inicjujący obiekt nie przechowujący

-
- tablicy (inicjujący pole `tab` wartością `NULL`, a pole `rozmiar` wartością 0),
- konstruktor tworzący tablicę o liczbie elementów podanej w argumencie,
 - konstruktor kopiący,
 - destruktor,
 - metodę `resize` zmieniającą rozmiar przechowywanej tablicy do rozmiaru podanego w argumencie. Zawartość starej tablicy powinna pozostać przepisana na początek nowej tablicy. W przypadku zmniejszenia rozmiaru tablicy, powinno być przepisane tyle elementów starej tablicy, ile się zmieści w nowej.
 - operator przypisania,
 - operator `[]` zwracający referencję do poszczególnych elementów tablicy (powinienny to być jedyny sposób dostępu z zewnątrz klasy do elementów tablicy). Przeciąż ten operator zarówno w wersji dla zmiennych jak i stałych obiektów.
- 4.5 (*) Napisz klasę `wektor` służącą do przechowywania wektorów. Klasa `wektor` powinna zawierać prywatne pola `tab` typu `double` oraz `rozmiar` typu `unsigned int`. Ponadto klasa `wektor` powinna posiadać:
- konstruktor `wektor(int n)` tworzący n-wymiarowy wektor,
 - konstruktor kopiący,
 - przeciążone operatory dodawania wektorów i mnożenia wektorów przez skalar,
 - przeciążony operator przypisania,
 - przeciążone operatory porównania `==` oraz `<=` (jeden wektor jest mniejszy lub równy od drugiego jeżeli jest mniejszy lub równy na wszystkich współrzędnych).
 - przeciążony operator `[]` zwracający referencję do współrzędnej wektora o podanym w argumencie indeksie (przeciąż ten operator w wersji dla obiektów zmiennych i stałych).
- 4.6 (r) Napisz klasę `napis` służącą do przechowywania napisów. Klasa powinna zawierać:
- bezargumentowy konstruktor tworzący pusty napis,
 - konstruktor inicjujący obiekt wartością tablicy znaków (`char *`) otrzymanej w argumencie. Zakładamy, że otrzymany w argumencie napis zakończony jest znakiem o numerze 0 (tak jak to jest w napisach w języku C),
 - konstruktor kopiący,
 - destruktor,
 - przeciążony operator przypisania,
 - przeciążony operator `==`,

- przeciążony operator `[]`, zwracający wartości poszczególnych znaków napisu,
 - przeciążony operator `+`, który ma działać jak operator katenacji (ma zwracać jako wynik sklejony napis). Zaimplementuj ten operator tak aby dało się go składać z samym sobą,
 - metodę `rozmiar` zwracającą liczbę znaków zawartych w przechowywanym napisie,
 - metodę `wypisz` wypisującą przechowywany napis na standardowym wyjściu.
- 4.7 (!) Napisz funkcję `niekopiowa` w taki sposób, żeby obiektów tej klasy nie dało się kopować ani za pomocą konstruktora kopiącego, ani za pomocą operatora przypisania.
- 4.8 (r,!,*) Napisz abstrakcyjną klasę `komperator` posiadającą jedynie czysto wirtualny operator `()` o dwóch argumentach typu `const napis&` z zadania 4.6. Napisz klasę `alfabetyczna` pochodną klasy `komperator` służącą do porównywania obiektów klasy `napis` z zadania 4.6. Przeciąż w klasie `alfabetyczna` operator `()` w taki sposób, żeby zwracał on wartość `true` jeżeli pierwszy argument jest referencją do wcześniejszego w kolejności alfabetycznej napisu i `false` w przeciwnym wypadku.
- 4.9 (r,!,*) Napisz funkcję `sortuj`, która otrzymuje jako argument tablicę elementów typu `napis` z zadania 4.6, rozmiar tablicy, oraz referencję do obiektu klasy `komperator` 4.8 i sortuje rosnąco podaną w argumentie tablicę używając do porównywania elementów tablicy otrzymanego w drugim argumentem obiektu klasy `komperator`.
- 4.10 Napisz program, który tworzy tablicę o elementach typu `napis` z zadania 4.6, nadaje jej elementom wartości wczytane ze standardowego wejścia, sortuje elementy tablicy przy pomocy funkcji `sortuj` z zadania 4.9 z argumentem typu `alfabetyczna` zdefiniowanego w zadaniu 4.8 i wypisuje posortowane elementy tablicy na standardowym wyjściu.
- 4.11 (!,*) Napisz klasę służącą do generowania liczb pseudolosowych. Klasa ta powinna przeciążony operator `()`, który powinien zwracać dwa rodzaje wartości pseudolosowych:
- liczbę całkowitą z zakresu `[0,n-1]`, gdzie `n` jest argumentem podanym przez użytkownika,
 - liczbę z zakresu `(0,1)`, jeżeli użytkownik nie poda żadnego argumentu.
- Klasa powinna posiadać konstruktor, którego argument powinien inicjować generator. Do generowania kolejnych wartości z zakresu `(0,1)` użądź funkcji $f(x)=1-x*x$.
- 4.12 (r,!) Napisz klasę `macierz` służącą do przechowywania tablic dwuwymiarowych o elementach typu `double`. Klasa `macierz` powinna udostępniać:

- konstruktor inicjujący macierz o podanych w argumentach wymiarach,
- przeciążony operator () , który otrzymuje jako argumenty dwie nieujemne liczby całkowite i zwraca jako wartość referencję do elementu przechowywanej w obiekcie tablicy o indeksach podanych w argumentach.

Dlaczego w zadaniu każemy przeciążyć operator () , a nie [] ?

- 4.13 Stwórz typ służący do przechowywania wartości logicznych logiki trójwartościowej (prawda, fałsz, wartość nieznana). Dla zdefiniowanego typu przeciąż operatory logiczne tak, aby działały zgodnie z następującymi tabelkami:

$a \&& b$	prawda	wart. nieznana	fałsz
prawda	prawda	wart. nieznana	fałsz
wart. nieznana	wart. nieznana	wart. nieznana	fałsz
fałsz	fałsz	fałsz	fałsz

$a \parallel b$	prawda	wart. nieznana	fałsz
prawda	prawda	prawda	prawda
wart. nieznana	prawda	wart. nieznana	wart. nieznana
fałsz	prawda	wart. nieznana	fałsz

a	$!a$
prawda	fałsz
wart. nieznana	wart. nieznana
fałsz	prawda

- 4.14 (r) Napisz struktury **punkt2D** i **punkt3D** służące do przechowywania współrzędnych punktów w przestrzeniach odpowiednio dwu- i trójwymiarowych. Przeciąż operator rzutowania w taki sposób, żeby można było zrzutować obiekty typu **punkt3D** na typ **punkt2D**. Rzutowanie powinno „obcinąć” trzecią współrzędną punktu.
- 4.15 Zaimplementuj dla klasy **zespolone** z zadania 4.1 operator rzutowania na typ **double**. Operator ten powinien zwracać część rzeczywistą rzutowanej liczby.
- 4.16 Zaimplementuj dla klasy **d_int** z zadania 4.3 operator rzutowania na typ **int**. W przypadku gdy wartość typu **d_int** jest zbyt duża lub zbyt mała by ją zapisać w zmiennej typu **int** operator powinien zwrócić wartość **INT_MAX** lub **INT_MIN** w zależności czy rzutowana wartość jest dodatnia czy ujemna.
- 4.17 Dla typu zdefiniowanego w zadaniu 4.13 przeciąż operator rzutowania do typu **bool**. Operator powinien rzutować wartość „prawda” na wartość **true**, zaś wartość „nieokreślona” i wartość „fałsz” na **false**.

4.18 (r,!,*) Dla struktury

```

1 struct elisty {
2     struct element {
3         element * nastepny, poprzedni;
4         int i;
5     };
6
7     element * wsk, *pierwszy, *ostatni;
8
9 };

```

przeciąż operatory:

- **++** przesuwający wskaźnik **wsk** na następny element listy (nadający mu wartość pola **nastepny** wskazywanej struktury). W przypadku gdy **wsk** wskazuje na ostatni element listy operator nie powinien zmieniać jego wartości.
- **--** przesuwający wskaźnik **wsk** na wcześniejszy element listy (nadający mu wartość pola **poprzedni** wskazywanej struktury). W przypadku gdy **wsk** wskazuje na pierwszy element listy operator nie powinien zmieniać jego wartości.

Przeciąż prefiksową i postfiksową wersję powyższych operatorów. Wersje prefiksowe przeciążanych operatorów powinny zwracać jako wartość referencję do otrzymanego w argumencie obiektu, a wersja postfiksowa kopię pierwotnej wartości obiektu.

4.19 Utwórz typ **iterator** umożliwiający dostęp do poszczególnych komórek tablicy o elementach typu **int**. Typ **iterator** powinien zawierać prywatne pole **wskaźnik** typu **int *** oraz posiadać:

- konstruktor, który ustawia wartość pola **wskaźnik** na wartość podaną w argumencie,
- przeciążony unary operator ***** zwracający jako wartość referencję do liczby całkowitej wskazywanej przez pole **wskaźnik**,
- przeciążone operatory inkrementacji **++** w taki sposób, żeby inkrementowały wartość pola **wskaźnik**,
- przeciążone operatory dekrementacji **--** w taki sposób, żeby odejmowały jedne od wartości pola **wskaźnik**,
- przeciążone operatory porównania **==** oraz **<** w taki sposób, żeby porównywały wartość pól **wskaźnik** porównywanych obiektów.

4.20 (!) Do klasy **tablica** z zadania 4.4 dopisz metody publiczne:

- **początek** zwracającą obiekt typu **iterator** z zadania 4.19, w którym pole **wskaźnik** wskazuje na pierwszą komórkę tablicy wskazywanej przez pole **tab** obiektu, na rzecz którego została wywołana metoda.
- **koniec** zwracającą obiekt typu **iterator** z zadania 4.19, w któ-

rym pole **wskaznik** wskazuje na komórkę pamięci znajdująca się tuż za ostatnią komórką tablicy wskazywanej przez pole **tab** obiektu, na rzecz którego została wywołana metoda.

- 4.21 (**r, !,***) Napisz klasę **n_int** zawierającą publiczne pole typu **int**. Klasa **n_int** powinna zawierać statyczną metodę **wypisz** wypisującą liczbę dynamicznie zaalokowanych pojedynczych obiektów typu **n_int** (nie licząc tych wchodzących w skład dynamicznych tablic) oraz liczbę dynamicznie zaalokowanych tablic o elementach typu **n_int**. Aby to umożliwić, przeciąż w odpowiedni sposób operatory **new** i **delete**.
- 4.22 (**!,***) Napisz klasę **n_char** zawierającą publiczne pole typu **char**. Przeciąż operatory **new** i **delete** dla tego typu w taki sposób, żeby wszystkie dynamicznie alokowane obiekty typu **n_char** znajdowały się w jednym wcześniejszym zaalokowanym obszarze pamięci. Zakładamy, że liczba znajdujących się jednocześnie w pamięci obiektów typu **n_char** nie przekracza 1000. Uniemożliw dynamiczne alokowanie tablic obiektów typu **n_char**.

ROZDZIAŁ 5

SZABLONY

W języku polskim często zamiennie używa się pojęć argument i parametr na określenie argumentów funkcji. Aby uniknąć nieporozumień w tym rozdziale i w całym skrypcie, konsekwentnie używamy pojęć argumentów funkcji i parametrów szablonu.

- 5.1 **(r)** Napisz szablon funkcji, która otrzymuje jako argumenty zmienne `a,b,c` o typie będącym parametrem szablonu i zwraca wartość `a-b+c`.
- 5.2 Napisz szablon funkcji, która wczytuje ze standardowego wejścia wartość zmiennej o typie będącym parametrem funkcji i zwraca ją jako wartość funkcji.
- 5.3 Napisz szablon funkcji, która otrzymuje jako argumenty dwie wartości o typie podanym jako parametr szablonu i zwraca wartość mniejszego z otrzymanych argumentów.
- 5.4 **(r)** Napisz szablon funkcji, która otrzymuje jako argumenty tablice o elementach typu podanego jako parametr szablonu oraz rozmiar tablicy i zwracającą jako wartość element tablicy o najmniejszej wartości.
- 5.5 **(r)** Napisz szablon funkcji, która otrzymuje jako argumenty dwie referencje do zmiennych typu będącego parametrem szablonu i zamienia wartościami zmienne, do których referencje otrzymała w argumentach.
- 5.6 **(r)** Napisz szablon funkcji `wypisz`, która dostaje jako argument `wsk` wskaźnik wskazujący na wartość o typie będącym parametrem szablonu i wypisującą na standardowym wyjściu wartość wskazywaną przez argument.
- 5.7 **(r)** Napisz szablon funkcji `wypisz_zakres`, która otrzymuje dwie zmienne `pocz` i `kon`, których typ jest parametrem szablonu i wywołuje funkcję `wypisz` z zadania 5.6 dla każdego elementu z zakresu od `pocz` do `kon-1`. Zakładamy, że kolejne elementy zakresu można otrzymać poprzez inkrementację zmiennej `pocz`.
- 5.8 **(r,!)** Napisz szablon funkcji, która otrzymuje jako argumenty tablice o elementach typu będącego parametrem szablonu oraz rozmiar tablicy i wypisuje wszystkie elementy otrzymane w argumencie tablicy przy pomocy funkcji `wypisz_zakres` z zadania 5.7.
- 5.9 **(r)** Rozwiąż zadania od 5.6 do 5.8 w taki sposób, żeby parametrami były typy argumentów (a nie typy wskazywane przez argumenty).
Napisz szablon funkcji `wypisz_vec`, która otrzymuje jako argumenty stałą referencję do wektora (typ `vector`) o elementach typu będącego parametrem szablonu i wypisuje wszystkie elementy otrzymanego w argumencie wektora przy pomocy funkcji `wypisz_zakres` z zadania 5.7.
- 5.10 **(r,!)** Napisz szablon operatora `>`, który działa dla wszystkich klas, dla których zdefiniowany jest operator `<`. Operator `>` powinie zwracać `true` wtedy i tylko wtedy gdy operator `<` zwraca `false`.

- 5.11 (r) Napisz szablon dwuargumentowej funkcji, której argumenty są typów T1 i T2 będących parametrami szablonu, która zwraca `true` wtedy i tylko wtedy, gdy oba argumenty są tego samego typu. Zakładamy, że T1 i T2 są klasami zawierającymi metody wirtualne .
- 5.12 (r) Napisz szablon funkcji `minimum` o dwóch parametrach T1 i T2. Zakładamy, że klasa T2 ma przeciążony operator () , który dla dwóch argumentów typu T1 zwraca wartość `true` wtedy i tylko wtedy, gdy pierwszy z argumentów jest mniejszy. Argumentami funkcji `minimum` powinny być tablica o elementach typu T1, rozmiar tablicy oraz obiekt typu T2. Funkcja powinna zwrócić wartość najmniejszego elementu otrzymanej w pierwszym argumencie tablicy względem porządku wyznaczonego przez ostatni argument.
- 5.13 (r,!) Napisz funkcję, która dostaje jako argumenty tablicę liczb całkowitych oraz jej rozmiar i wypisuje na standardowym wyjściu najmniejszy i największy element otrzymanej w argumencie tablicy. Do znalezienia największego i najmniejszego elementu użyj funkcji `minimum` z zadania 5.12.
- 5.14 Napisz szablon funkcji `dla_kazdego` o dwóch parametrach T1 i T2. Zakładamy, że klasa T2 ma przeciążony operator () tak, że jego argumentem może być zmienna typu T1. Argumentami funkcji `dla_kazdego` powinny być tablica t o elementach typu T1, rozmiar tablicy oraz stała referencja f do obiektu typu T2. Funkcja powinna wywołać operator () na rzecz obiektu f dla wszystkich elementów tablicy t.
- 5.15 Napisz program, który wczytuje ze standardowego wejścia tablicę napisów, zamienia wszystkie małe litery w napisach na duże i wypisuje na standardowym wyjściu efekty swojej pracy. Wykorzystaj do przetwarzania tablicy napisów szablon funkcji `dla_kazdego` z zadania 5.14.
- 5.16 (r) Napisz szablon funkcji `przepisz` o jednym parametrze n typu `unsigned int`, która dostaje jako argumenty dwie tablice dwuwymiarowe n na n typu `int [n] [n]` i przepisuje zawartość tablicy otrzymanej w pierwszym argumencie do tablicy otrzymanej w drugim argumencie.
- 5.17 (r) Napisz klasę `tablica` służącą do przechowywania tablicy liczb typu `int`. Klasa `tablica` powinna posiadać publiczne pola `tab` typu `int *` oraz `rozmiar_tab` typu `unsigned int` i udostępniać:
- konstruktor, który otrzymuje jako argument dodatnią liczbę całkowitą n, tworzy n-elementową tablicę o elementach typu `int`, przypisuje do pola `tab` wskaźnik do nowo utworzonej tablicy i przypisuje polu `rozmiar` wartość n,
 - destruktor,
 - szablon metody `sortuj`, której parametrem jest typ T o operatrze () przeciążonym w taki sposób, że dla dwóch liczb podanych w argumencie zwraca `true` wtedy i tylko wtedy, gdy pierwsza liczba

jest mniejsza według pewnego ustalonego porządku. Metoda `sortuj` powinna otrzymywać w argumencie obiekt typu `T` i sortować elementy przechowywanej tablicy zgodnie z porządkiem wyznaczonym przez operator `()` otrzymanego w argumencie obiektu.

- 5.18 Napisz klasę `obudowany_vector`, która posiada prywatne pole `wektor` typu `vector<double>` oraz publiczny szablon metody `kopiuj`, który dostaje jako argument stałą referencję do obiektu typu `vector<T>` i przekopiuje zawartość otrzymanego w argumencie wektora do pola `wektor`. Zakładamy, że `T` może być dowolnym standardowym typem liczbowym.
- 5.19 Napisz klasę `statystyka`, która posiada:
- prywatne pola `maksimum`, `minimum` i `srednia` typu `double`,
 - publiczne metody `ZwrocMaks`, `ZwrocMin`, `ZwrocSred` zwracające wartość pól odpowiednio `maksimum`, `minimum` i `srednia`,
 - szablon metody `wczytaj` otrzymujący jako argument zmienną `wek` typu `const vector<T>&`, gdzie `T` jest parametrem szablonu i nadający połom `maksimum`, `minimum` i `srednia` odpowiednio minimalną, maksymalną i średnią wartość elementów wektora `wek`.
- 5.20 W klasie `statystyka` z zadania 19 przeciąż szablon metody `wczytaj` tak, żeby działał także dla podanych w argumencie dwuwymiarowych kwadratowych tablic automatycznych o elementach standardowych typów liczbowych i różnych rozmiarach.
- 5.21 Napisz klasę `plik` służącą do zapisywania danych do plików tekstowych. Klasa `plik` powinna posiadać:
- konstruktor otwierający do pisania plik, do którego ścieżkę dostępu otrzymał w argumencie,
 - prywatne pole przechowujące strumień skojarzony z otworzonym plikiem,
 - szablon metody `zapisz` zapisujący do pliku zawartość dwuwymiarowej automatycznej tablicy kwadratowej o typie elementów oraz wymiarach będących parametrami szablonu. Zakładamy, że dla typu elementów tablicy przeciążony jest operator `<<` umożliwiający pisanie do strumienia. Wszystkie elementy jednego wiersza tablicy powinny zostać zapisane w jednym wierszu pliku oddzielone pojedynczymi odstępami. Każdy wiersz tablicy powinien być zapisany w oddzielnym wierszu pliku.
- 5.22 (r) Napisz szablon klasy `liczba`, posiadającego prywatne pole `wart` typu będącego parametrem szablonu. Klasa `liczba` powinna posiadać następujące publiczne metody:
- `wczytaj` wczytującą ze standardowego wejścia wartość pola `wart`,
 - `wartosc` zwracającą wartość pola `wart`.
- 5.23 Napisz szablon klasy `tablica` o jednym parametrze `n` typu `unsigned int`.

Klasa **tablica** powinna posiadać publiczne pole będące n-elementową automatyczną tablicę o elementach typu **int**.

- 5.24 Napisz szablon funkcji **przepisz** o jednym parametrze **n** typu **unsigned int**, która dostaje jako argumenty zmienną **a** typu **tablica<n>&** i b typu **const tablica<n>&**, i przepisuje zawartość obiektu, do którego referencję przechowuje **b** do zmiennej, do której referencję zawiera **a**. Szablon **tablica** został zdefiniowany w zadaniu 5.33.
- 5.25 (r) Napisz szablon klasy **para** o dwóch parametrach **T1** i **T2**, który posiada dwa pola publiczne **pierwsze** typu **T1** i **drugie** typu **T2**. Zdefiniuj dla klasy **para**:
- dwuargumentowy konstruktor nadający połom tej klasy wartości podane w argumentach,
 - operator **<** w taki sposób, że element **a** jest mniejszy od elementu **b** wtedy i tylko wtedy, gdy **a.pierwsze < b.pierwsze** lub gdy **a.pierwsze = b.pierwsze** i **a.drugie < b.drugie**.
- 5.26 Napisz szablon **tablica** posiadającej publiczne pola **tab** typu **T***, gdzie typ **T** jest parametrem szablonu oraz stałe pole **rozmiar** typu **unsigned int**. Klasa **tablica** powinna posiadać konstruktor, który otrzymuje w argumencie zmienną **n** typu **unsigned int**, przypisuje wartość **n** stałej **rozmiar**, tworzy dynamicznie **n**-elementową tablicę o elementach typu **T** i przypisuje wskaźnik do niej do pola **tab**. Zdefiniuj dla typu **tablica** konstruktor kopiący, destruktor oraz operator przypisania.
- 5.27 (r,!) Napisz szablon klasy **tablica2** posiadający dwa parametry: typ **T** i wartość **n** typu **unsigned int**. Klasa **tablica2** powinna posiadać jako publiczne pole **n**-elementową automatyczną tablicę **tab** o elementach typu **T** i statyczną stałą **rozmiar** typu **int**, która powinna zostać zainicjowana wartością **n**. Czy powinniśmy zdefiniować dla typu **tablica2** konstruktor kopiący, destruktor albo operator przypisania?
- 5.28 (r,!) Napisz szablon klasy **wektor** posiadający jako parametr **n** typu **unsigned int**. Klasa **wektor** powinna posiadać jako publiczne pole **n**-elementową automatyczną tablicę o elementach typu **double** przechowująca współrzędne wektora. Dla klasy **wektor** przeciąż operatory dodawania i odejmowania wektorów oraz mnożenia wektora przez skalar (wartość typu **double**).
- 5.29 (r,!) Napisz szablon klasy **punkt** posiadający jeden parametr **n** typu **unsigned int**, służący do przechowywania współrzędnych punktów w **n**-wymiarowym kartezjańskim układzie współrzędnych. Zakładamy, że współrzędne są przechowywane w publicznej **n**-wymiarowej automatycznej tablicy o elementach typu **double**.
- 5.30 (r,!) Napisz szablon funkcji **zrzutuj** o jednym parametrze **n**, typu **unsigned int**, która otrzymuje jako argument stałą referencję do obiek-

tu typu `punkt<n>` z zadania 5.29 i zwraca jako wartość obiekt typu `punkt<n-1>` powstały z argumentu funkcji poprzez pominięcie ostatniej współrzędnej.

- 5.31 (r,!) Do szablonu klasy `punkt` z zadania 5.29 dopisz konstruktor, który dostaje jako argument stałą referencję `p` do obiektu klasy `punkt<n+1>` i przepisuje do tworzonego punktu wszystkie współrzędne `p` poza ostatnią. Czym różnią się w użyciu: zdefiniowany tym w tym zadaniu konstruktor i zdefiniowana w poprzednim zadaniu funkcja `zrzutuj`.
- 5.32 Do szablonu klasy `punkt` z zadania 29 dopisz szablon konstruktora o parametrze `m`, który dostaje jako argument stałą referencję `p` do obiektu klasy `punkt<m>` i:
 - jeżeli `m` jest większe lub równe `n`, to przepisuje do tworzonego punktu pierwsze `n` współrzędnych punktu `p`,
 - jeżeli `m` jest mniejsze od `n`, to pierwszym `m` współrzędnym nowo tworzonego punktu przypisuje wartości współrzędnych z punktu `p`, natomiast pozostałym współrzędnym nadaje wartość 0.
- 5.33 Napisz szablon klasy `macierz`, która posiada dwa parametry `m` i `n` typu `unsigned int`. Klasa `macierz` powinna przechowywać publiczną dwuwymiarową automatyczną tablicę o wymiarach `m` na `n` i elementach typu `int`.
- 5.34 Napisz szablon funkcji `kopiuj` o dwóch parametrach `n` i `m` typu `unsigned int`, która otrzymuje jako argumenty zmienną `a` typu `macierz<n,m>&` i `b` typu `const macierz<n,m>&`, i przepisuje zawartość obiektu do którego referencję przechowuje `b` do zmiennej, do której referencję zawiera `a`. Szablon `macierz` został zdefiniowanych w zadaniu 5.33.
- 5.35 Przeciąż operator `*` dla szablonu klasy `macierz` z zadania 5.33. Operator `*` powinien zwracać jako wartość iloczyn macierzy otrzymanych w argumentach.
- 5.36 Zdefiniuj szablon klasy `porownywacz` o jednym parametrze `T` i przeciążonym operatorze `()`. Klasę `porownywacz` zdefiniuj w taki sposób, żeby mogła służyć jako drugi parametr (a obiekt tej klasy jako trzeci argument) w szablonie funkcji `minimum` z zadania 5.12. Operator `()` przeciąż w taki sposób, żeby funkcja `minimum` zwracała jako wartość największy, zgodnie z porządkiem na wartościach `T` określonych przez operator `<`, element otrzymanej w argumencie tablicy .
- 5.37 Zdefiniuj szablon klasy `porownywacz2` o jednym parametrze `T` i przeciążonym operatorze `()`. Klasę `porownywacz2` zdefiniuj w taki sposób, żeby mogła służyć jako drugi parametr (a obiekt tej klasy jako trzeci argument) w szablonie funkcji `minimum` z zadania 5.12. Klasa `porownywacz2` powinna posiadać konstruktor o argumencie typu `bool`, którego wartość powinna decydować o sposobie działania operatora `()`. W obiektach

utworzonych z podaniem jako argumentu konstruktora wartości `true` operator `()` powinien działać jak operator `<`. Podanie do konstruktora wartości `false` powinno sprawić, że operator `()` działa jak operator `>`.

- 5.38 (r,*) Napisz szablon abstrakcyjnej klasy `lista`, którego parametrem jest typ `T` elementów przechowywanych w liście. Szablon klasy `lista` powinien udostępniać następujące czysto wirtualne publiczne metody (opisy dotyczą zachowania poszczególnych metod w klasach pochodnych klasy `lista`):

- `wstaw_z_przodu` wstawiającą na początek listy element podany w argumencie metody,
- `pierwszy` zwracającą wartość pierwszego elementu listy,
- `usun_pierwszy` usuwającą pierwszy element listy,
- `wstaw_z_tylu` wstawiającą na koniec listy element podany w argumencie metody,
- `ostatni` zwracającą wartość ostatniego elementu listy,
- `usun_ostatni` usuwającą ostatni element listy,
- `pusta` zwracającą `true` gdy lista jest pusta i `false` w przeciwnym wypadku.

Pamiętaj o zdefiniowaniu w klasie `lista` wirtualnego destruktora.

- 5.39 (r,*) Napisz szablon klasy `lista_wskaz` pochodnej klasy `lista` z zadania 5.38. Klasa `lista_wskaz` powinna przechowywać elementy listy w liście wskaźnikowej. Klasa `lista_wskaz` powinna posiadać bezargumentowy konstruktor oraz destruktor.
- 5.40 Napisz szablon klasy `lista_tab` pochodnej klasy `lista` z zadania 5.38. Klasa `lista_tab` powinna przechowywać elementy listy w powiększanej w miarę potrzeby tablicy. Klasa `lista_tab` powinna posiadać bezargumentowy konstruktor oraz destruktor.
- 5.41 Napisz szablon klasy `stos` będący implementacją stosu elementów o typie podanym jako parametr szablonu. Klasa `stos` powinna wykorzystywać szablon klasy `lista_wskaz` z zadania 5.38 do przechowywania stosu. Klasa `stos` powinna udostępniać następujące metody publiczne:
- `z_wierzchu` zwracającą jako swoją wartość element znajdujący się na wierzchu stosu (pierwszy na liście),
 - `usun_z_wierzchu` usuwający element znajdujący się na wierzchu stosu,
 - `poloz_na_stos` kładący na stosie (wstawiający na początek listy) element o wartości podanej w argumencie,
 - `pusty` zwracającą `true` jeżeli stos jest pusty i `false` w przeciwnym wypadku.
- 5.42 (r) Napisz szablon klasy `kolejka` o jednym parametrze `T`. Szablon `kolejka` powinien być implementacją kolejki o elementach typu `T` prze-

chowywanych w obiekcie klasy pochodnej klasy `lista` z zadania 5.38. Klasa `kolejka` powinna udostępniać następujące metody publiczne:

- konstruktor, który w argumencie otrzymuje wskaźnik do obiektu klasy pochodnej klasy `lista` z zadania 5.38, w którym mają być przechowywane elementy kolejki,
- bezargumentowy konstruktor inicjujący kolejkę przechowującą swoje elementy w obiekcie typu `lista_wskaz` z zadnia 5.39,
- `pierwszy` zwracająca wartość pierwszego elementu kolejki,
- `usun_pierwszy` usuwającą pierwszy element kolejki,
- `dodaj_na_koniec`, dodającą na koniec kolejki wartość otrzymaną w argumencie
- `pusta` zwracającą `true` jeżeli kolejka jest pusta i `false` w przeciwnym wypadku.

5.43 (r) Napisz szablon klasy `tablica` o jednym parametrze typie T. Klasa `tablica` powinna posiadać prywatne pola `tab` typu `T *` i `roz` typu `unsigned int` oraz udostępniać:

- bezargumentowy konstruktor przypisujący polu `roz` wartość 0.
- konstruktor, który dostaje w argumencie wartość `n`, tworzy `n`-elementową tablicę o elementach typu T, przypisuje do pola `tab` wartość wskaźnika do nowo utworzonej tablicy i nadaje polu `roz` wartość `n`.
- konstruktor kopiący,
- destruktor,
- bezargumentową metodę `rozmiar` zwracającą wartość pola `roz`,
- przeciążony operator przypisania.
- przeciążony operator `[]`, który dla podanego indeksu `i` zwraca referencję do komórki tablicy `tab` o indeksie `i`,
- operator `[]` przeciążony jako stała metoda i zwracający stałą referencję do elementu tablicy `tab` o podanym w argumencie indeksie.

5.44 (r) Stwórz klasę `napis` będącą konkretyzacją szablonu `tablica` z zadania 5.43 dla typu `char`. Dla klasy `napis` przeciąż operator + tak, żeby działał jak operator katenacji.

5.45 (*) Dla klasy `napis` z zadania 5.44 przeciąż operatory `<<` i `>>` w taki sposób, żeby umożliwić wczytywanie ze standardowego wejścia i wypisywanie na standardowym wyjściu wartości obiektów klasy `napis` przy pomocy strumieni `cin` i `cout`.

5.46 (r,!,*) Dla szablonu klasy `tablica` z zadania 5.43 przeciąż operator porównania `==` w taki sposób, żeby zwracał `true` wtedy i tylko wtedy, gdy dwie tablice mają ten sam rozmiar i taką samą zawartość. Operator porównania przeciąż jako metodę szablonu klasy `tablica`. Przeciąż go tak, by można nim było porównywać obiekty typów będących konkretyzacją szablonu `tablica` różnymi porównywalnymi typami (czyli

przechowujących tablice o elementach różnych typów, których wartości można porównywać np. tablice o elementach typu `int` i `double`).

- 5.47 (*) Napisz szablon klasy `macierz` o dwóch parametrach `n` i `m` typu `int` wyznaczających wymiary macierzy. Szablon klasy `macierz` powinien posiadać publiczne pole `M` będące automatyczną tablicą o elementach typu `double` i wymiarach `n` na `m`. Napisz szablon metody `pomnoz` otrzymującej jako argument stałą referencję do obiektu typu `macierz` i zwracającą jako wartość wynik mnożenia obiektu, na rzecz którego została wywołana metoda i macierzy otrzymanej w argumencie metody.
- 5.48 (**r,!**) Napisz program wczytujący ze standardowego wejścia pewną podaną przez użytkownika liczbę wartości typu `int` i wypisującą ją w kolejności od najmniejszej do największej. Do przechowywania wczytywanych liczb wykorzystaj szablon `tablica` z zadania 5.43.
Rozwiążanie zadania 5.43 umieść w oddzielnym module znajdującym się w oddzielnym pliku.
- 5.49 Napisz program wczytujący ze standardowego wejścia i przechowujący podaną przez użytkownika liczbę wartości typu `napis` z zadania 5.44. Do przechowywania napisów wykorzystaj szablon `tablica` z zadania 5.43. Rozwiązań zadań 5.43 oraz 5.44 i 5.45 umieść w dwóch osobnych modułach znajdujących się w oddzielnych plikach.
- 5.50 Napisz program wczytujący ze standardowego wejścia pewną podaną przez użytkownika liczbę wartości typu `int` i wypisujący je na standardowym wyjściu w odwrotnej kolejności niż kolejność wczytania. W rozwiążaniu wykorzystaj rozwiązania zadań 5.38, 5.39 i 5.41 . Kod programu rozmieść w kilku plikach. Stwórz trzy moduły: pierwszy zawierający rozwiązanie zadania 5.38, drugi z rozwiązaniem zadania 5.39 oraz trzeci z rozwiązaniem zadania 5.41.

ROZDZIAŁ 6

STL

W niniejszym rozdziale znajdują się zadania pozwalające przećwiczyć wykorzystanie różnych elementów biblioteki STL. Podobne zadania dotyczące operacji na plikach oraz na napisach czytelnik znajdzie w pierwszej części skryptu.

- 6.1 (r) Napisz program, który wczytuje ze standardowego wejścia dodatnie liczby całkowite. Program powinien skończyć wczytywanie liczb po wczytaniu liczby 0. Na koniec program powinien wypisać na standardowym wyjściu wszystkie wczytane liczby łącznie z zerem.

W rozwiązaniu zadania wykorzystaj któryś z zdefiniowanych w STL-u kontenerów.

- 6.2 Napisz program, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą n , następnie n liczb całkowitych i wypisuje na standardowym wyjściu wczytane liczby.

W rozwiązaniu zadania wykorzystaj któryś z zdefiniowanych w STL-u kontenerów.

- 6.3 (r,!) Napisz funkcję, która otrzymuje jako argument referencję do obiektu klasy `vector<int>` i:

- odwraca kolejność elementów w otrzymanym w argumencie kontenerze,
- sortuje rosnąco elementy otrzymanego w argumencie kontenera,
- sortuje malejąco elementy otrzymanego w argumencie kontenera,
- sortuje rosnąco elementy otrzymanego w argumencie kontenera względem ich wartości bezwzględnych,
- sortuje rosnąco elementy otrzymanego w argumencie kontenera względem ich reszty z dzielenia przez 1000.

- 6.4 (r,C++11) Jak różniłoby się rozwiązanie zadania 6.3, gdyby argumentem funkcji była referencja do obiektu klasy `array<int,n>`, gdzie n jest pewną liczbą całkowitą. Czy można zaimplementować rozwiązanie tak, żeby działało dla obiektów klasy `array<int,n>` o dowolnym n ?

- 6.5 (r) Napisz klasę `macierz`, która służy do przechowywania macierzy liczb wymiernych. Klasa `macierz` powinna udostępniać następujące publiczne metody:

- konstruktor, który otrzymuje dwie liczby całkowite i inicjuje macierz o wymiarach podanych w argumentach,
- przeciążony operator `()`, który dla liczb całkowitych i, j podanych jako argumenty zwraca referencję do elementu macierzy znajdującego się i -tej kolumnie i j -wierszu.

Do implementacji klasy `macierz` wykorzystaj typ `vector`.

- 6.6 (C++11) Napisz klasę `macierz1010` służącą do przechowywania macierzy liczb wymiernych o wymiarach 10 na 10. Klasa `macierz1010` powinna udostępniać przeciążony operator `()`, który dla liczb całkowitych

i, j podanych jako argumenty zwraca referencję do elementu macierzy znajdującego się i -tej kolumnie i j -wierszu.

Do implementacji klasy `macierz` wykorzystaj typ `array`.

- 6.7 Napisz klasę `tablica3D` analogiczną do klasy `macierz` z zadania 6.5 ale przechowującą trójwymiarową tablicę przy użyciu obiektów typu `vector`.
- 6.8 Napisz program, który wczytuje ze standardowego wejścia dodatnie liczby całkowite n i m ($m \leq n$) oraz n liczb całkowitych, a następnie wypisuje na standardowym wyjściu m -tą pod względem wielkości liczbę.
- 6.9 (r) Napisz program, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą n a następnie wczytuje n par liczb całkowitych i wstawia je do listy (liczby nieujemne program powinien wstawić na koniec listy, a ujemne na początek listy). Na końcu program powinien wypisać na standardowym wyjściu wszystkie wczytane elementy w kolejności ich występowania na liście.

W rozwiązaniu wykorzystaj kontener `list`.

- 6.10 Zaimplementuj klasę `kolejka` z zadania 1.46 przy użyciu kontenera `list`.
- 6.11 Napisz klasę służącą do przechowywania listy napisów. Klasa ta powinna udostępniać następujące publiczne metody:
- `wczytaj` wczytującą ze standardowego wejścia napis i umieszczającą go na końcu przechowywanej listy,
 - `wypisz` wypisującą na standardowym wyjściu dziesięć pierwszych elementów listy. Metoda `wypisz` powinna usunąć z listy wypisane elementy.

W rozwiązaniu zadania użyj szablonu `queue`.

- 6.12 (r) W rozwiązaniu zadania 6.11 użyj szablonu klasy `queue` w taki sposób, żeby używał kontenera `list` w miejsce `dequeue`.
- 6.13 (*) Napisz program wczytujący ze standardowego wejścia wyrażenie arytmetyczne zapisane w odwrotnej notacji polskiej i wypisujący na standardowym wyjściu jego wartość. Użyj w programie szablonu `stack` z STL-a.
- 6.14 Napisz klasę `bufor` udostępniającą publiczną metodę `dodaj` dodającą do bufora napis podany w argumencie metody. W przypadku dodania do bufora dziesiątego napisu metoda `dodaj` powinna wypisać na standardowym wyjściu przechowywane w buforze napisy w kolejności ich dodawania i opróżnić bufor.
- 6.15 Klasę `bufor` z zadania 6.14 napisz w taki sposób, żeby funkcja bufor zamiast na standardowe wyjście zapisywała przechowywane napisy do strumienia podanego w argumencie konstruktora.
- 6.16 Napisz funkcję, która dostaje jako argument napis zawierający wyrażenie arytmetyczne zapisane w odwrotnej notacji polskiej i zwraca wartość

wyrażenia otrzymanego w argumencie. Zakładamy, że liczby oraz operatory arytmetyczne w wyrażeniu są oddzielone pojedynczymi znakami odstępu. Do manipulowania na napisie użyj klasy **stringstream**.

- 6.17 Napisz program, który wczytuje ze standardowego wejścia nazwę pliku, dodatnią liczbę całkowitą n oraz n liczb całkowitych, a następnie zapisuje w sposób tekstowy przeczytane liczby do pliku o podanej nazwie w kolejności od najmniejszej do największej.
- 6.18 Napisz program, który wczytuje ze standardowego wejścia nazwę pliku tekstowego, czyta zawartość pliku (zakładamy, że w pliku znajdują się liczby oddzielone białymi znakami oraz że cała zawartość pliku zmieści się w pamięci programu) i wypisuje na standardowym wyjściu medianę z liczb przeczytanych z pliku.
- 6.19 (r) Napisz klasę **rekrutacja** służącą do przechowywania i wyszukiwania danych kandydatów na pracowników. Klasa **rekrutacja** powinna przechowywać następujące dane kandydatów: imię, nazwisko, pesel, telefon kontaktowy oraz liczbę punktów zdobytych podczas rekrutacji i udostępniać następujące publiczne metody:
- **dodaj** dodającą do listy kandydata o danych podanych jako argumenty metody,
 - **najlepszy** zwracającą jako wartość strukturę zawierającą dane kandydata o maksymalnej liczbie punktów zdobytych podczas rekrutacji spośród kandydatów przechowywanych w bazie. Jeżeli więcej niż jeden kandydat ma maksymalną liczbę punktów metoda powinna zwrócić któregokolwiek z nich.
 - **usun** usuwającą z bazy dane kandydata o największej liczbie punktów zdobytych podczas rekrutacji. W przypadku gdy więcej niż jeden kandydat ma maksymalną liczbę punktów metoda powinna usunąć tego z nich, którego zwróciłaby metoda **najlepszy**.

W rozwiązaniu użyj szablonu **priority_queue**.

- 6.20 Napisz klasę **rekrutacja** w taki sposób, żeby **priority_queue** używała **dequeue**.
- 6.21 Napisz funkcję sortującą elementy otrzymanej w argumencie listy liczb całkowitych przy wykorzystaniu szablonu klasy **priority_queue**.
- 6.22 (*) Zaimportuj klasę **rekrutacja** z zadania 6.19 używając wektora oraz operacji **make_heap**, **push_heap** oraz **pop_heap** zamiast klasy **priority_queue**.
- 6.23 Napisz klasę **punkty** przechowującą współrzędne punktów w trójwymiarowym kartezjańskim układzie współrzędnych udostępniającą następujące publiczne metody:
- **dodaj** dodającą do przechowywanych punktów punkt o współrzędnych podanych w argumencie funkcji,

-
- **najblizszy** podającą współrzędne przechowywanego punktu najbliższego początkowi układu współrzędnych.
- 6.24 (r) Napisz klasę **słownik** służącą do przechowywania słownika ortograficznego. Klasa **słownik** powinna udostępniać następujące publiczne metody:
- **dodaj** - dodającą do słownika wyraz podany w argumencie metody,
 - **znajdz** - sprawdzającą czy podany w argumencie wyraz należy do słownika (czyli czy jest poprawnie zapisanym słowem). Metoda powinna zwrócić **true** jeżeli podany w argumencie wyraz należy do słownika i **false** w przeciwnym wypadku.
- 6.25 (r) Do klasy **słownik** z zadania 6.24 dopisz metodę **zakres**, która dla dwóch słów **slowo1** i **slowo2** podanych w argumentach wypisuje na standardowym wyjściu wszystkie słowa należące do słownika znajdujące się w porządku alfabetycznym po **slowo1** a przed **slowo2**.
- 6.26 Napisz funkcję **suma**, która otrzymuje jako argumenty stałe referencje do dwóch zbiorów (typ **set**) o elementach typu **int** i zwraca jako wartość zbiór zawierający teoriomnogościową sumę zbiorów otrzymanych w argumentach.
- 6.27 Napisz funkcję **suma** analogiczną do tej z zadania 26, ale działającą na multizbiorach. Zgodnie z definicją sumy na multizbiorach liczba wystąpień poszczególnych elementów w wynikowym multizbiorze powinna być sumą liczby wystąpień tych elementów w multizbiorach będących argumentami sumy.
- 6.28 Napisz klasę **encyklop** służącą do przechowywania haseł encyklopedii. Klasa **encyklop** powinna udostępniać następujące publiczne metody:
- **dodaj**, która dodaje do encyklopedii hasło i jego definicję podane jako argumenty,
 - **znajdz**, która zwracaj jako wartość definicję hasła podanego w argumencie. Jeżeli dane hasło nie należy do encyklopedii metoda powinna zwrócić pusty napis (zakładamy, że hasło może mieć co najwyżej jedną definicję).
- 6.29 Przerób klasę **encyklop** z zadania 6.28 w taki sposób, żeby umożliwiały przechowywanie wielu definicji dla jednego hasła. Metoda **znajdz** powinna zwracać jako wartość listę definicji dla podanego w argumencie hasła.
- 6.30 Do klas **encyklop** z zadań 6.28 i 6.29 dodaj publiczną metodę **zakres** wypisującą na standardowym wyjściu definicje haseł z zakresu podanego w argumentach metody (dla argumentów **haslo1** i **haslo2** metoda powinna wypisać definicje haseł następujących w kolejności leksykograficznej po **haslo1** a przed **haslo2**).
- 6.31 Napisz klasę **uczniowie** służącą do przechowywania adresów uczniów. Klasa **uczniowie** powinna udostępniać następujące publiczne metody:

- dodaj dodającą ucznia, którego dane zostały podane w argumentach,
- adres zwracająca jako wartość adres ucznia o imieniu i nazwisku podanych w argumentach,
- lista wypisującą na standardowym wyjściu przechowywane dane uczniów. Dane powinny być wypisywane w porządku alfabetycznym względem nazwisk i imion uczniów.

Implementując klasę `uczniowie` dopuść możliwość, że dwaj różni uczniowie mogą mieć takie samo imię i nazwisko.

- 6.32 (r) Napisz szablon funkcji, która dostaje jako argumenty iteratory (forward iterators) do początku oraz końca przedziału kontenera i wypisuje na standardowym wyjściu wszystkie elementy przedziału.

Pamiętaj, że w STL-u przedział wyznaczony przez iteratory `początek` i `koniec` zawiera `*początek`, a nie zawiera `*koniec`. Iterator `koniec` wskazuje tuż za koniec przedziału.

- 6.33 Napisz szablon funkcji, która dostaje jako argumenty iteratory (forward iterators) do początku oraz końca przedziału kontenera (zakładamy, że elementami przedziału są liczby) i wypełnia otrzymany w argumentie przedział zerami.

- 6.34 Napisz szablon funkcji, która dostaje jako argumenty iteratory (forward iterators) do początku oraz końca przedziału kontenera (zakładamy, że elementami przedziału są liczby) i zwraca jako wartość drugi co do wielkości element przedziału otrzymanego w argumentach.

- 6.35 Napisz szablon funkcji, która dostaje jako argumenty iteratory (bidirectional iterators) do początku oraz końca przedziału kontenera i odwraca kolejność elementów przedziału. Rozwiąż zadanie nie korzystając z biblioteki `algorithm`.

- 6.36 Napisz szablon funkcji, która otrzymuje jako argumenty iteratory (random access iterators) do początku oraz końca przedziału kontenera i zwraca jako wartość losowy element otrzymanego w argumentach przedziału.

- 6.37 Napisz szablon funkcji, która dostaje jako argumenty iteratory (random access iterators) do początku oraz końca przedziału kontenera i w losowy sposób zmienia kolejność elementów otrzymanego w argumentie przedziału.

- 6.38 (r) Napisz funkcję `vec_abs`, która otrzymuje jako argument referencję do wektora o elementach typu `int` i nadaje wszystkim elementom wektora ich wartość bezwzględne. W funkcji wykorzystaj szablon `for_each` z biblioteki `algorithm`.

- 6.39 (r,C++11) Napisz funkcję, która dostaje jako argument wektor o elementach typu `int` i zwraca `true` jeżeli

- a) któryś z elementów wektora jest podzielny przez liczbę z zakresu od 2 do 10,
 b) żaden z elementów wektora nie jest podzielny przez liczbę z zakresu od 2 do 10
 c) każdy z elementów jest podzielny przez liczbę z zakresu od 2 do 10
 Do rozwiązanie zadania wykorzystaj, któryś z szablonów `all_of`, `any_of`, `none_of` z biblioteki `algorithm`.
- 6.40 Napisz funkcję, która dostaje jako argument wektor o elementach typu `int` i zwraca:
 a) liczbę elementów wektora równych 0,
 b) liczbę elementów wektora z przedziału od 2 do 10
 c) liczbę elementów parzystych wektora.
 Do rozwiązanie zadania wykorzystaj któryś z szablonów `count`, `count_if` z biblioteki `algorithm`.
- 6.41 Napisz funkcję, która dostaje jako argumenty dwa wektory `w1`, `w2` liczb całkowitych `i`:
 a) wypełnia zerami pozycje, na których wektory `w1` i `w2` się różnią (jeżeli `w1[i] != w2[i]`, to funkcja ma nadać elementom `w1[i]` i `w2[i]` wartość 0),
 b) wypełnia zerami pozycje, na których wektory `w1` i `w2` mają elementy dająceną resztę z dzielenia przez 2.
 Do rozwiązanie zadania wykorzystaj szablon `mismatch` z biblioteki `algorithm`.
- 6.42 (r) Napisz funkcję, która otrzymuje jako argument napis typu `string` i zwraca wartość `true` wtedy i tylko wtedy, gdy podany w argumencie napis jest palindromem. W rozwiązaniu zadania wykorzystaj szablon `equal` z biblioteki `algorithm`.
- 6.43 Napisz funkcję, która otrzymuje jako argument wektor `wek` liczb całkowity i zwraca jako wartość:
 a) indeks pierwszego elementu równego 0 wektora `wek` lub `-1` jeżeli taki element nie istnieje,
 b) wartość pierwszego dodatniego elementu wektora `wek` lub `0` jeżeli taki element nie istnieje.
 W funkcji wykorzystaj któryś z szablonów `find`, `find_if`, `find_if_not` (ten ostatni tylko w C++11) z biblioteki `algorithm`.
- 6.44 Napisz funkcję, która otrzymuje jako argument wektor `wek` dodatnich liczb całkowitych `i`:
 a) zwraca jako wartość indeks pierwszego wystąpienia w wektorze `wek` liczby pierwnej nie większej niż 100,
 b) zwraca jako wartość indeks pierwszego wystąpienia w wektorze `wek` liczby podzielnej przez liczbę pierwszą nie większą niż 100.
 c) wypisuje na standardowym wyjściu wszystkie występujące w wek-

torze `wek` liczby podzielne przez jakąś liczbę pierwszą mniejszą niż 100.

Do rozwiązania zadania wykorzystaj szablon `find_first_of` z biblioteki `algorithm`.

- 6.45 (**C++11**) Napisz funkcję, która dostaje dwa argumenty stałą referencję `wek1` do wektora liczb całkowitych oraz referencję `wek2` do wektora liczb całkowitych i przekopiowuje do `wek2` wszystkie parzyste liczby zawarte w `wek1`. Do rozwiązania wykorzystaj szablon `copy_if` z biblioteki `algorithm`.
- 6.46 Napisz funkcję, która sortuje wektor liczb wymiernych otrzymanych w argumencie przy pomocy algorytmu quicksort. W funkcji wykorzystaj szablon `partition` z biblioteki `algorithm`.
- 6.47 (***) Napisz własny alokator, który posiada dużą statyczną automatyczną tablicę o elementach będących typu będącego parametrem szablonu i udostępnia jej fragmenty. Napisz alokator tak, aby mógł być używany jako parametr w kontenerach STL-a (np.: jako drugi parametr szablonu klasy `vector`).

ROZDZIAŁ 7

WYJĄTKI

7.1 (r) Napisz funkcję, która otrzymuje jako argument dodatnią liczbę `n`, alokuje `n`-elementową tablicę o elementach typu `double` i zwraca jako wartość wskaźnik do pierwszego elementu świeżo utworzonej tablicy.

W przypadku gdy alokacja tablicy się nie powiedzie, funkcja powinna wypisać na standardowym wyjściu komunikat:

Nieudana próba alokacji tablicy i zwrócić jako wartość `NULL`.

Zadanie rozwiąż na dwa sposoby, używając dwóch wersji operatora `new`: rzucającej i nierzucającej wyjątki.

7.2 (r) Napisz funkcję `zamien`, która otrzymuje jako argument trzy wektory `v1, v2, v3` liczb całkowitych oraz trzy nieujemne liczby całkowite `11, 12, 13` i zamienia ze sobą wartościami elementy `v1[11], v2[12]` oraz `v3[13]` w taki sposób, że stara wartość `v1[11]` ma być zapisana w `v2[12]`, stara wartość `v2[12]` w `v3[13]`, zaś stara wartość `v3[13]` w `v1[11]`. W przypadku gdy któryś z indeksów `11, 12, 13` leży poza zakresem indeksów wektora, do którego się odnosi, funkcja powinna rzucić standardowy wyjątek `std::out_of_range`. Przed rzuceniem wyjątku funkcja nie powinna zmieniać wartości żadnego z elementów wektorów `v1, v2, v3`.

7.3 Operator `[]` w klasie `tablica` z zadania 4.4 przeciąż w taki sposób, żeby przy podaniu indeksu spoza zakresu dozwolonych indeksów rzucił wyjątek `std::out_of_range`.

7.4 (r) Zdefiniuj hierarchię wyjątków występujących przy operacjach na zmiennych liczbowych przechowujących liczby wymierne. Uwzględnij wyjątki rzucane w przypadku: dzielenia przez zero, liczenia pierwiastka kwadratowego z liczby ujemnej, użycia w działaniu niezainicjowanej zmiennej, próbie przypisania wartości do stałej, działania o wyniku spoza zakresu itd.

7.5 (rozw) Napisz funkcję będącą obudową funkcji

- a) `pow`
- b) `sqrt`

z biblioteki `cmath`. Twoja funkcja powinna różnić się od oryginalnej funkcji tym, że zamiast zmieniać wartość makra `errno` w przypadku błędu, rzuca jeden z wyjątków z hierarchii zdefiniowanej w zadaniu 7.4.

7.6 (r,C) Napisz funkcje obudowujące działania arytmetyczne na zmiennych typu `double` w taki sposób, że w przypadku wystąpienia błędu funkcje te:

- a) ustawiają odpowiednią wartość błędu w makrze `errno`,
- b) rzucają odpowiedni wyjątek z hierarchii wyjątków zdefiniowanej w zadaniu 7.4.

7.7 (r,C) Napisz funkcję `rownanie`, która otrzymuje jako argumenty zmienne `a, b, c` typu `double` i wypisuje na standardowym wyjściu rozwiązania równania $a \cdot x^2 + b \cdot x + c = 0$ lub napis `nie ma rozwiązań`. Funk-

cję napisz tak, żeby wykrywała sytuacje, w których rozwiązanie może nie być poprawne. W funkcji wykorzystaj rozwiązania zadań 7.5 i 7.6. Porównaj rozwiązania bazujące na wyjątkach oraz na makrzu `errno`.

- 7.8 (r) Napisz wersję funkcji `rownanie` z zadania 7.7, która po złapaniu jakiegoś wyjątku wypisuje odpowiedni komunikat na standardowym wyjściu i rzuca z powrotem wyjątek.
- 7.9 Napisz klasę `nowy_double` służącą do przechowywania zmiennych typu `double`. Obiekty klasy `nowy_double` powinny pamiętać, czy zostały zainicjowane jakąś wartością, czy nie, i czy są stałe, czy nie. Klasa `nowy_double` powinna posiadać:
- bezargumentowy konstruktor inicjujący obiekt o niezdefiniowanej przechowywanej wartości,
 - jednoargumentowy konstruktor inicjujący obiekt o wartości podanej w argumencie konstruktora,
 - konstruktor kopiący,
 - przeciążone operatory arytmetyczne w taki sposób, aby w przypadku niedozwolonego działania (na przykład dzielenia przez 0 lub podania jako argumentu zmiennej o nieokreślonej wartości) rzucały odpowiedni wyjątek z hierarchii wyjątków zdefiniowanej w zadaniu 7.4,
 - bezargumentowe metody `stała`, `zmienna` powodujące ustalenie wartości przechowywanej w obiekcie jako odpowiednio stałej lub zmiennej.
 - operator przypisania, który rzuca odpowiedni wyjątek z hierarchii zdefiniowanej w zadaniu 7.4 w przypadku użycia po lewej stronie operatora obiektu przechowującego wartość oznaczoną jako stałą.
- 7.10 Napisz funkcję zwracającą jako wartość pierwiastek kwadratowy z liczby typu `nowy_double` otrzymanej w argumencie. Zwrócona wartość powinna być typu `nowy_double`. W przypadku gdy argument jest liczbą ujemną lub ma nieokreślona wartość funkcja, powinna rzucić odpowiedni wyjątek z hierarchii zdefiniowanej w zadaniu 7.4.
- 7.11 Napisz funkcję, która otrzymuje jako argumenty zmienne `a`, `b`, `c` typu `nowy_double` z zadania 7.9 i wypisuje na standardowym wyjściu rozwiązania równania $a \cdot x^2 + b \cdot x + c = 0$ lub napis `nie ma rozwiązań`. Zadanie rozwiąż na dwa sposoby z wykorzystaniem instrukcji `if` oraz bez instrukcji `if`, ale przy użyciu wyjątków.
- 7.12 Stwórz hierarchię wyjątków służącą do obsługi zdarzeń związanych z alokacją i obsługą tablicy. Uwzględnij takie zdarzenia jak: problem z alokacją tablicy, odnoszenie się do elementu tablicy za pomocą indeksu mniejszego niż 0, odnoszenie się do elementu tablicy za pomocą zbyt dużego indeksu, próba dereferencji iteratora nie zainicjowanego żadną wartością, próba dereferencji iteratora wskazującego poza zakres tablicy,

próba dereferencji nieaktualnego iteratora, próba inkrementacji/dekrementacji iteratora nie zainicjowanego żadną wartością itd.

- 7.13 Napisz klasę **tablica** służącą do przechowywania tablicy liczb całkowitych. Klasa **tablica** powinna udostępniać:
- konstruktor tworzący tablicę liczb całkowitych o rozmiarze podanych w argumencie, w przypadku problemów z alokacją tablicy konstruktor powinien rzucić odpowiedni wyjątek,
 - przeciążony operator `[]`, w przypadku podania niewłaściwego indeksu powinien rzucić odpowiedni wyjątek.
- W zadaniu wykorzystaj wyjątki zdefiniowane w zadaniu 7.12.
- 7.14 Zdefiniuj iterator dla klasy **tablica** z zadania 7.13. Iterator powinien mieć przeciążone operatory `++`, `--`, `*` (operator dereferencji). W przypadku niemożności wykonania odpowiedniej operacji (na przykład operacji arytmetycznej na niezainicjowanym iteratorze, albo dereferencji nieaktualnego lub niewskazującego na żaden element tablicy iteratora) operator powinien rzucić odpowiedni wyjątek zdefiniowany w zadaniu 7.12.
- 7.15 Napisz funkcję, która otrzymuje jako argumenty dwie referencje do iteratorów o typie zdefiniowanym w zadaniu 7.14 wskazujące na początek i koniec pewnego zakresu, i wypisuje na standardowym wyjściu wszystkie wartości z podanego zakresu. Funkcja powinna obsługiwać wszystkie mogące wystąpić wyjątki i informować o nich odpowiednim komunikatem wypisanym na standardowym wyjściu.
- 7.16 Napisz wersję funkcji z zadania 7.15, która w przypadku złapania jakiegoś wyjątku poza wypisaniem odpowiedniego komunikatu na standardowym wyjściu rzuca dalej ten sam wyjątek.
- 7.17 (r) Napisz funkcję **bezpieczna**, która wypisuje na standardowym wyjściu **jestem bezpieczna**. Funkcję **bezpieczna** zadeklaruj jako nierzucającą wyjątków.
- 7.18 (r) Napisz funkcję **podziel**, która dla dwóch argumentów całkowitoliczbowych zwraca jako wartość ich iloraz. W przypadku dzielenia przez zero funkcja powinna rzucić odpowiedni wyjątek. W deklaracji funkcji zdefiniuj wyjątki jakie może rzucić ta funkcja.
- 7.19 (r,!,*) Napisz program, który wczytuje ze standardowego wejścia 10 par liczb i wypisuje na standardowym wyjściu ilorazy wszystkich tych par, w których druga liczba jest różna od 0. Do dzielenia wykorzystaj funkcję **podziel** z zadania 7.18. Przechwytyj wyjątki rzucane przez funkcję **podziel**. Zdefiniuj swoją funkcję **unexpected** w taki sposób, żeby w wypadku rzucenia w funkcji **podziel** wyjątku spoza listy wypisała ona komunikat **nieznany blad przy dzieleniu** i rzuciła dalej zdefiniowany na taką okoliczność wyjątek. Uwzględnij ten wyjątek w liście dopuszczalnych wyjątków funkcji **podziel**.

- 7.20 (r,!,*) W programie z zadania 7.19 zdefiniuj swoją funkcję `terminate` wypisującą na standardowym wyjściu komunikat `niezlapany wyjątek` i wywołującą funkcje `exit`.
- 7.21 Zadania od 7.9 do 7.11 oraz od 7.13 do 7.16 rozwiąż w taki sposób, żeby wszystkie zaimplementowane funkcje i metody deklarowały listę wyjątków jakie mogą rzucić.
- 7.22 (r,C,!,*) Zadania 7.18 i 7.19 napisz symulując wyjątki przy pomocy operacji `setjmp` i `longjmp`. W nowym rozwiązaniu zadania 7.19 nie definiuj swojej funkcji `unexpected`.

ROZDZIAŁ 8

ROZWIAZANIA

8.1.	Rozwiązań zadań z rozdziału 1	64
8.2.	Rozwiązań zadań z rozdziału 2	81
8.3.	Rozwiązań zadań z rozdziału 3	85
8.4.	Rozwiązań zadań z rozdziału 4	93
8.5.	Rozwiązań zadań z rozdziału 5	102
8.6.	Rozwiązań zadań z rozdziału 6	115
8.7.	Rozwiązań zadań z rozdziału 7	121

8.1. Rozwiązania zadań z rozdziału 1

Zadanie 1.1

Listing 8.1. rozwiązanie zadania 1.1

```

1 class poczta {
2     public:
3         std::string nadawca, odbiorca, temat, tresc;
4     };

```

Zadanie 1.2

Listing 8.2. rozwiązanie zadania 1.2

```

void wypisz(poczta p){
    std::cout << "nadawca:" << p.nadawca << std::endl;
    std::cout << "odbiorca:" << p.odbiorca << std::endl;
    std::cout << "temat:" << p.temat << std::endl;
    std::cout << "tresc:" << p.tresc << std::endl;
}

```

Zadanie 1.3

Listing 8.3. rozwiązanie zadania 1.3

```

void wczytaj(poczta &p){
    std::cout << "Podaj_nadawce:" ;
    std::cin >> p.nadawca;
    std::cout << "Podaj_odbiorce:" ;
    std::cin >> p.odbiorca;
    std::cin.ignore();
    std::cout << "Podaj_temat:" ;
    getline(std::cin, p.temat);
    std::cout << "Podaj_tresc:" ;
    getline(std::cin, p.tresc);
}

```

Zadanie 1.4

Listing 8.4. rozwiązanie zadania 1.4

```

1 class poczta {
2     public:
3         std::string nadawca, odbiorca, temat, tresc;
4
5     void wczytaj();
6     void wypisz();
7 };

```

```
9 void poczta::wypisz() {
    std::cout<<"nadawca:"<<nadawca<<std::endl;
11   std::cout<<"odbiorca:"<<odbiorca<<std::endl;
    std::cout<<"temat:"<<temat<<std::endl;
13   std::cout<<"tresc:"<<tresc<<std::endl;
}
15

17 void poczta::wczytaj() {
    std::cout<<"Podaj nadawce:";
19   std::cin>>nadawca;
    std::cout<<"Podaj odbiorce:";
21   std::cin>>odbiorca;
    std::cin.ignore();
23   std::cout<<"Podaj temat:";
    getline(std::cin, temat);
25   std::cout<<"Podaj tresc:";
    getline(std::cin, tresc);
27 }
```

Zadanie 1.5 Struktury i klasy to w języku C++ pojęcia niemal tożsame. Różnią się szczegółami takimi jak to, że pola struktur domyślnie są publiczne, zaś pola klas domyślnie są prywatne. Poniżej definicja struktury `poczta2` oraz operujących na niej funkcji:

Listing 8.5. rozwiązanie zadania 1.5

```
1 struct poczta2 {
    std::string nadawca, odbiorca, temat, tresc;
3 }

5 void wypisz(poczta2 p){
    std::cout<<"nadawca:"<<p.nadawca<<std::endl;
7   std::cout<<"odbiorca:"<<p.odbiorca<<std::endl;
    std::cout<<"temat:"<<p.temat<<std::endl;
9   std::cout<<"tresc:"<<p.tresc<<std::endl;
}
11

13 void wczytaj(poczta2 &p){
    std::cout<<"Podaj nadawce:";
15   std::cin>>p.nadawca;
    std::cout<<"Podaj odbiorce:";
17   std::cin>>p.odbiorca;
    std::cin.ignore();
19   std::cout<<"Podaj temat:";
    getline(std::cin, p.temat);
21   std::cout<<"Podaj tresc:";
    getline(std::cin, p.tresc);
}
```

Poniżej wersja struktury `poczta2` z dopisanym metodami do wczytywania i wypisywania zawartości:

Listing 8.6. rozwiązań zadania 1.5

```

struct poczta2 {
    std :: string nadawca , odbiorca , temat , tresp ;

    void wczytaj();
    void wypisz();
};

void poczta2 :: wypisz(){
    std :: cout<<"nadawca : " << nadawca << std :: endl ;
    std :: cout<<"odbiorca : " << odbiorca << std :: endl ;
    std :: cout<<"temat : " << temat << std :: endl ;
    std :: cout<<"tresp : " << tresp << std :: endl ;
}

void poczta2 :: wczytaj(){
    std :: cout<<"Podaj nadawce : ";
    std :: cin>>nadawca ;
    std :: cout<<"Podaj odbiorce : ";
    std :: cin>>odbiorca ;
    std :: cout<<"Podaj temat : ";
    std :: cin . ignore ();
    getline ( std :: cin , temat );
    std :: cout<<"Podaj tresp : ";
    getline ( std :: cin , tresp );
}

```

Zadanie 1.15

Listing 8.7. rozwiązań zadania 1.15

```

class liczba{
2 private:
    int wart_licz ;
4 public:
    void wczytaj();
    void wypisz();
    void nadaj_w(int);
    int wartosc();
    unsigned int abs();
10 };

12 void liczba :: wczytaj(){
    std :: cout<<"Podaj wartosc liczby : ";

```

```
14     std :: cin>>wart_licz;
15 }
16 void liczba :: wypisz(){
17     std :: cout<<"Przechowywana liczba to: "<<wart_licz<<std :: endl;
18 }
19 void liczba :: nadaj_w(int n){
20     wart_licz=n;
21 }
22 int liczba :: wartosc(){
23     return wart_licz;
24 }
25 unsigned int liczba :: abs(){
26     if (wart_licz>=0)
27         return wart_licz;
28     else
29         return -wart_licz;
30 }
31 }
```

Zadania 1.17–1.20

Listing 8.8. rozwiązanie zadań 1.17–1.20

```
class punkt{
2     int x,y;
3     friend punkt rzutuj(punkt3);
4 public:
5     void rzutuj(punkt3);
6     void wczytaj();
7     void wypisz();
8 };

10 class punkt3{
11     int x,y,z;
12     friend punkt rzutuj(punkt3);
13     friend void punkt :: rzutuj(punkt3);
14 public:
15     void wczytaj();
16     void wypisz();
17 };
18 void punkt :: wczytaj(){
19     std :: cout<<"Podaj wspolrzedne punktu: ";
20     std :: cin>>x>>y;
21 }
22

24 void punkt :: wypisz(){
25     std :: cout<<"Wspolrzedne punktu to: ";
```

```

26         std :: cout<<x<<" " <<y<<std :: endl ;
    }
28

30 void punkt3 :: wczytaj () {
    std :: cout<<"Podaj wspolrzedne punktu : ";
32     std :: cin>>x>>y>>z ;
    }
34

35 void punkt3 :: wypisz () {
36     std :: cout<<"Wspolrzedne punktu to : ";
37     std :: cout<<x<<" "<<y<<" "<<z<<std :: endl ;
38 }
39     punkt rzutuj (punkt3 p3){
40         punkt p2;
        p2 . x=p3 . x;
42         p2 . y=p3 . y;
        return p2;
44 }
45     void punkt :: rzutuj (punkt3 p3){
46         x=p3 . x;
        y=p3 . y;
48 }

```

Zadanie 1.26Listing 8.9. rozwiązanie zadania 1.26

```

1 class ukryta _ liczba{
2 private:
3     int liczba ;
4 public:
5     void zeruj ();
6 };
7     void ukryta _ liczba :: zeruj (){
8         liczba=0;
9     }

```

Zadanie 1.27Listing 8.10. rozwiązanie zadania 1.27

```

1 class ukryta _ liczba{
2     private:
3         int liczba ;
4         friend void inkrementuj (ukryta _ liczba&);
5 public:
6     void zeruj ();
7 };

```

```
9 void ukryta_liczba::zeruj() {
    liczba=0;
11 }

13 void inkrementuj(ukryta_liczba& u) {
    u.liczba++;
15 }
```

Zadanie 1.28

Listing 8.11. rozwiązanie zadania 1.28

```
1 class ukryta_liczba{
2 private:
3     int liczba;
4 public:
5     void zeruj();
6     void inkrementuj(ukryta_liczba& );
7 };
8
9 void ukryta_liczba::zeruj() {
10    liczba=0;
11 }
12
13 void ukryta_liczba::inkrementuj(ukryta_liczba& u){
14     u.liczba++;
15 }
```

Zadania 1.33-1.35

Listing 8.12. rozwiązanie zadań 1.33–1.35

```
1
2 class wskaznik{
3 private:
4     int *wsk;
5 public:
6     wskaznik();
7     void utworz(int);
8     int* zwroc();
9     void zwolnij();
10    void kopij(wskaznik& );
11    ~wskaznik();
12    friend void przepisz(int*,wskaznik & );
13 };

15
16    wskaznik::wskaznik() {
17        wsk=NULL;
18    }
```

```

19     void wskaznik :: utworz( int n) {
21         wsk = new int [n];
22     }
23
24     int* wskaznik :: zwroc () {
25         return wsk;
26     }
27
28     void wskaznik :: zwolnij () {
29         delete [] wsk;
30         wsk = NULL;
31     }

32 void wskaznik :: kopiuj (wskaznik & ref) {
33     ref.wsk=wsk;
34 }

36 wskaznik :: ~ wskaznik () {
37     if (wsk!=NULL)
38         delete [] wsk;
39     }
40
41     void przepisz (int * t, wskaznik& ref){
42         ref.wsk = t;
43     }

```

Zadanie 1.36

Listing 8.13. rozwiązań zadania 1.36

```

1 class identyfikator {
2 private:
3     unsigned int liczba;
4 public:
5     unsigned int id ();
6 };
7
8 unsigned int identyfikator :: id () {
9     return liczba++;
10 }

```

Zadanie 1.37

Listing 8.14. rozwiązań zadania 1.37

```

1 class identyfikator2 {
2 private:
3     unsigned int liczba;
4

```

```
    public:  
6       identyfikator2();  
      unsigned int id();  
8   };  
  
10  identyfikator2 :: identyfikator2(){  
     liczba=0;  
12 }  
  
14  unsigned int identyfikator2 :: id(){  
     return liczba++;  
16 }
```

Zadanie 1.38

Listing 8.15. rozwiązanie zadania 1.38

```
class semafor_bin{  
2 private:  
    bool wolny;  
4 public:  
    semafor_bin(){wolny=true;}  
6    void rezerwuj(){wolny=false;}  
    void zwolnij(){wolny=true;}  
8    bool stan(){return wolny}  
};
```

Zadanie 1.43

Listing 8.16. rozwiązanie zadania 1.43

```
1 class osoba{  
  private:  
3   std::string imie, nazwisko;  
  public:  
5   osoba(std::string, std::string);  
    void wczytaj();  
7    void wypisz();  
  };  
9  
osoba :: osoba(std::string im, std::string nazw){  
11   imie=im;  
    nazwisko=nazw;  
13 }  
  void osoba :: wczytaj(){  
15   std::cout<<"Podaj imię : ";  
    std::cin>>imie;  
17   std::cout<<"Podaj nazwisko : ";  
    std::cin>>nazwisko;  
19 }
```

```

21 void osoba::wypisz(){
22     std::cout<<"imie"<<imie<<std::endl;
23     std::cout<<"nazwisko"<<nazwisko<<std::endl;
24 }
```

Zadanie 1.44

Listing 8.17. rozwiązań zadania 1.44

```

std::vector<osoba> ZrobWektor( int n){
2    return std::vector<osoba>(n, osoba( "Jan", "Kowalski"));
}
```

Zwróćmy uwagę, że nie moglibyśmy napisać podobnej funkcji zwracającej jako wartość tablicy obiektów klasy `osoba`. Jest to spowodowane tym, że klasa `osoba` nie posiada bezargumentowego konstruktora. Pewne rozwiązanie tego problemu można znaleźć w rozwiązańiu kolejnego zadania.

Zadanie 1.45

Listing 8.18. rozwiązań zadania 1.45

```

1 class tab_osoba{
2     private:
3         osoba ** tab;
4         int rozmiar;
5     public:
6         tab_osoba( int , std::string , std::string );
7         osoba& at( int );
8         ~tab_osoba();
9     };
10 tab_osoba::tab_osoba( int n, std::string imie,
11                         std::string nazwisko){
12     tab = new osoba*[n];
13     rozmiar=n;
14     for( int i=0;i<n;i++)
15         tab[ i ] = new osoba( imie , nazwisko );
16 }
17
18 osoba& tab_osoba::at( int i){
19     return *tab[ i ];
20 }
21 tab_osoba::~tab_osoba(){
22     for( int i=0;i<rozmiar ;i++)
23         delete tab[ i ];
24     delete [] tab;
25 }
```

Zadanie 1.46

Listing 8.19. rozwiązanie zadania 1.46

```
1 class kolejka {
2     private:
3         struct element{
4             int liczba;
5             element * nastepny;
6         };
7         element *pierw, *ostatni;
8     public:
9         kolejka();
10        kolejka(const kolejka&);
11        ~kolejka();
12        int pierwszy();
13        void usun_pierwszy();
14        void dodaj_na_koniec(int);
15        bool pusta();
16    };
17
18 kolejka::kolejka(){
19     pierw=ostatni=NULL;
20 }
21
22 kolejka::kolejka(const kolejka & kol){
23     if (kol.pierw==kol.ostatni){
24         pierw=ostatni=NULL;
25     }
26     else {
27         element *pom;
28         pom=kol.pierw;
29         pierw= new element;
30         pierw->liczba=pom->liczba;
31         ostatni=pierw;
32         while(pom!=kol.ostatni){
33             pom=pom->nastepny;
34             ostatni->nastepny=new element;
35             ostatni=ostatni->nastepny;
36             ostatni->liczba = pom->liczba;
37         }
38     }
39 }
40
41 kolejka::~kolejka(){
42     if (pierw!=NULL){
43         element *pom;
44         while(pierw!=ostatni){
45             pom=pierw;
46             pierw=pierw->nastepny;
47             delete pom;
48         }
49 }
```

```

50         }
51     }
52     int kolejka::pierwszy(){
53         return pierw->liczba;
54     }
55
56     void kolejka::usun_pierwszy(){
57         element* pom=pierw;
58         if (pierw==ostatni)
59             pierw=ostatni=NULL;
60         else
61             pierw=pierw->nastepny;
62         delete pom;
63     }
64 }

66     void kolejka::dodaj_na_koniec(int liczba){
67         if (ostatni!=NULL){
68             ostatni->nastepny=new element;
69             ostatni=ostatni->nastepny;
70         }
71         else{
72             ostatni=pierw=new element;
73         }
74         ostatni->liczba=liczba;
75     }
76 }

78     bool kolejka::pusta(){
79         return (pierw==NULL);
80     }

```

Zadanie 1.57

Listing 8.20. rozwiązań zadania 1.57

```

1 class figura{
2 public:
3     double pole, obwod;
4 };

6 class trojkat: public figura{
7 public:
8     double a,h;
9 };
10
11 class prostokat: public figura{
12 public:
13     double a,b;
14 };

```

Zadanie 1.58

Listing 8.21. rozwiàzanie zadania 1.58

```
1 struct figura{  
2     public:  
3         double pole, obwod;  
4     };  
  
6 struct trojkat: public figura{  
7     public:  
8         double a,h;  
9     };  
10  
11 struct prostokat: public figura{  
12     public:  
13         double a,b;  
14     };
```

W przeciwnieństwie do klas, które domyślnie dziedziczą w sposób prywatny, struktury dziedziczą domyślnie w sposób publiczny. Klasy mogą dziedziczyć po strukturach, a struktury po klasach. O tym, jaki rodzaj dziedziczenia jest domyślny, decyduje to, co jest typem pochodnym, struktura czy klasa.

Zadanie 1.62

Listing 8.22. rozwiązanie zadania 1.62

```

    cena = c;
24    nazwa = n;
    opis = o;
26    godzina_odejazdu = go;
    godzina_przyjazdu = gp;
28 }
```

Zadanie 1.63

Listing 8.23. rozwiązań zadania 1.63

```

class lista {
2 protected:
    struct element{
4        int liczba;
        element * nastepny, *poprzedni;
6    };
    element *pierwszy, *ostatni;
8 public:
    lista();
10   lista(const lista&);
    ~lista();
12   void dodaj_przod(int);
    void dodaj_tyl(int);
14   void usun_przod();
    void usun_tyl();
16   int pierwszy_el();
    int ostatni_el();
18   bool pusta_lista();
    };
20
    lista::lista(){
22     pierwszy=ostatni=NULL;
    }
24
    lista::lista(const lista & list){
26     if (list.pierwszy==list.ostatni){
         pierwszy=ostatni=NULL;
28     }
     else {
30         element *pom;
         pom=list.pierwszy;
32         pierwszy= new element;
         pierwszy->liczba=pom->liczba;
34         ostatni=pierwszy;
         while(pom!=list.ostatni){
36             pom=pom->nastepny;
             ostatni->nastepny=new element;
38             ostatni->nastepny->poprzedni=ostatni;
             ostatni=ostatni->nastepny;
             ostatni->liczba = pom->liczba;
40         }
     }
}
```

```
        }
42    }
44
45    lista::~lista(){
46        if (pierwszy!=NULL){
47            element *pom;
48            while(pierwszy!=ostatni){
49                pom=pierwszy;
50                pierwszy=pierwszy->nastepny;
51                delete pom;
52            }
53            delete pierwszy;
54        }
55    }
56
57    void lista::dodaj_przod(int liczba){
58        if (pierwszy!=NULL){
59            pierwszy->poprzedni=new element;
60            pierwszy->poprzedni->nastepny=pierwszy;
61            pierwszy=pierwszy->poprzedni;
62        }
63        else{
64            ostatni=pierwszy=new element;
65        }
66        pierwszy->liczba=liczba;
67    }
68
69    void lista::dodaj_tyl(int liczba){
70        if (ostatni!=NULL){
71            ostatni->nastepny=new element;
72            ostatni->nastepny->poprzedni=ostatni;
73            ostatni=ostatni->nastepny;
74        }
75        else{
76            ostatni=pierwszy=new element;
77        }
78        ostatni->liczba=liczba;
79    }
80
81    void lista::usun_przod(){
82        element* pom=pierwszy;
83        if (pierwszy==ostatni)
84            pierwszy=ostatni=NULL;
85        else
86            pierwszy=pierwszy->nastepny;
87        delete pom;
88    }
89
90    void lista::usun_tyl(){
91        element* pom=ostatni;
```

```

92     if (pierwszy==ostatni)
93         pierwszy=ostatni=NULL;
94     else
95         pierwszy=pierwszy->nastepny ;
96     delete pom;
97 }
98
99 int lista :: pierwszy_el(){
100    return pierwszy->liczba ;
101 }
102
103 int lista :: ostatni_el(){
104    return ostatni->liczba ;
105 }
106
107 bool lista :: pusta_lista(){
108    return (pierwszy==NULL);
109 }
```

W klasie `lista` stworzyliśmy sekcję chronioną do której dostępu będzie potrzebowało rozwiążanie zadania 1.68.

Zadanie 1.64

Listing 8.24. rozwiązanie zadania 1.64

```

1 class kolejka {
2     private:
3     lista list ;
4     public:
5     int pierwszy () {return list .pierwszy_el();}
6     void usun_pierwszy (){list.usun_przod();}
7     void dodaj_na_koniec(int n){list.dodaj_tyl(n);}
8     bool pusta (){return list .pusta_lista();}
9 }
```

Zadanie 1.65

Listing 8.25. rozwiązanie zadania 1.65

```

1 class kolejka: private lista {
2     public:
3     int pierwszy () {return pierwszy_el();}
4     void usun_pierwszy (){usun_przod();}
5     void dodaj_na_koniec(int n){dodaj_tyl(n);}
6     bool pusta (){return pusta_lista();}
7 }
```

Jak widać na przykładzie rozwiązań zadań 1.64 i 1.65, dziedziczenie prywatne jest podobne w użyciu do posiadania prywatnego pola danego typu. W zasadzie jedynym istotnym powodem dla którego używa się dziedziczenia prywatnego jest potrzeba dostępu do chronionej sekcji klasy bazowej.

Zadanie 1.68

Listing 8.26. rozwiązanie zadania 1.68

```
1 class lepsza_lista:public lista{
2     public:
3         class iterator {
4             private:
5                 lepsza_lista& list;
6                 lista::element *wsk;
7             public:
8                 iterator (lepsza_lista& );
9                 int& element();
10                void nastepny();
11                void poprzedni();
12                bool poczatek();
13                bool koniec();
14
15            };
16        };
17
18        lepsza_lista::iterator::iterator (lepsza_lista & lista):list (lista){
19            wsk=list.pierwszy;
20        }
21
22        int& lepsza_lista::iterator::element(){
23            return wsk->liczba;
24        }
25
26        void lepsza_lista::iterator::nastepny(){
27            wsk=wsk->nastepny;
28        }
29
30        void lepsza_lista::iterator::poprzedni(){
31            wsk=wsk->poprzedni;
32        }
33
34        bool lepsza_lista::iterator::poczatek(){
35            return wsk==list.pierwszy;
36        }
37
38        bool lepsza_lista::iterator::koniec(){
39            return wsk==list.ostatni;
40        }
```

Zadanie 1.69

Listing 8.27. rozwiązań zadania 1.69

```

1 void zeruj(lepssa_lista& list){
2     if (!list.pusta_lista()){
3         lepssa_lista::iterator it(list);
4         while (!it.koniec()){
5             if (it.element() == 0);
6                 it.nastepny();
7         }
8         if (it.element() == 0);
9     }
10 }
```

Zadanie 1.70

Listing 8.28. rozwiązań zadania 1.70

```

1 class stala1{
2 public:
3     const int i;
4     stala1():i(5){}
5 };
```

Miejscem gdzie podajemy wartości stałych pól klasy jest lista inicjalizacyjna konstruktora. Wewnątrz konstruktora jest już za późno na nadawanie wartości stałym polom.

Zadanie 1.71

Listing 8.29. rozwiązań zadania 1.71

```

1 class stala2{
2 public:
3     const double d;
4     stala2(double dd):d(dd){}
5 };
```

Zadanie 1.72

Listing 8.30. rozwiązań zadania 1.72

```

1 class stale:public stala2{
2 public:
3     stala2 liczba;
4     stale(double arg1, double arg2):stala2(arg1),liczba(arg2){}
5 };
```

Lista inicjalizacyjna to jedyne miejsce, gdzie możemy podać argumenty konstruktora klasy bazowej. Jeżeli klasa bazowa lub któryś z typów pól nie posiada bezargumentowego konstruktora, to wywołanie odpowiedniego konstruktora za argumentami musimy umieścić na liście inicjalizacyjnej naszego konstruktora.

Zadanie 1.80–1.81

Listing 8.31. rozwiązanie zadań 1.80–1.81

```
1 class liczba{  
2 };  
3  
4 class wymierne: public liczba{  
5     double d;  
6 };  
7  
8 class calkowite:public liczba{  
9     int i;  
10 };  
11  
12 liczba* kopiuje(liczba* tab, int n){  
13     liczba * pom=new liczba[n];  
14     for(int i=0;i<n;i++)  
15         pom[i]=tab[i];  
16     return pom;  
17 }
```

8.2. Rozwiązania zadań z rozdziału 2

Zadanie 2.1

Listing 8.32. rozwiązanie zadania 2.1

```
1 class bazowa{  
2     public:  
3         void typ_wskaznika(){  
4             std :: cout<<"bazowa"<<std :: endl;  
5         }  
6  
7         virtual void typ_obiektu(){  
8             std :: cout<<"bazowa"<<std :: endl;  
9         }  
10     };  
11  
12 class pochodna1: public bazowa{  
13     public:
```

```

15     void typ_wskaznika() {
16         std :: cout<<"pochodna1"<<std :: endl;
17     }
18     void typ_obiektu() {
19         std :: cout<<"pochodna1"<<std :: endl;
20     }
21 };
22 class pochodna2: public bazowa{
23 public:
24     void typ_wskaznika() {
25         std :: cout<<"pochodna2"<<std :: endl;
26     }
27     void typ_obiektu() {
28         std :: cout<<"pochodna2"<<std :: endl;
29     }
30 };

```

Zadanie 2.2

Listing 8.33. rozwiązanie zadania 2.2

```

1 class liczba{
2 public:
3     double re;
4     virtual double modul();
5     bool wieksza(liczba&);
6 };
7
7 double liczba :: modul(){
8     if (re>=0)
9         return re;
10    else
11        return -re;
12 }
13
14 bool liczba :: wieksza(liczba &ref){
15     if (this->modul()<ref.modul())
16         return true;
17     else
18         return false;
19 }
20

```

W zadaniu dla większej przejrzystości kodu użyliśmy wskaźnika `this`. Użycie samej nazwy metody miałoby taki sam skutek.

Zadanie 2.3

Listing 8.34. rozwiązanie zadania 2.3

```
1 class zespolone: public liczba {
2     public:
3         double im;
4         double modul();
5     };
6
7     double zespolone :: modul() {
8         return sqrt(re*re+im*im);
9     }

```

Nie ma potrzeby przeciążania metody `wieksza`, gdyż wykorzystuje ona wirtualną metodę `modul`. Pomimo tego, że metoda `wieksza` została zdefiniowana w klasie `liczba`, to używa wersji metody `modul` odpowiedniej dla porównywanych liczb.

Zadanie 2.4

Listing 8.35. rozwiązanie zadania 2.4

```
1 class funkcja {
2     public:
3         double x;
4         virtual double wartosc()=0;
5     };

```

Zadanie 2.5

Listing 8.36. rozwiązanie zadania 2.5

```
1 class funkcja_liniowa:public funkcja {
2     public:
3         double a,b;
4         double wartosc(){return a*x+b;}
5     };

```

Zadanie 2.6

Listing 8.37. rozwiązanie zadania 2.6

```
1 double bisekcja(funkcja * f, double p, double k, double d){
2     double wp, wk, ws;
3     f->x=p;
4     wp=f->wartosc();
5     f->x=k;
6     wk=f->wartosc();

```

```

7     if (wp*wk<=0)
9       while(k-p>d) {
11         f->x=(k+p)/2;
13         ws=f->wartosc();
15         if (wp*ws<=0)
17           k=f->x;
18       }
19     return p;
20 }
```

Zadanie 2.22

Listing 8.38. rozwiązanie zadania 2.22

```

1 class kolejka{
2 public:
3   virtual int pierwszy()=0;
4   virtual void usun_pierwszy()=0;
5   virtual void wstaw_na_koniec(int)=0;
6   virtual bool pusta()=0;
7   virtual ~kolejka(){}
8 };
```

Jeżeli przewidujemy, że po danej klasie mogą dziedziczyć inne klasy, i że będziemy chcieli przy usuwaniu z pamięci obiektów klas pochodnych podać jako argument operatora `delete` wskaźnik do klasy bazowej, to w klasie bazowej powinniśmy zdefiniować wirtualny (ale nie czysto wirtualny) destruktor. Nawet gdyby ten destruktor miał nic nie robić.

Zadanie 2.23

Listing 8.39. rozwiązanie zadania 2.23

```

1 class kolejka_listowa: public kolejka {
2 private:
3   struct element{
4     int wartosc;
5     element * nastepny, * poprzedni;
6   };
7   element * pierw, *ost;
8 public:
9   kolejka_listowa(){pierw=ost=NULL;}
10  ~kolejka_listowa();
11  int pierwszy(){return pierw->wartosc;}
12  void usun_pierwszy();
13  void dodaj_na_koniec(int);
14  bool pusta(){return (pierw==NULL);}
```

```
16 };  
17 kolejka_listowa::~kolejka_listowa(){  
18     if (pierw!=NULL){  
19         element * pom=pierw;  
20         while(pierw!=ost){  
21             pierw=pierw->nastepny;  
22             delete pom;  
23             pom=pierw;  
24         }  
25         delete pom;  
26     }  
27     void kolejka_listowa::usun_pierwszy(){  
28         if (pierw!=ost){  
29             element * pom = pierw;  
30             pierw=pierw->nastepny;  
31             delete pom;  
32         }  
33         else if (pierw==NULL){  
34             delete pierw;  
35             pierw=ost=NULL;  
36         }  
37     }  
38  
39     void kolejka_listowa::dodaj_na_koniec(int w){  
40         if (pierw==NULL){  
41             pierw=ost=new element;  
42             pierw->wartosc = w;  
43         }  
44         else {  
45             ost->nastepny = new element;  
46             ost= ost->nastepny;  
47             ost->wartosc = w;  
48         }  
49     }  
50 }
```

8.3. Rozwiązania zadań z rozdziału 3

Zadanie 3.1

Listing 8.40. rozwiązanie zadania 3.1

```
1 class stale{  
2     static const double pi=3.1415;  
3     static const double e=2.7;  
4 };
```

Zadanie 3.2

Listing 8.41. rozwizanie zadania 3.2

```
1 class liczba{  
2     public:  
3         static int licz;  
4     };  
5     int liczba :: licz=0;
```

Zadanie 3.6

Listing 8.42. rozwizanie zadania 3.6

Zadanie 3.7

Listing 8.43. rozwiązanie zadania 3.7

```

#include <iostream>
2
...
4
int main() {
6    zespolone z(0,1);
7    for(int i=0;i<100;i++)
8        if (i==0)
9            std :: cout<<"re = "<<z . re<<"im = "<<z . im
10           <<std :: endl;
11        else {
12            z= zesp :: podziel(zesp :: dodaj(
13                zesp :: pomnoz(zespolone(2,0),z),
14                zesp :: pomnoz(zespolone(10,0),zesp :: i)),z);
15            std :: cout<<"re = "<<z . re<<"im = "<<z . im
16           <<std :: endl;
17        }
18    }

```

W powyższym rozwiążaniu w miejscu trzech kropek należy wstawić definicje klas **zespolone** i **zesp**.

Zadania 3.8–3.9

Listing 8.44. rozwiązanie zadań 3.8–3.9

```

#include <iostream>
2
class zespolone{
4 public:
    double re ,im;
6    zespolone() {}
7    zespolone(double r, double i) : re(r),im(i){}
8};

10 namespace zesp{
11
12 const zespolone i(0,1);

14 zespolone dodaj(zespolone a, zespolone b){
15     return zespolone(a.re+b.re,a.im+b.im);
16 }

18 zespolone odejmij(zespolone a, zespolone b){
19     return zespolone(a.re-b.re,a.im-b.im);
20 }

```

```

22 zespolone pomnoz(zespolone a, zespolone b){
    return zespolone(a.re*b.re-a.im*b.im, a.re*b.im+a.im*b.re);
24 }

26 zespolone podziel(zespolone a, zespolone b){
    return zespolone((a.re*b.re+a.im*b.im)/(b.re*b.re+b.im*b.im),
28                               (a.im*b.re-a.re*b.im)/(b.re*b.re+b.im*b.im));
29 }
30 }

32 int main(){
    zespolone z(0,1);
34     for(int i=0;i<100;i++)
        if (i==0)
36         std::cout<<"re = "<<<z.re<<" im = "<<<z.im<<std::endl;
        else {
38         z=zesp::podziel(
            zesp::dodaj(zesp::pomnoz(zespolone(2,0),z),
40             zesp::pomnoz(zespolone(10,0),zesp::i)),z);
            std::cout<<"re = "<<<z.re<<" im = "<<<z.im<<std::endl;
42     }
44 }

```

Zadanie 3.16

Listing 8.45. rozwiązań zadania 3.16

```

class zesp{
2 private:
    zesp() {}
4 public:
    static const zespolone i;
6
    static zespolone dodaj(zespolone a, zespolone b);
8    static zespolone odejmij(zespolone a, zespolone b);
    static zespolone pomnoz(zespolone a, zespolone b);
10   static zespolone podziel(zespolone a, zespolone b);
    };
12
    const zespolone zesp::i(0,1);
14
    zespolone zesp::dodaj(zespolone a, zespolone b){
16        return zespolone(a.re+b.re,a.im+b.im);
    }
18

20 zespolone zesp::odejmij(zespolone a, zespolone b){
    return zespolone(a.re-b.re,a.im-b.im);
22 }

```

```
24 zespolone zesp::pomnoz(zespolone a, zespolone b){  
    return zespolone(a.re*b.re-a.im*b.im, a.re*b.im+a.im*b.re);  
26 }  
  
28 zespolone zesp::podziel(zespolone a, zespolone b){  
    return zespolone((a.re*b.re+a.im*b.im)/(b.re*b.re+b.im*b.im),  
30             (a.im*b.re-a.re*b.im)/(b.re*b.re+b.im*b.im));  
}
```

Zadanie 3.17

Listing 8.46. rozwiązanie zadania 3.17

```
1 class finalna{  
    private:  
3     finalna(){}  
    public:  
5     static finalna * utworz(){ return new finalna; }  
};
```

Po klasie **finalna** nie można dziedziczyć, gdyż z poziomu ewentualnej klasy pochodnej nie byłoby możliwe uruchomienie konstruktora klasy **finalna** (jak pamiętamy, przy tworzeniu obiektu, przed wywołaniem konstruktora klasy pochodnej, wywoływanie są konstruktorami klas bazowych).

Zauważmy, że ponieważ konstruktor klasy **finalna** jest prywatny, to nie możemy zadeklarować w programie zmiennych tego typu. Obiekty typu **finalna** możemy tworzyć w następujący sposób:

```
finalna *f = finalna::utworz();
```

Standard C++11 oznaczenie słowem **final** klas, po których nie można dziedziczyć oraz metod, których nie można przedefiniowywać w klasach potomnych. W momencie pisania skryptu kompilator g++ nie posiadał jeszcze zaimplementowanej tej możliwości.

Zadanie 3.18

Listing 8.47. rozwiązanie zadania 3.18

```
1 class dynamiczna{  
    protected:  
3     dynamiczna(int n){  
        tab = new int[n];  
5     }  
    public:  
7     int * tab;
```

```

9      static dynamiczna * utworz(int n)
10     {return new dynamiczna(n);}
11   };

```

Zwróćmy uwagę, że dzięki temu, iż klasa `dynamiczna` ma chroniony, a nie prywatny, konstruktor, to można tworzyć jej klasy pochodne.

Zadanie 3.19

Listing 8.48. rozwiązań zadania 3.19

```

1 class tab_info{
2 public:
3     unsigned int w_rozmiar;
4     bool czy_rozmiar;
5     int w_wartosc;
6     bool czy_wartosc;
7     int w_zmiany;
8     bool czy_zmiany;
9     tab_info();
10    tab_info& rozmiar(unsigned int);
11    tab_info& wartosc(int);
12    tab_info& zmiany(int);
13 };
14
15 tab_info::tab_info(){
16     czy_rozmiar = false;
17     czy_wartosc = false;
18     czy_zmiany = false;
19 }
20
21 tab_info& tab_info::rozmiar(unsigned int n){
22     w_rozmiar = n;
23     czy_rozmiar = true;
24     return *this;
25 }
26
27 tab_info& tab_info::wartosc(int n){
28     w_wartosc = n;
29     czy_wartosc = true;
30     return *this;
31 }
32
33 tab_info& tab_info::zmiany(int n){
34     w_zmiany = n;
35     czy_zmiany = true;
36     return *this;
37 }

```

Zadanie 3.20

Listing 8.49. rozwiązanie zadania 3.20

```
1 class tablica {
2     private:
3         int *tab;
4         int *ztab;
5         int roz;
6     public:
7         tablica(const tab_info&);
8         ~tablica();
9         int podaj_w(int);
10        void nadaj_w(unsigned int, int);
11        int licznik(int);
12        int rozmiar();
13    };
14
15    tablica::tablica(const tab_info& t) {
16        int zm, wart;
17        if (t.czy_rozmiar)
18            roz=t.w_rozmiar;
19        else
20            roz=100;
21        if (t.czy_wartosc)
22            wart=t.w_wartosc;
23        else
24            wart=0;
25        if (t.czy_zmiany)
26            zm=t.w_zmiany;
27        else
28            zm=10;
29        tab = new int[roz];
30        ztab = new int[roz];
31
32        for(int i=0;i<roz; i++){
33            tab[i]=wart;
34            ztab[i]=zm;
35        }
36    }
37
38    tablica::~tablica(){
39        delete [] tab;
40        delete [] ztab;
41    }
42
43    int tablica::podaj_w(int i){
44        return tab[i];
45    }
46
47    void tablica::nadaj_w(unsigned int i, int w){
48        if (ztab[i]!=0)
```

```

50         tab[i]=w;
51     if (ztab[i]>0)
52         ztab[i]--;
53 }
54 int tablica::licznik(int i){
55     return ztab[i];
56 }
57 int tablica::rozmiar(){
58     return roz;
59 }
60 }
```

Zadanie 3.21

Listing 8.50. rozwiązanie zadania 3.21

```

1 tablica* alokuj(){
2     return new tablica(tab_info().rozmiar(50).zmiany(1));
3 }
```

Zadanie 3.24

Listing 8.51. rozwiązanie zadania 3.24

```

1 class napis{
2 public:
3     std::string nap;
4     bool stala(){return false;}
5     bool stala() const {return true;}
6 };
7 }
```

Zadanie 3.25

Listing 8.52. rozwiązanie zadania 3.25

```

1 class napis2: public std::string{
2 public:
3     napis2(){}
4     napis2(const char nap[]):std::string(nap){}
5     napis2(const std::string& nap):std::string(nap){}
6     bool stala(){return false;}
7     bool stala() const {return true;}
8 };
```

Klasa `napis2` udostępnia nam wszystkie publiczne metody klasy `string` z biblioteki standardowej. Do tego obiekty klasy `napis2` możemy podawać jako argumenty wszędzie tam, gdzie argumentem może być referencja

do typu `string`. Dzięki temu klasę `napis2` możemy traktować jako bogatszą wersję typu `string`.

Zadanie 3.27

Listing 8.53. rozwiązanie zadania 3.27

```

1 bool porownaj(bazowa* arg1 , bazowa *arg2){
2     return (typeid(*arg1)==typeid(*arg2));
3 }
```

Zadanie 3.28

Listing 8.54. rozwiązanie zadania 3.28

```

1 pochodna * rzutuj(bazowa* b){
2     if (typeid(pochodna)==typeid(*b))
3         return dynamic_cast<pochodna*>(b);
4     else
5         return NULL;
6 }
```

Zadanie 3.30

Listing 8.55. rozwiązanie zadania 3.30

```

1 void wypisz(liczba2** tab , int n){
2     for(int i=0;i<n;i++)
3         if (typeid(rzeczywista2)==typeid(tab[ i ]))
4             std :: cout<<dynamic_cast<rzeczywista2*>
5                 (tab[ i ])>>wartosc<<std :: endl ;
6 }
```

8.4. Rozwiązania zadań z rozdziału 4

Zadanie 4.1

Listing 8.56. rozwiązanie zadania 4.1

```

1 class zespolone{
2 public:
3     double re ,im;
4 };
6 zespolone operator+(const zespolone& a , const zespolone& b){
7     zespolone z;
8     z.re=a.re+b.re;
```

```

10     z . im=a . im+b . im ;
11     return z ;
12 }
13
14 zespolone operator-(const zespolone& a, const zespolone& b){
15     zespolone z ;
16     z . re=a . re-b . re ;
17     z . im=a . im-b . im ;
18     return z ;
19 }
20
21 zespolone operator*(const zespolone& a, const zespolone& b){
22     zespolone z ;
23     z . re=a . re*b . re-a . im*b . im ;
24     z . im=a . im*b . re+a . re*b . im ;
25     return z ;
26 }
27
28 zespolone operator/(const zespolone& a, const zespolone& b){
29     zespolone z ;
30     z . re=(a . re*b . re+a . im*b . im)/(b . re*b . re+b . im*b . im) ;
31     z . im=(a . im*b . re-a . re*b . im)/(b . re*b . re+b . im*b . im) ;
32     return z ;
33 }
34 std :: ostream& operator<<(std :: ostream& out, const zespolone& z ){
35     out<<"re = "z . re<<"im = "z . im ;
36     return out ;
37 }
38
39 std :: istream& operator>>(std :: istream& in , zespolone& z ){
40     in>>z . re>>z . im ;
41     return in ;
42 }

```

Zadanie 4.4

Listing 8.57. rozwiązanie zadania 4.4

```

1 class tablica {
2 private:
3     int* tab;
4     unsigned int rozmiar;
5 public:
6     tablica();
7     tablica(unsigned int);
8     tablica(const tablica&);
9     ~tablica();
10    void resize(unsigned int);
11    const tablica& operator=(const tablica&);
12    int& operator[](unsigned int);

```

```
14     const int& operator []( unsigned int ) const;
15 };
16 tablica::tablica(){
17     tab=NULL;
18     rozmiar=0;
19 }
20 tablica::tablica(unsigned int n){
21     rozmiar=n;
22     tab = new int [rozmiar];
23 }
24
26 tablica::tablica(const tablica & t){
27     rozmiar = t.rozmiar;
28     tab = new int [rozmiar];
29     for(unsigned int i=0;i<rozmiar;i++)
30         tab [i]=t.tab [i];
31 }
32 tablica::~tablica(){
33     if (rozmiar>0)
34         delete [] tab;
35 }
36
38 void tablica::resize(unsigned int n){
39     if (n==0){
40         if (rozmiar>0)
41             delete [] tab;
42         tab=NULL;
43     }
44     else{
45         int * pom = new int [n];
46         for(unsigned int i=0;(i<n)&&(i<rozmiar);i++)
47             pom [i]=tab [i];
48         if (rozmiar>0)
49             delete [] tab;
50         tab = pom;
51     }
52     rozmiar = n;
53 }
54 const tablica& tablica::operator =(const tablica& t){
55     if (rozmiar>0)
56         delete [] tab;
57     rozmiar=t.rozmiar;
58     if (rozmiar==0){
59         tab=NULL;
60     }
61     else {
62         tab = new int [rozmiar];
```

```

64         for (unsigned int i=0; i<rozmiar; i++)
65             tab [ i ] = t . tab [ i ];
66     }
67     return t ;
68 }

70 int& tablica :: operator []( unsigned int i ){
71     return tab [ i ];
72 }

74 const int& tablica :: operator []( unsigned int i ) const{
75     return tab [ i ];
76 }

```

Zadanie 4.6

Listing 8.58. rozwiązań zadania 4.6

```

class napis{
2 private:
3     char *nap;
4     unsigned int rozm;
5 public:
6     napis();
7     napis(const char *);
8     napis(const napis& );
9     ~napis();
10    const napis& operator=(const napis& );
11    bool operator==(const napis&) const;
12    napis operator+(const napis&) const;
13    char operator[](unsigned int) const;
14    unsigned int rozmiar() const;
15    void wypisz() const;
16};

18 napis :: napis(){
19     nap=NULL;
20     rozm=0;
21 }
22
23 napis :: napis(const char * tab){
24     for(rozm=0; tab [ rozm ]!=0; rozm++);
25     if (rozm==0)
26         nap=NULL;
27     else{
28         nap=new char[ rozm ];
29         for (unsigned int i=0; i<rozm; i++)
30             nap [ i ] = tab [ i ];
31     }
32 }

```

```
34 napis::napis(const napis & np) {
35     rozm=np.rozm;
36     if (rozm==0)
37         nap=NULL;
38     else {
39         nap=new char[rozm];
40         for(unsigned int i=0;i<rozm; i++)
41             nap[i]=np.nap[i];
42     }
43 }
44 napis::~napis() {
45     if (rozm>0)
46         delete [] nap;
47 }
48 }

50 const napis& napis::operator=(const napis& np) {
51     if ((rozm>0)&&(rozm!=np.rozm))
52         delete [] nap;
53     rozm=np.rozm;
54     if (rozm==0)
55         nap=NULL;
56     else {
57         nap = new char[rozm];
58         for(unsigned int i=0;i<rozm; i++)
59             nap[i]=np.nap[i];
60     }
61     return np;
62 }

64 bool napis::operator==(const napis& np) const{
65     if (rozm!=np.rozm)
66         return false;
67     for(unsigned int i=0;i<rozm; i++)
68         if (nap[i]!=np.nap[i])
69             return false;
70     return true;
71 }
72 napis napis::operator+(const napis& np) const{
73     napis wynik;
74     wynik.rozm=rozm+np.rozm;
75     wynik.nap=new char[wynik.rozm];
76     unsigned int i=0;
77     for(; i<rozm; i++)
78         wynik.nap[i]=nap[i];
79     for(; i<rozm+np.rozm; i++)
80         wynik.nap[i]=np.nap[i-rozm];
81     return wynik;
82 }
```

```

86     char napis::operator []( unsigned int i ) const{
87         return nap[ i ];
88     }
89
90     unsigned int napis::rozmiar() const{
91         return rozm;
92     }
93
94     void napis::wypisz() const{
95         for(unsigned int i=0;i<rozm; i++)
96             std :: cout<<nap[ i ];
97         std :: cout<<std :: endl;
98     }

```

Zadanie 4.8

Listing 8.59. rozwiązań zadania 4.8

```

1 class komperator {
2     public:
3         virtual bool operator()(const napis&, const napis&)=0;
4     };
5
6     class alfabetyczna: public komperator{
7     public:
8         bool operator()(const napis&, const napis&);
9     };
10
11    bool alfabetyczna::operator ()(const napis& np1,
12                                     const napis& np2){
13        unsigned int i=0;
14        for (;(i<np1.rozmiar())&&(i<np2.rozmiar()); i++)
15            if (np1[ i]<np2[ i])
16                return true;
17            else if (np1[ i]>np2[ i])
18                return false;
19            if (np1.rozmiar()<np2.rozmiar())
20                return true;
21        return false;
22    }

```

Zadanie 4.9

Listing 8.60. rozwiązań zadania 4.9

```

void sortuj(napis* tab, unsigned int n, komperator& komp){
1     napis pom;
2     for(unsigned int i=0;i<n-1;i++)
3         for(unsigned int j=0;j<n-i-1;j++)
4             if (komp( tab[ j+1],tab[ j ])){

```

```

6           pom=tab [ j ];
7           tab [ j ]=tab [ j +1];
8           tab [ j +1]=pom;
9       }
10 }
```

Zadanie 4.12

Listing 8.61. rozwiązanie zadania 4.12

```

class macierz{
1 private:
2     double ** tab;
4 public:
5     macierz(unsigned int , unsigned int );
6     double& operator ()(unsigned int , unsigned int );
7 };
8
9     macierz :: macierz(unsigned int n, unsigned int m){
10    tab = new double*[n];
11    for(unsigned int i=0;i<m; i++)
12        tab [ i ] = new double[m];
13    }
14
15    double& macierz :: operator ()(unsigned int i , unsigned int j ){
16        return tab [ i ][ j ];
17 }
```

W przeciwnieństwie do operatora [] operator () nie ma z góry kreślonej arności i dzięki temu mogliśmy go przeciążyć tak, żeby na raz podawać mu oba indeksy w dwuwymiarowej tablicy. Osiągnięcie podobnego efektu w przypadku operatora [] jest możliwe, ale jest to bardziej skomplikowane.

Zadanie 4.14

Listing 8.62. rozwiązanie zadania 4.14

```

1 struct punkt2D{
2     double x,y;
3 };
4
5 struct punkt3D{
6     double x,y,z;
7
8     operator punkt2D() const;
9 };
10
11 punkt3D :: operator punkt2D() const{
12     punkt2D punkt;
```

```
13     punkt.x=x;
14     punkt.y=y;
15     return punkt;
16 }
```

Zadanie 4.18

Listing 8.63. rozwiązanie zadania 4.18

```
1 struct elisty {
2 struct element{
3     element * nastepny, poprzedni;
4     int i;
5 };
6     element * wsk, *pierwszy, *ostatni;
7
8     elisty& operator++(); //wersja prefiksowa
9     elisty operator++(int); //wersja postfiksowa
10    elisty& operator--(); //wersja prefiksowa
11    elisty operator--(int); //wersja postfiksowa
12 };
13
14    elisty& elisty ::operator ++(){
15        if (wsk!=ostatni)
16            wsk=wsk->nastepny;
17        return *this;
18    }
19
20    elisty elisty ::operator ++(int i){
21        elisty pom=*this;
22        if (wsk!=ostatni)
23            wsk=wsk->nastepny;
24        return pom;
25    }
26
27    elisty& elisty ::operator--(){
28        if (wsk!=pierwszy)
29            wsk=wsk->poprzedni;
30        return *this;
31    }
32
33    elisty elisty ::operator--(int i){
34        elisty pom=*this;
35        if (wsk!=pierwszy)
36            wsk=wsk->poprzedni;
37        return pom;
38    }
39 }
```

Zadanie 4.21

Listing 8.64. rozwiązanie zadania 4.21

```
1 class n_int {
2     private:
3         static unsigned int liczba_ob, liczba_tab;
4
5     public:
6         int liczba;
7         void* operator new(size_t);
8         void* operator new[](size_t);
9         void operator delete(void*);
10        void operator delete[](void*);
11        static void wypisz();
12    };
13
14    unsigned int n_int::liczba_ob=0, n_int::liczba_tab=0;
15
16
17    void* n_int::operator new(size_t rozmiar) {
18        liczba_ob++;
19        return malloc(rozmiar);
20    }
21
22    void* n_int::operator new[](size_t rozmiar) {
23        liczba_tab++;
24        return malloc(rozmiar);
25    }
26
27    void n_int::operator delete(void* wsk){
28        liczba_ob--;
29        free(wsk);
30    }
31
32    void n_int::operator delete[](void* wsk){
33        liczba_tab--;
34        free(wsk);
35    }
36
37    void n_int::wypisz(){
38        std::cout<<"Zaallokowano dynamicznie "<<liczba_ob
39                      <<" pojedynczych obiektow typu n_int"
40                      <<std::endl;
41    }
42    std::cout<<" oraz "<<liczba_tab<<" dynamicznych tablic tego typu .";
43}
```

Dla uproszczenia w powyższym rozwiązaniu nie uwzględniliśmy faktu, że standardowy operator **new** nie zwraca nigdy wartości **NULL**, a w przypadku niepowodzenia alokacji pamięci rzuca wyjątek. Warto pamiętać,

że standardowy operator `new` posiada specjalną wersję nierzucającą wyjątków.

8.5. Rozwiązania zadań z rozdziału 5

Zadanie 5.1

Listing 8.65. rozwiązanie zadania 5.1

```
template<class T>
2 T funkcja(T a, T b, T c){
    return a-b+c;
4 }
```

Zadanie 5.4

Listing 8.66. rozwiązanie zadania 5.4

```
template<class T>
2 T najmniejszy(T* tab, unsigned int n) {
    unsigned int min=0;
4     for(unsigned int i=1;i<n; i++)
        if (tab[ i]<tab[ min])
6         min=i ;
    return tab[ min];
8 }
```

Zadanie 5.5

Listing 8.67. rozwiązanie zadania 5.5

```
template<class T>
2 void zamien(T& a, T& b){
    T c=a;
4     a=b;
    b=c;
6 }
```

Zadanie 5.6

Listing 8.68. rozwiązanie zadania 5.6

```
template<class T>
2 void wypisz(T* a){
    std::cout<<*a<<std::endl;
4 }
```

Zadanie 5.7

Listing 8.69. rozwiązanie zadania 5.7

```
1 template<class T>
2 void wypisz_zakres(T* pocz, T* kon) {
    while(pocz!=kon){
        wypisz<T>(pocz);
        pocz++;
    }
}
```

W rozwiązaniu wywołując szablon `wypisz`, podaliśmy wprost, choć nie musieliśmy, parametr szablonu. Często, choć nie zawsze, możemy pominąć parametry przy wywoływaniu szablonu funkcji (kompilator dedukuje wówczas parametry szablonu z typów argumentów).

Zadanie 5.8

Listing 8.70. rozwiązanie zadania 5.8

```
1 template<class T>
2 void wypisz_tab(T * tab, unsigned int n){
3     wypisz_zakres<T>(tab, tab+n);
4 }
```

Zadanie 5.9

Listing 8.71. rozwiązanie zadania 5.9

```
1 template<class T>
2 void wypisz(T a){
    std::cout<<*a<<std::endl;
4 }

6 template<class T>
7 void wypisz_zakres(T pocz, T kon) {
8     while(pocz!=kon){
9         wypisz(pocz);
10        pocz++;
11    }
12 }

14 template<class T>
15 void wypisz_tab(T tab, unsigned int n){
16     wypisz_zakres(tab, tab+n);
17 }
18

19 template<class T>
20 void wypisz_vec(const std::vector<T>& vec){
```

```

22 } wypisz_zakres(vec.begin(), vec.end());

```

Zadanie 5.10

Listing 8.72. rozwiązań zadania 5.10

```

1 template<class T>
2     bool operator>(const T& a, const T& b){
3         return !(a<b);
4     }

```

Zadanie 5.11

Listing 8.73. rozwiązań zadania 5.11

```

template<class T1, class T2>
2 bool typ(T1* a, T2* b){
    return typeid(*a)==typeid(*b));
4 }

```

Zadanie 5.12

Listing 8.74. rozwiązań zadania 5.12

```

template<class T1, class T2>
2 void minimum(T1 tab[], unsigned int n, T2 f){
    unsigned int min;
4     for(int i=1;i<n;i++)
        if (f(tab[i],tab[min]))
6         min=i;
    return tab[min];
8 }

```

Zadanie 5.13

Listing 8.75. rozwiązań zadania 5.13

```

class mniejszy {
2 public:
    bool operator()(int a, int b){return (a<b);}
4 };

6 class wiekszy {
public:
8     bool operator()(int a, int b){return (a>b);}
10 };
void skrajne(int tab[], unsigned int n){

```

```
12     std :: cout<<"Najmniejszy_element_tablicy_to_"
           <<minimum(tab ,n, mniejszy ())<<std :: endl ;
14     std :: cout<<"Najwiekszy_element_tablicy_to_"
           <<minimum(tab ,n, wiekszy ())<<std :: endl ;
16 }
```

Zadanie 5.16

Listing 8.76. rozwiązanie zadania 5.16

```
template<unsigned int n>
2 void przepisz(int m1[][n], int m2[][n]){
    for(unsigned int i=0;i<n; i++)
        for(unsigned int j=0;j<n; j++)
            m2[i][j]=m1[i][j];
6 }
```

Zauważmy, że w języku C++ nie możemy tworzyć wielowymiarowych tablic automatycznych o rozmiarach podanych przez zmienne. Szablony takie jak powyżej pozwalają nam obejść część związanych z tym niedogodności, ale nie pozwalają na tworzenie tablic o rozmiarze nieznanym w trakcie komplikacji programu.

Zadanie 5.17

Listing 8.77. rozwiązanie zadania 5.17

```
class tablica{
2 public:
    int * tab;
4     unsigned int rozmiar_tab;

6     tablica(unsigned int);
    ~tablica();
8     template<class T>
    void sortuj(T);
10 }

12 tablica::tablica(unsigned int n){
    tab = new int[n];
14     rozmiar_tab = n;
}
16     tablica::~tablica(){
18         delete [] tab;
}
20     template<class T>
22 void tablica::sortuj(T f){
```

```

24     int pom;
25     for(unsigned int i=0;i<rozmiar_tab-1;i++)
26         for(unsigned int j=0;j<rozmiar_tab-i-1;j++)
27             if (f(tab[j+1],tab[j])) {
28                 pom=tab[j];
29                 tab[j]=tab[j+1];
30                 tab[j+1]=pom;
31             }
32     }

```

Zadanie 5.22

Listing 8.78. rozwiązanie zadania 5.22

```

1 template<class T>
2     class liczba {
3     private:
4         T wart;
5     public:
6         void wczytaj();
7         void wypisz();
8     };
9
10    template<class T>
11    void liczba<T>::wczytaj(){
12        std :: cout<<"Podaj wartość ";
13        std :: cin>>wart;
14    }
15
16    template <class T>
17    void liczba<T>::wypisz(){
18        std :: cout<<"Przechowywana wartość " <<wart <<std :: endl;
19    }

```

Zadanie 5.25

Listing 8.79. rozwiązanie zadania 5.25

```

1 template<class T1, class T2>
2     class para{
3     public:
4         T1 pierwsze;
5         T2 drugie;
6
7         para(T1, T2);
8     };
9
10    template<class T1, class T2>
11    para<T1,T2>::para(T1 a, T2 b){
12        pierwsze = a;
13    }

```

```

13     drugie = b;
14 }
15
16 template<class T1, class T2>
17 bool operator<(para<T1,T2> a, para<T1,T2> b){
18     if (a.pierwsze<b.pierwsze)
19         return true;
20     else if ((a.pierwsze==b.pierwsze)&&(a.drugie<b.drugie))
21         return true;
22     return false;
23 }
```

Zadanie 5.27

Listing 8.80. rozwiązanie zadania 5.27

```

1 template<class T1,unsigned int n>
2     class tablica2 {
3     public:
4         T1 tab[n];
5         static const unsigned int rozmiar = n;
6     };
```

Ponieważ tablica `tab` jest tablicą automatyczną, jest ona poprawnie kopowana przez standardowy konstruktor kopiący i operator przypisania, zaś pamięć po niej jest automatycznie zwalniana w trakcie zwalniania pamięci przez obiekt, który ją przechowuje. W związku z tym nie ma potrzeby definiowania w szablonie klasy `tablica2` konstruktora kopiącego, operatora przypisania czy destruktora.

Zadanie 5.28

Listing 8.81. rozwiązanie zadania 5.28

```

template<unsigned int n>
2 class wektor {
3     public:
4         double tab[n];
5     };
6
7 template<unsigned int n>
8 wektor<n> operator+(const wektor<n>& a, const wektor<n>& b) {
9     wektor<n> wynik;
10    for (unsigned int i=0;i<n; i++)
11        wynik.tab[i]=a.tab[i]+b.tab[i];
12    return wynik;
13 }
14 template<unsigned int n>
```

```

16 wektor<n> operator-(const wektor<n>& a, const wektor<n>& b) {
    wektor<n> wynik;
18     for(unsigned int i=0;i<n; i++)
        wynik.tab[i]=a.tab[i]-b.tab[i];
20     return wynik;
}
22
23     template<unsigned int n>
24     wektor<n> operator*(int a, const wektor<n>& b) {
            wektor<n> wynik;
26         for(unsigned int i=0;i<n; i++)
            wynik.tab[i]=a*b.tab[i];
28         return wynik;
}

```

Zwróćmy uwagę, że w powyższym przykładzie nie potrzebujemy przechowywać ilości wymiarów wektora w polu klasy, gdyż jest to parametr szablonu. Co więcej, próba dodania do siebie wektorów o różnej liczbie wymiarów zostanie wychwycona już na etapie kompilacji.

Zadania 5.29–5.31

Listing 8.82. rozwiązywanie zadań 5.29–5.31

```

1 template<unsigned int n>
2     class punkt {
3     public:
4         double tab[n];
5
6         punkt(){}
7         punkt(const punkt<n+1>&);
8     };
9
10    template<unsigned int n>
11    punkt<n>::punkt(const punkt<n+1> & p) {
12        for(unsigned int i=0;i<n; i++)
13            tab[i]=p.tab[i];
14    }
15
16    template<unsigned int n>
17    punkt<n-1> zrzutuj(const punkt<n>& p) {
18        punkt<n-1> wynik;
19        for(unsigned int i=0;i<n-1; i++)
20            wynik.tab[i]=p.tab[i];
21        return wynik;
}

```

Funkcja `zrzutuj` jest wywoływana tylko wtedy gdy ją jawnie wywołamy. W przypadku zdefiniowanego powyżej konstruktora może być on wywołany

także niejawnie (na przykład gdy po lewej stronie operatora przypisania jest zmienna typu `punkt<n>` a po prawej typu `punkt<n+1>`). Niejawne wywoływanie konstruktora może być wygodne, ale niesie za sobą także pewne niebezpieczeństwa (przykładowo możemy nie zauważyc, że dokonaliśmy przypisania pomiędzy punktami o różnej liczbie współrzędnych w wyniku czego utraciliśmy wartość ostatniej współrzędnej).

Zadanie 5.38

Listing 8.83. rozwiązanie zadania 5.38

```

1 template<class T>
2 class lista {
3     public:
4         virtual void wstaw_z_przodu(T)=0;
5         virtual T pierwszy()=0;
6         virtual void usun_pierwszy()=0;
7         virtual void wstaw_z_tylu(T)=0;
8         virtual T ostatni()=0;
9         virtual void usun_ostatni()=0;
10        virtual bool pusta()=0;
11        virtual ~lista(){}
12    };

```

Zadanie 5.39

Listing 8.84. rozwiązanie zadania 5.39

```

1 template<class T>
2 class lista_wskaz: public lista<T>{
3     private:
4         struct element {
5             T wartosc;
6             element * nastepny, *poprzedni;
7         };
8         element *pierw, *ost;
9     public:
10        lista_wskaz();
11        ~lista_wskaz();
12        void wstaw_z_przodu(T);
13        T pierwszy();
14        void usun_pierwszy();
15        void wstaw_z_tylu(T);
16        T ostatni();
17        void usun_ostatni();
18        bool pusta();
19    };
20 template<class T>
21 lista_wskaz<T>::lista_wskaz(){
22     pierw=ost=NULL;

```

```

        }

24    template<class T>
26 lista_wskaz<T>::~lista_wskaz() {
27     if (pierw!=NULL){
28         element* pom;
29         while(pierw!=ost){
30             pom=pierw;
31             pierw=pierw->nastepny;
32             delete pom;
33         }
34         delete pierw;
35     }
36 }

38 template<class T>
39     void lista_wskaz<T>::wstaw_z_przodu(T t){
40     if (pierw==NULL){
41         pierw=ost=new element;
42     }
43     else{
44         pierw->poprzedni = new element;
45         pierw=pierw->poprzedni;
46     }
47     pierw->wartosc = t;
48 }

50 template<class T>
51     T lista_wskaz<T>::pierwszy(){
52     return pierw->wartosc;
53 }
54

56 template<class T>
57     void lista_wskaz<T>::usun_pierwszy(){
58         element *pom=pierw;
59         if (pierw==ost)
60             pierw=ost=NULL;
61         else
62             pierw=pierw->nastepny;
63             delete pom;
64 }

66 template<class T>
67     void lista_wskaz<T>::wstaw_z_tylu(T t){
68     if (ost==NULL){
69         pierw=ost=new element;
70     }
71     else{
72         ost->nastepny = new element;
73         ost=ost->nastepny;
74     }
75 }
```

```

74      }
    ost->wartosc = t;
76 }

78 template<class T>
    T lista_wskaz<T>::ostatni(){
80     return ost->wartosc;
81 }
82

84 template<class T>
    void lista_wskaz<T>::usun_ostatni(){
86     element *pom=ost;
87     if (pierw==ost)
88         pierw=ost=NULL;
89     else
90         ost=ost->poprzedni;
91     delete pom;
92 }

94 template<class T>
    bool lista_wskaz<T>::pusta(){
96     return (pierw==NULL);
97 }
```

Zadanie 5.42

Listing 8.85. rozwiązanie zadania 5.42

```

1 template<class T>
2     class kolejka {
3     private:
4         lista<T> * list;
5     public:
6         kolejka(lista<T>* l){list = l;}
7         kolejka(){list = new lista_wskaz<T>};
8         ~kolejka(){delete list;}
9         T pierwszy(){return list->pierwszy();}
10        void usun_pierwszy(){list->usun_pierwszy();}
11        void dodaj_na_koniec(T t){list->wstaw_z_tylu(t);}
12        bool pusta(){return list->pusta();}
13    };

```

Zadanie 5.43

Listing 8.86. rozwiązanie zadania 5.43

```

1 template<class T>
2     class tablica {
3     private:

```

```
      T * tab;
5     unsigned int roz;
public:
7     tablica();
8     tablica(unsigned int);
9     tablica(const tablica&);
~tablica();
11    unsigned int rozmiar() const;
12    const tablica & operator=(const tablica&);
13    T& operator[](unsigned int);
14    const T& operator[](unsigned int) const;
15 }

17 template<class T>
18     tablica<T>::tablica(){
19         roz=0;
20     }
21
22 template<class T>
23     tablica<T>::tablica(unsigned int n){
24         roz=n;
25         tab = new T[roz];
26     }
27
28 template<class T>
29     tablica<T>::tablica(const tablica<T> & t){
30         roz=t.roz;
31         tab = new T[roz];
32         for(unsigned int i=0;i<roz;i++)
33             tab[i]=t.tab[i];
34     }
35

36 template<class T>
37     tablica<T>::~tablica(){
38         delete [] tab;
39     }
40
41 template<class T>
42     unsigned int tablica<T>::rozmiar() const{
43         return roz;
44     }
45

46 template<class T>
47     const tablica<T>& tablica<T>::operator=(const tablica<T>& t){
48         if (roz>0)
49             delete [] tab;
50         roz=t.roz;
51         tab = new T[roz];
52         for(unsigned int i=0;i<roz;i++)
53             tab[i]=t.tab[i];
54     }
```

```

55     return t;
56 }
57
58 template<class T>
59 T& tablica<T>::operator[]( unsigned int i){
60     return tab[i];
61 }

63 template<class T>
64 const T& tablica<T>::operator[]( unsigned int i) const{
65     return tab[i];
66 }

```

Zadanie 5.44

Listing 8.87. rozwiązanie zadania 5.44

```

typedef tablica<char> napis;
2
3     napis operator+(const napis& a, const napis& b ){
4         napis c(a.rozmiar()+b.rozmiar());
5         for(unsigned int i=0;i<a.rozmiar();i++)
6             c[i]=a[i];
7         for(unsigned int i=a.rozmiar();i<c.rozmiar();i++)
8             c[i]=b[i-a.rozmiar()];
9         return c;
10    }

```

Zadanie 5.46

Listing 8.88. rozwiązanie zadania 5.46

```

template<class T>
2 class tablica {
3     private:
4         T * tab;
5         unsigned int roz;
6     public:
7         tablica();
8         tablica(unsigned int);
9         tablica(const tablica&);
10        ~tablica();
11        unsigned int rozmiar() const;
12        const tablica & operator=(const tablica&);
13        T& operator[]( unsigned int);
14        const T& operator[]( unsigned int) const;
15        template<class R>
16        bool operator==(const tablica<R>&) const;
17    };
18

```

```

18     template<class T>
19     template<class R>
20         bool tablica<T>::operator==(const tablica<R> & t) const{
21             if (roz!=t.rozmiar())
22                 return false;
23             for(unsigned int i=0; i<roz; i++)
24                 if (tab[i]!=t[i])
25                     return false;
26             return true;
27 }
28 }
```

Pominęliśmy definicje wszystkich klas poza operatorem `==`, gdyż czytelnik może je znaleźć w rozwiążaniu zadania 5.43. Zwróćmy uwagę na dwie rzeczy. Po pierwsze w definicji operatora mamy szablon szablonów, a nie jeden szablon o dwóch parametrach. Po drugie w ciele operatora używamy wyłącznie publicznych metod obiektu `t`. Dzieje się tak, gdyż dla `T` różnego od `R`, `tablica<T>` i `tablica<R>` to dwa różne typy, a więc metody klasy `tablica<T>` nie mają dostępu do prywatnych pól klasy `tablica<R>`. Zauważmy też, że powyższy operator porównania działa także dla klasy `napis` z zadania 5.44.

Zadanie 5.48

Listing 8.89. zawartość pliku glowny.cc

```

1 #include <iostream>
2 #include "tablica.h"

4 int main() {
5     unsigned int n,pom;
6     std :: cout<<" ile liczb chcesz podac ";
7     std :: cin>>n;
8     tablica<int> tab(n);
9     std :: cout<<" podaj " <<n<<" liczb : "<<std :: endl;
10    for(unsigned int i=0;i<n; i++)
11        std :: cin>>tab[ i ];
12
13    for(unsigned int i=0;i<tab.rozmiar()-1; i++)
14        for(unsigned int j=0;j<tab.rozmiar()-i-1; j++)
15            if (tab[ j]>tab[ j+1]){
16                pom=tab[ j ];
17                tab[ j]=tab[ j+1];
18                tab[ j+1]=pom;
19            }
20    for(unsigned int i=0;i<tab.rozmiar(); i++)
21        std :: cout<<tab[ i]<<std :: endl;
22 }
```

Całe rozwiązanie zadania 5.43 powinno zostać umieszczone w pliku `tablica.h`. Modułów z szablonami nie rozdziela się na plik nagłówkowy i kod, gdyż szablony muszą być kompilowane razem z jednostkami kompilacji, w których są użyte (inaczej kompilator nie wie, jakie konkretyzacje będą potrzebne).

8.6. Rozwiązania zadań z rozdziału 6

Zadanie 6.1

Listing 8.90. rozwiązanie zadania 6.1

```
1 #include <iostream>
2 #include <vector>
3
4 int main() {
5     std::vector<int> v;
6     int liczba;
7     do {
8         std::cin >> liczba;
9         v.push_back(liczba);
10    } while(liczba);
11
12    for(unsigned int i=0;i<v.size();i++)
13        std::cout << v[i] << std::endl;
14}
```

Zadanie 6.3 Poniżej rozwiązani poszczególnych podpunktów zadania 6.3:

Listing 8.91. rozwiązanie zadania 6.3a

```
1 void pomieszaj(std::vector<int>& v) {
2     std::reverse(v.begin(), v.end());
3 }
```

Listing 8.92. rozwiązanie zadania 6.3b

```
1 void pomieszaj(std::vector<int>& v) {
2     std::sort(v.begin(), v.end());
3 }
```

Listing 8.93. rozwiązanie zadania 6.3c

```
1 void pomieszaj(std::vector<int>& v) {
2     std::sort(v.begin(), v.end(), std::greater<int>());
3 }
```

Listing 8.94. rozwiązań zadania 6.3d

```

1 class wartosc_bezwzgledna{
2     public:
3         bool operator()(int a, int b){
4             if (a<0)
5                 a==a;
6             if (b<0)
7                 b==b;
8             return (a<b);
9         }
10    };
11
12    void pomieszaj(std::vector<int>& v){
13        std::sort(v.begin(),v.end(),wartosc_bezwzgledna());
14    }

```

Listing 8.95. rozwiązań zadania 6.3e

```

1 class reszta{
2     public:
3         bool operator()(int a, int b){
4             return (a%1000<b%1000);
5         }
6     };
7
8    void pomieszaj(std::vector<int>& v){
9        std::sort(v.begin(),v.end(),reszta());
10    }

```

Zadanie 6.4 W przypadku gdy funkcja ma działać dla kontenera `array` o określonym z góry rozmiarze, jego użycie nie różni się od użycia wektora:

Listing 8.96. rozwiązań zadania 6.4

```

1 void pomieszaj(std::array<int,10>& v){
2     std::reverse(v.begin(),v.end());
3 }

```

Jednak, gdy chcemy aby funkcję można było użyć dla kontenerów o różnych rozmiarach musimy stworzyć szablon funkcji

Listing 8.97. rozwiązań zadania 6.4

```

1 template<unsigned int n>
2     void pomieszaj(std::array<int,n>& v){
3         std::reverse(v.begin(),v.end());
4     }

```

Zadanie 6.5

Listing 8.98. rozwiązanie zadania 6.5

```

1 class macierz {
2 private:
3     std::vector<std::vector<int>> v;
4 public:
5     macierz(unsigned int n, unsigned int m):
6         v(n, std::vector<int>(m){})
7     int& operator()(unsigned int i, unsigned int j)
8         {return v[i][j];}
9 };

```

Zadanie 6.9

Listing 8.99. rozwiązanie zadania 6.9

```

1 #include <iostream>
2 #include <list>
3
4 int main() {
5     std::list<int> l;
6     unsigned int n;
7     int pom;
8     std::cin>>n;
9     for(unsigned int i=0;i<n; i++){
10         std::cin>>pom;
11         if (pom>=0)
12             l.push_back(pom);
13         else
14             l.push_front(pom);
15     }
16     for(std::list<int>::iterator it=l.begin(); it!=l.end(); it++)
17         std::cout<<*it<<std::endl;
18 }

```

Zadanie 6.12

Listing 8.100. rozwiązanie zadania 6.12

```

1 class lista_napisow {
2 private:
3     std::queue<std::string, std::list<std::string>> kolejka;
4 public:
5     void wczytaj();
6     void wypisz();
7 };
8
9 void lista_napisow::wczytaj(){
10     std::string s;

```

```

12     std :: cin>>s;
13     kolejka .push(s);
14 }
14
15 void lista_napisow::wypisz(){
16     for(unsigned int i=0;(i<10)&&(!kolejka.empty());i++){
17         std :: cout<<kolejka.front()<<std :: endl;
18         kolejka.pop();
19     }
20 }
```

Zadanie 6.19

Listing 8.101. rozwiązań zadania 6.19

```

class porownaj {
2 public:
    porownaj(){}
4     bool operator()(const dane& d1, const dane& d2){
5         if (d1.punkty<d2.punkty)
6             return true;
7         else if ((d1.punkty==d2.punkty)&&(d1.pesel<d2.pesel))
8             return true;
9         else
10            return false;
11
12    }
13};

14
15 class rekrutacja {
16 private:
    std :: priority_queue<dane,
18     std :: vector<dane>, porownaj> kolejka;
public:
20     void dodaj(std :: string , std :: string , std :: string ,
21                 std :: string , unsigned int );
22     dane najlepszy();
23     void usun();
24 };

26 void rekrutacja::dodaj(std :: string imie,
27                         std :: string nazwisko, std :: string pesel,
28                         std :: string tel, unsigned int punkty){
29     dane pom;
30     pom.imie = imie;
31     pom.nazwisko = nazwisko;
32     pom.pesel = pesel;
33     pom.tel = tel;
34     pom.punkty = punkty;
35     kolejka.push(pom);
36 }
```

```
38 dane rekrutacja :: najlepszy () {
        return kolejka . top () ;
40 }

42 void rekrutacja :: usun () {
        kolejka . pop () ;
44 }
```

Przy rozwiązywaniu zadania założyliśmy, że nie może być w bazie dwóch różnych osób o takim samym numerze pesel.

Zadania 6.24–6.25

Listing 8.102. rozwiązanie zadań 6.24–6.25

```
1 class slownik {
2     private:
3         std :: set<std :: string> zbior;
4     public:
5         void dodaj(const std :: string& );
6         bool znajdz(const std :: string& );
7         void zakres(const std :: string&, const std :: string& );
8     };
9
10    void slownik :: dodaj(const std :: string& slowo){
11        zbior . insert (slowo) ;
12    }
13
14    bool slownik :: znajdz(const std :: string& slowo){
15        return (zbior . find (slowo)!=zbior . end ());
16    }
17
18    void slownik :: zakres(const std :: string& slowo1 ,
19                           const std :: string& slowo2){
20        std :: set<std :: string >:: iterator pocz , kon;
21        pocz=zbior . upper _ bound (slowo1);
22        kon=zbior . lower _ bound (slowo2);
23        while(pocz!=kon){
24            std :: cout << *pocz << std :: endl ;
25            pocz++;
26        }
27    }
```

Zadanie 6.32

Listing 8.103. rozwiązanie zadania 6.32

```
1 template<class T>
2     void wypisz(T pocz , T kon){
```

```

3     while( pocz!=kon) {
4         std :: cout<<*pocz<<std :: endl ;
5         pocz++;
6     }
7 }
```

Zadanie 6.38

Listing 8.104. rozwiązań zadania 6.38

```

1 struct bezwzgledna{
2     void operator()( int& i ){
3         if ( i<0)
4             i=-i ;
5     }
6 };
7
void vec_abs( std :: vector<int>& v){
8     std :: for_each(v.begin() ,v.end() ,bezwzgledna());
9 }
```

Zadanie 6.39 W rozwiązaniach wszystkich podpunktów będziemy wykorzystywali następującą strukturę:

Listing 8.105. rozwiązań zadania 6.39

```

struct podzielny{
1   bool operator()( int i ) const{
2       if (( i%2==0) || ( i%3==0) || ( i%5==0) || ( i%7==0))
3           return true ;
4       return false ;
5   }
6 };
```

Poniżej rozwiązania dla poszczególnych podpunktów:

Listing 8.106. rozwiązań zadania 6.39a

```

1 bool ktorykolwiek( const std :: vector<int>& v){
2     return any_of(v.begin() ,v.end() ,podzielny());
3 }
```

Listing 8.107. rozwiązań zadania 6.39b

```

1 bool zaden( const std :: vector<int>& v){
2     return none_of(v.begin() ,v.end() ,podzielny());
3 }
```

Listing 8.108. rozwiązanie zadania 6.39c

```

1 bool kazdy(const std::vector<int>& v){
    return all_of(v.begin(), v.end(), podzielny());
3 }
```

Zadanie 6.42

Listing 8.109. rozwiązanie zadania 6.42

```

1 bool palindrom(std::string s){
    return std::equal(s.begin(), s.end(), s.rbegin());
3 }
```

W rozwiązaniu tego zadania warto zwrócić uwagę na użycie `v.rbegin()` – iteratora poruszającego się od końca (`reverse_iterator`).

8.7. Rozwiązania zadań z rozdziału 7**Zadanie 7.1**

Listing 8.110. rozwiązanie zadania 7.1

```

1 double* alokuj_wer_1(unsigned int n){
    try {
3         return new double[n];
    }
5     catch(...) {
        std::cout << "Nieudana proba alokacji tablicy "
7                         << std::endl;
        return NULL;
9    }
11   }
13   double* alokuj_wer_2(unsigned int n){
14       double * pom = new (std::nothrow) double[n];
15       if (pom==NULL)
16           std::cout << "Nieudana proba alokacji tablicy "
18                         << std::endl;
17   return pom;
}
```

Zadanie 7.2

Listing 8.111. rozwiązanie zadania 7.2

```

void zamien(std::vector<int> v1, std::vector<int> v2,
2             std::vector<int> v3, unsigned int l1,
              unsigned int l2, unsigned int l3){
```

```

4     if ((v1.size ()<=11) || (v2.size ()<=12) || (v3.size ()<=13))
5         throw std::out_of_range("indeks_spoza_zakresu");
6     else {
7         int pom;
8         pom = v1[11];
9         v1[11]=v3[13];
10        v3[13]=v2[12];
11        v2[12]=pom;
12    }
13 }
```

Zadanie 7.4

Listing 8.112. rozwiązanie zadania 7.4

```

1 class blad {
2     std::string opis;
3 public:
4     blad(const std::string& s):opis(s){}
5     std::string co(){return opis;}
6 };
7
8 class spoza_dziedziny: public blad{
9 public:
10    spoza_dziedziny(std::string s):blad(s){}
11    spoza_dziedziny():blad("argumenty_spoza_dziedziny"){}
12 };
13
14 class dzielenie_przez_zero: public spoza_dziedziny{
15 public:
16     dzielenie_przez_zero():
17     spoza_dziedziny("dzielenie_przez_zero"){}
18 };
19
20 class pierwiastek_z_ujemnej: public spoza_dziedziny {
21 public:
22     pierwiastek_z_ujemnej():
23     spoza_dziedziny("pierwiastek_z_liczby_ujemnej"){}
24 };
25
26
27 class niezainicjowana_zmienne: public blad {
28 public:
29     niezainicjowana_zmienne():
30     blad("uzycie_niezainicjowanej_zmiennej"){}
31 };
32
33
34 class przypisanie_do_stalej: public blad {
35 public:
36     przypisanie_do_stalej():


```

```

37     blad( "proba_{przypisania_{nowej_{wartosci_{do_{stalej}}}}{}{"
38   };
39
40   class wynik_spoza_zakresu: public blad {
41   public:
42     wynik_spoza_zakresu():
43       blad( "wynik_{spoza_{zakresu_{uzytego_{typu}}}}{}{"
44     };
45
46   class inny_blad: public blad {
47   public:
48     inny_blad(): blad( "inny_{blad}" ){}
49 }

```

Powyżej stworzyliśmy swoją pełną hierarchię wyjątków. W praktyce dobrym pomysłem jest umieszczenie swoich wyjątków w hierarchii standardowych wyjątków poprzez dziedziczenie po odpowiednich klasach wyjątków.

Zadanie 7.5

Listing 8.113. rozwiązanie zadania 7.5a

```

1 double pot(double p, double w){
2   if ((!std::isfinite(p))||(!std::isfinite(w)))
3     throw spoza_dziedziny();
4   if ((p==0)&&(w<=0))
5     throw spoza_dziedziny();
6   if ((p<0)&&(w!=trunc(w)))
7     throw spoza_dziedziny();
8   errno = 0;
9   double pom = pow(p,w);
10  if (errno==ERANGE)
11    throw wynik_spoza_zakresu();
12  if (errno==EDOM)
13    throw inny_blad();
14  return pom;
15 }

```

Listing 8.114. rozwiązanie zadania 7.5b

```

1 double pierw(double p{
2   if (!std::isfinite(p))
3     throw spoza_dziedziny();
4   errno = 0;
5   if (p<0)
6     throw pierwiastek_z_ujemnej();
7   double pom= sqrt(p);
8   if (errno==ERANGE)
9     throw wynik_spoza_zakresu();

```

```

11     if (errno==EDOM)
12         throw spoza_dziedziny();
13     return pom;
14 }
```

W powyższych przypadkach nie mogliśmy w pełni polegać na kodach zapisanych w `errno`, gdyż nasze klasyfikacja błędów nie pokrywa się z tą stosowaną przez funkcje standardowe (nie tylko dlatego, że nasza klasyfikacja jest bardziej szczegółowa).

W rozwiązaniu użyliśmy funkcji `isfinite` sprawdzającej czy wartością podaną w jej argumentie nie jest `inf` ani `nan`. Funkcja `isfinite` została dodana oficjalnie dopiero w standardzie C++11x, ale już wcześniej była obsługiwana przez wiele kompilatorów (w języku C ta funkcja jest dostępna od standardu C99).

Zadanie 7.6

Listing 8.115. rozwiązanie zadania 7.6a

```

1 double dodaj(double a, double b){
2     if ((!std::isfinite(a)) || (!std::isfinite(b))){
3         errno = EDOM;
4         return 0;
5     }
6     double c= a+b;
7     if (!std::isfinite(c))
8         errno = ERANGE;
9     return c;
10 }
11
12 double odejmij(double a, double b){
13     if ((!std::isfinite(a)) || (!std::isfinite(b))){
14         errno = EDOM;
15         return 0;
16     }
17     double c= a-b;
18     if (!std::isfinite(c))
19         errno = ERANGE;
20     return c;
21 }
22
23 double pomnoz(double a, double b){
24     if ((!std::isfinite(a)) || (!std::isfinite(b))){
25         errno = EDOM;
26         return 0;
27     }
28     double c= a*b;
29     if (!std::isfinite(c))
```

```
31         errno = ERANGE;
32     }
33
34     double podziel(double a, double b){
35         if ((!std::isfinite(a)) || (!std::isfinite(b)) || (b==0)){
36             errno = EDOM;
37         }
38         double c= a/b;
39         if (!std::isfinite(c))
40             errno = ERANGE;
41         return c;
42     }
43 }
```

Listing 8.116. rozwiązanie zadania 7.6b

```
1 double dodaj(double a, double b){
2     if ((!std::isfinite(a)) || (!std::isfinite(b)))
3         throw spoza_dziedziny();
4     double c= a+b;
5     if (!std::isfinite(c))
6         throw wynik_spoza_zakresu();
7     return c;
8 }
9
10 double odejmij(double a, double b){
11     if ((!std::isfinite(a)) || (!std::isfinite(b)))
12         throw spoza_dziedziny();
13     double c= a-b;
14     if (!std::isfinite(c))
15         throw wynik_spoza_zakresu();
16     return c;
17 }
18
19 double pomnoz(double a, double b){
20     if ((!std::isfinite(a)) || (!std::isfinite(b)))
21         throw spoza_dziedziny();
22     double c= a*b;
23     if (!std::isfinite(c))
24         throw wynik_spoza_zakresu();
25     return c;
26 }
27
28 double podziel(double a, double b){
29     if ((!std::isfinite(a)) || (!std::isfinite(b)))
30         throw spoza_dziedziny();
31     if (b==0)
32         throw dzielenie_przez_zero();
33     double c= a/b;
```

```

35     if (!std::isfinite(c))
36         throw wynik_spoza_zakresu();
37     return c;

```

Zadanie 7.7

Listing 8.117. rozwiązań zadania 7.7 wersja oparta na wyjątkach

```

1 void rownanie(double a, double b, double c){
2     try {
3         double delta = odejmij(pomnoz(b,b),pomnoz(4,
4                                         pomnoz(a,c)));
5         double pierw_delta = pierw(delta);
6         double x1 = podziel(dodaj(odejmij(0,b),pierw_delta),
7                               pomnoz(2,a));
8         double x2 = podziel(odejmij(odejmij(0,b),pierw_delta),
9                               pomnoz(2,a));
10        std::cout<<"x1="<<x1<<"x2="<<x2<<std::endl;
11    }
12    catch(pierwiastek_z_ujemnej& e){
13        std::cout<<"Brak rozwiązań"<<std::endl;
14    }
15    catch(wynik_spoza_zakresu){
16        std::cout<<"Wyniki działań nie mieścią się w typie"
17                           <<std::endl;
18    }
19    catch(spoza_dziedziny& e){
20        std::cout<<"Argumenty działania spoza dziedziny"
21                           <<std::endl;
22    }
23    catch(...){
24        std::cout<<"Nieznany błąd"<<std::endl;
25    }
}

```

Listing 8.118. rozwiązań zadania 7.7 wersja oparta na errno

```

void rownanie(double a, double b, double c){
2     errno = 0;
3     double delta = odejmij(pomnoz(b,b),
4                             pomnoz(4,pomnoz(a,c)));
5     if (delta>=0){
6         double pierw_delta = sqrt(delta);
7         double x1 = podziel(dodaj(odejmij(0,b),
8                               pierw_delta),pomnoz(2,a));
9         double x2 = podziel(odejmij(odejmij(0,b),
10                            pierw_delta),pomnoz(2,a));
11        if (errno==0){
12            std::cout<<"x1="<<x1;

```

```

14         if (delta>0)
15             std :: cout<<"_x2_=_"<<x2<<std :: endl;
16     }
17     if ((errno==0)&&(delta<0))
18         std :: cout<<"Brak rozwiazan "<<std :: endl;
19     if (errno==ERANGE)
20         std :: cout<<"Wyniki dzialan nie mieszczaja sie w typie "
21                         <<std :: endl;
22     else if (errno==EDOM)
23         std :: cout<<"Argumenty dzialan spoza dziedziny "
24                                         <<std :: endl;
25     else if (errno>0)
26         std :: cout<<"Nieznany blad "<<std :: endl;
27 }
```

Jednym z podstawowych kłopotów przy użyciu `errno`, jest fakt, że każda kolejna wykonywana operacja może nadpisać jego wartość. Oznacza to, że jeżeli dokładnie chcielibyśmy wyśledzić pierwotną przyczynę wystąpienia błędu, musielibyśmy sprawdzać wartość `errno` po wykonaniu każdej operacji. Użycie wyjątków pozbawione jest tej wady.

Zadanie 7.8

Listing 8.119. rozwizanie zadania 7.8

```

25         throw;
26     }
27     catch (...) {
28         std :: cout<<"Nieznany _blad "<<std :: endl;
29         throw;
30     }
31 }
```

Zadanie 7.17

Listing 8.120. rozwiązanie zadania 7.17

```

1 void bezpieczna() throw(){
2     std :: cout<<"jestem _bezpieczna "<<std :: endl;
3 }
```

Zadanie 7.18

Listing 8.121. rozwiązanie zadania 7.18

```

1 int podziel2(int a, int b) throw(dzielenie_przez_zero){
2     if (b==0)
3         throw dzielenie_przez_zero();
4     return a/b;
5 }
```

Zadania 7.19–7.20

Listing 8.122. rozwiązanie zadań 7.19–7.20

```

1 #include <iostream>
2 #include <cstdlib>
3 #include <vector>
4 #include "wyjatki.h"
5
6 int podziel(int a, int b) throw(dzielenie_przez_zero,
7             inny_blad){
8     if (b==0)
9         throw dzielenie_przez_zero();
10    return a/b;
11 }
```



```

13 void unexpected(){
14     std :: cout<<"nieznany _blad _przy _dzieleniu "
15                             <<std :: endl;
16     throw inny_blad();
17 }
```



```

19 void terminate(){
20     std :: cout<<"niezlapanego wyjatek "<<std :: endl;
```

```

21         exit(1);
}
23
24 int main() {
25     std::set_unexpected(unexpected);
26     std::set_terminate(terminate);
27     std::vector<std::pair<int, int>> v(10);

28     for(int i=0;i<10;i++)
29         std::cin>>v[i].first>>v[i].second;

30     for(int i=0;i<10;i++) try {
31         std::cout<<podziel(v[i].first,v[i].second)
32                         <<std::endl;
33     }
34     catch(...) {
35     }
36 }
37 }
```

Zakładamy, że dołączony w nagłówku programu plik `wyjatki.h` zawiera rozwiązanie zadania 7.22.

Zadanie 7.18

Listing 8.123. rozwiązanie zadania 7.22

```

#include <iostream>
2 #include <vector>
#include <csetjmp>
4
5     jmp_buf bufor;
6
7     int podziel(int a, int b){
8         if (b==0)
9             longjmp(bufor,1);
10        return a/b;
11    }
12 int main(){
13     std::vector<std::pair<int, int>> v(10);
14     for(int i=0;i<10;i++){
15         std::cin>>v[i].first>>v[i].second;
16     }
17     for(int i=0;i<10;i++) {
18         if (setjmp(bufor)!=0){
19             }
20         else
21             std::cout<<podziel(v[i].first,v[i].second)
22                             <<std::endl;
23     }
24 }
```

BIBLIOGRAFIA

- [1] Marshall Cline, *C++ FAQ — Frequently Asked Questions*, <http://www.parashift.com/c++-faq-lite/>
- [2] Bruce Eckel, *Thinking in C++*, Helion, Warszawa 2002
- [3] Jerzy Grębosz, *Symfonia C++ standard*, Editions, Kraków 2008
- [4] Bjarne Stroustrup, *Język C++*, Wydawnictwa Naukowo-Techniczne, Warszawa 2002
- [5] Standard C++03x, *ISO/IEC 14882:2003*, ISO 2003
- [6] Standard C++11x, *ISO/IEC 14882:2011*, ISO 011
- [7] *C++ reference*, <http://en.cppreference.com/w/cpp>

Zadania z programowania w języku C/C++, cz. I



KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓŁNOŚCI



UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY



Projekt „Programowa i strukturalna reforma systemu kształcenia na Wydziale Mat-Fiz-Inf”.
Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Człowiek-najlepsza inwestycja

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
WYDZIAŁ MATEMATYKI, FIZYKI I INFORMATYKI
INSTYTUT INFORMATYKI

Zadania z programowania w języku C/C++, cz. I

Jacek Krzaczkowski



LUBLIN 2011

**Instytut Informatyki UMCS
Lublin 2011**

Jacek Krzaczkowski
ZADANIA Z PROGRAMOWANIA W JĘZYKU C/C++, CZ. I

Recenzent: Grzegorz Matecki

Opracowanie techniczne: Marcin Denkowski
Projekt okładki: Agnieszka Kuśmierska

Praca współfinansowana ze środków Unii Europejskiej w ramach
Europejskiego Funduszu Społecznego

Publikacja bezpłatna dostępna on-line na stronach
Instytutu Informatyki UMCS: informatyka.umcs.lublin.pl.

Wydawca

Uniwersytet Marii Curie-Skłodowskiej w Lublinie
Instytut Informatyki
pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin
Redaktor serii: prof. dr hab. Paweł Mikołajczak
www: informatyka.umcs.lublin.pl
email: dyrii@hektor.umcs.lublin.pl

Druk

ESUS Agencja Reklamowo-Wydawnicza Tomasz Przybylak
ul. Ratajczaka 26/8
61-815 Poznań
www: wwwesus.pl

ISBN: 978-83-62773-06-0

SPIS TREŚCI

PRZEDMOWA	vii
1 PODSTAWY JĘZYKÓW C I C++, OPERACJE STERUJĄCE	1
1.1. Wprowadzenie	2
1.2. Podstawy języków C i C++, operacje wejścia wyjścia.	2
1.3. Instrukcje warunkowe i wyboru	3
1.4. Pętle	4
2 FUNKCJE	7
2.1. Wprowadzenie	8
2.2. Zadania	8
3 WSKAŹNIKI I REFERENCJE	13
3.1. Wprowadzenie	14
3.2. Zadania	14
4 TABLICE JEDNOWYMIAROWE	17
4.1. Wprowadzenie	18
4.2. Zadania	18
5 NAPISY	23
5.1. Wprowadzenie	24
5.2. Zadania	25
6 TABLICE WIELOWYMIAROWE	31
6.1. Wprowadzenie	32
6.2. Zadania	32
7 ZŁOŻONE TYPY DANYCH, LISTY WSKAŹNIKOWE	37
7.1. Wprowadzenie	38
7.2. Złożone typy danych	38
7.3. Listy jednokierunkowe	41
8 OPERACJE NA PLIKACH	47

8.1.	Wprowadzenie	48
8.2.	Zadania	48
9	INSTRUKCJE PREPROCESORA, APLIKACJE WIELOPLIKOWE, MAKEFILE.	51
9.1.	Wprowadzenie	52
9.2.	Makra	52
9.3.	Aplikacje wieloplikowe, makefile	53
10	ROZWIĄZANIA I WSKAZÓWKI	57
10.1.	Rozwiązania do zadań z rozdziału 1.2	58
10.2.	Rozwiązania do zadań z rozdziału 1.3	62
10.3.	Rozwiązania do zadań z rozdziału 1.4	65
10.4.	Rozwiązania do zadań z rozdziału 2.2	70
10.5.	Rozwiązania do zadań z rozdziału 3.2	77
10.6.	Rozwiązania do zadań z rozdziału 4.2	79
10.7.	Rozwiązańa do zadań z rozdziału 5.2	86
10.8.	Rozwiązańa do zadań z rozdziału 6.2	97
10.9.	Rozwiązańa do zadań z rozdziału 7.2	105
10.10.	Rozwiązańa do zadań z rozdziału 7.3	113
10.11.	Rozwiązańa do zadań z rozdziału 8.2	129
10.12.	Rozwiązańa do zadań z rozdziału 9.2	138
10.13.	Rozwiązańa do zadań z rozdziału 9.3	139
	BIBLIOGRAFIA	151

PRZEDMOWA

Książka ta jest adresowana przede wszystkim do czytelników uczących się języka C lub strukturalnego C++ jako swojego pierwszego języka programowania. Ponadto może być użyteczna dla programistów C i programistów C++ pragnących szybko nauczyć się podstaw drugiego z omawianych języków. Ponieważ niniejszy zbiór jest przeznaczony przede wszystkim dla początkujących programistów, wiele spośród zawartych w nim zadań ma służyć nie tyle sprawdzeniu znajomości języka C lub C++, co wyrobieniu umiejętności algorytmicznego myślenia i programowania w ogóle. Z tego samego powodu w zbiorze tym jest niewiele zadań sprawdzających znajomość funkcji bibliotecznych. Nie znaczy to, że w skrypcie zabrakło zadań trudniejszych, wymagających znajomości bardziej zaawansowanych możliwości języków C i C++.

Niniejszy zbiór zadań może być używany zarówno w ramach kursu na uczelni, jak i przez osoby uczące się programować samodzielnie. Skrypt ten został napisany przy założeniu, że czytelnik korzysta jednocześnie z książki lub materiałów zawierających systematyczny opis składni języka C lub C++. Przykłady takich książek i materiałów, także takich, które są bezpłatnie dostępne w internecie, czytelnik znajdzie w bibliografii.

Pisząc niniejszy zbiór zadań autor wykorzystał doświadczenia nabycie w ciągu kilku lat prowadzenia zajęć z przedmiotów „Programowanie w języku C” i „Programowanie w języku C++” na kierunku informatyka na Uniwersytecie Marii Curie-Skłodowskiej w Lublinie. W pierwszych dziewięciu rozdziałach skryptu znajdują się podzielone tematycznie zadania. Kolejność rozdziałów odpowiada kolejności, w jakiej zdaniem autora należy uczyć się strukturalnego programowania w językach C i C++. Czytelnik uczący się języka C od podstaw może ominąć rozdział 3 i wrócić do niego w trakcie czytania rozdziału 5.

Zadania w poszczególnych rozdziałach są zazwyczaj ułożone od najprostszych do najtrudniejszych. Gwiazdką zostały oznaczone zadania trudniejsze lub wymagające szczególowej znajomości języka C lub C++. Czytelnik, który dopiero zaczyna swoją przygodę z programowaniem, może pominąć w trakcie pierwszego czytania te zadania i wróć do nich później.

W ostatnim rozdziale czytelnik znajdzie rozwiązania wielu spośród zadań zawartych w niniejszym zbiorze zadań. Rozwiązania te są integralną częścią skryptu. Dołączone do rozwiązań komentarze mają na celu m.in. zwrócenie uwagi czytelnika na ciekawy sposób rozwiązania zadania, pojawiające się w trakcie rozwiązywania zadania problemy czy też błędy często popełniane przez niedoświadczonych programistów. W przypadkach gdy ma to wartość dydaktyczną przedstawiono kilka wariantów rozwiązań poszczególnych zadań.

W rozdziale z rozwiązaniami prezentowane są zarówno rozwiązania w języku C, jak i w C++. W wypadku gdy rozwiązania danego zadania w obu językach są takie same lub podobne, prezentowane jest tylko jedno rozwiązanie. Dla zadań, dla których wzorcowe rozwiązania w językach C i C++ istotnie się różnią, prezentowane są rozwiązania w obu językach. Często jednak nawet w takich sytuacjach rozwiązanie w języku C jest równocześnie poprawnym rozwiązaniem w C++.

Wszystkie rozwiązania zawarte w tym skrypcie były komplilowane przy pomocy GNU Compiler Collection w wersji 4.4.1 na komputerze z zainstalowanym systemem operacyjnym OpenSUSE 11.2. O ile nie podano inaczej, przykładowe programy napisane w C były komplilowane przy pomocy polecenia `gcc <nazwa programu>`. Przykładowe programy w C++ były komplilowane przy pomocy polecenia `g++ <nazwa programu>`.

Aby ułatwić posługiwanie się zbiorem zadań wprowadzono następujące oznaczenia:

- * trudne zadanie,
- r zadanie rozwiążane w ostatnim rozdziale,
- ! zadanie, którego rozwiązanie z różnych powodów jest szczególnie interesujące,
- C zadanie do rozwiązania wyłącznie w języku C,
- C++ zadanie do rozwiązania wyłącznie w języku C++,
- róż zadanie ilustrujące różnicę pomiędzy językiem C a językiem C++.

ROZDZIAŁ 1

PODSTAWY JĘZYKÓW C I C++, OPERACJE STERUJĄCE

1.1.	Wprowadzenie	2
1.2.	Podstawy języków C i C++, operacje wejścia wyjścia.	2
1.3.	Instrukcje warunkowe i wyboru	3
1.4.	Pętle	4

1.1. Wprowadzenie

W tym rozdziale czytelnik znajdzie zadania pozwalające przećwiczyć najbardziej podstawowe elementy języków C i C++. W podrozdziale 1.2 znajdują się zadania sprawdzające podstawową wiedzę na temat pisania programów w językach C i C++ oraz podstawy operacji wejścia/wyjścia w tych językach. Podrozdziały 1.3 i 1.4 zawierają zadania pozwalające przećwiczyć używanie instrukcji sterujących. Podrozdziały te są przeznaczone szczególnie dla tych, którzy uczą się swojego pierwszego imperatywnego języka programowania. W rozwiązaniach zadań celowo nie użyto instrukcji takich jak `goto`, `break` (za wyjątkiem wnętrza instrukcji `switch`) czy `continue`, które są uważane za niezgodne z zasadami programowania strukturalnego.

1.2. Podstawy języków C i C++, operacje wejścia wyjścia.

- 1.2.1 **(r)** Napisz program wypisujący na standardowym wyjściu napis „Hello World”.
- 1.2.2 Napisz program wypisujący w kolejnych wierszach standardowego wyjścia pojedyncze słowa następującego napisu „Bardzo dlugi napis”.
- 1.2.3 Napisz program wypisujący na standardowym wyjściu następujący napis: „Napis zawierajacy rozne dziwne znaczki // \ \\$ & %”.
- 1.2.4 **(r,!)** Napisz program, który wczytuje ze standardowego wejścia liczbę całkowitą i wypisuje ją na standardowym wyjściu.
- 1.2.5 **(r,!)** Napisz program, który wczytuje ze standardowego wejścia liczbę wymierną i wypisuje ją na standardowym wyjściu
- 1.2.6 Napisz program, który wczytuje ze standardowego wejścia trzy liczby całkowite, a następnie wypisuje je w oddzielnych liniach na standardowym wyjściu.
- 1.2.7 Napisz program, który wczytuje ze standardowego wejścia liczbę całkowitą i wypisuje na standardowym wyjściu liczbę o jeden większą.
- 1.2.8 **(r,!)** Napisz program, który wczytuje ze standardowego wejścia trzy liczby całkowite i wypisuje na standardowym wyjściu ich średnią arytmetyczną.
- 1.2.9 **(r,!rów)** Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę wymierną x i wypisuje na standardowym wyjściu \sqrt{x} .
- 1.2.10 Napisz program, który wczytuje ze standardowego wejścia liczbę wymierną x i wypisuje na standardowym wyjściu wartość bezwzględną z x .

- 1.2.11 (r) Napisz program, który wczytuje ze standardowego wejścia liczbę wymierną i wypisuje ją na standardowym wyjściu z dokładnością do dwóch miejsc po przecinku.
- 1.2.12 Napisz program, który wczytuje ze standardowego wejścia liczbę wymierną i wypisuje ją na standardowym wyjściu w notacji wykładniczej (czyli takiej, w której 0.2 to $2.0\text{e-}1$).

1.3. Instrukcje warunkowe i wyboru

- 1.3.1 (r) Napisz program, który wczytuje ze standardowego wejścia liczbę całkowitą n i wypisuje na standardowe wyjście wartość bezwzględną z n .
Do rozwiązania zadania nie używaj funkcji bibliotecznych za wyjątkiem operacji wejścia/wyjścia.
- 1.3.2 Napisz program, który wczytuje ze standardowego wejścia dwie liczby całkowite i wypisuje na standardowym wyjściu większą z nich (w przypadku gdy podane liczby są równe, program powinien wypisać każdą z nich).
- 1.3.3 Napisz program, który wczytuje ze standardowego wejścia trzy liczby całkowite i wypisuje na standardowym wyjściu największą z ich wartości (pamiętaj o przypadku gdy wszystkie podane liczby lub dwie z nich są równe).
- 1.3.4 Napisz program, który wczytuje ze standardowego wejścia dwie liczby całkowite i wypisuje tą o większej wartości bezwzględnej.
- 1.3.5 (r) Napisz program obliczający pole trójkąta na podstawie wymiarów podanych przez użytkownika. Użytkownik powinien mieć możliwość podania długości podstawy i wysokości lub wszystkich boków trójkąta.
- 1.3.6 Napisz program, który wczytuje ze standardowego wejścia współczynniki układu dwóch równań liniowych z dwoma niewiadomymi i wypisuje na standardowym wyjściu rozwiązanie układu równań. W przypadku nieskończonej liczby lub braku rozwiązań program powinien wypisać na standardowym wyjściu odpowiednią informację.

Podpowiedź: zaimplementuj algorytm rozwiązywania układów równań metodą wyznaczników (inaczej nazywaną wzorami Cramera).

- 1.3.7 Napisz program, który wczytuje ze standardowego wejścia współczynniki równania kwadratowego z jedną niewiadomą i wypisuje na standardowym wyjściu wszystkie rozwiązania rzeczywiste tego równania lub odpowiednią informację w przypadku braku rozwiązań.
- 1.3.8 (r) Napisz program, który w zależności od wyboru użytkownika wczytuje ze standardowego wejścia wymiary: kwadratu, prostokąta lub trójkąta.

kąta i wypisuje na standardowym wyjściu pole figury o wczytanych wymiarach.

- 1.3.9 Napisz program kalkulator, który wykonuje wybraną przez użytkownika operację arytmetyczną na dwóch wczytanych liczbach. Program powinien wczytywać dane ze standardowego wejścia i wypisywać wynik na standardowym wyjściu.

1.4. Pętle

- 1.4.1 (**r,!**) Napisz program wczytujący ze standardowego wejścia dwie dodatnie liczby całkowite n i m , i wypisujący w kolejnych wierszach na standardowym wyjściu wszystkie dodatnie wielokrotności n mniejsze od m .
- 1.4.2 Napisz program wczytujący ze standardowego wejścia dwie dodatnie liczby całkowite n i m , i wypisujący na standardowym wyjściu m pierwszych wielokrotności liczby n .
- 1.4.3 Napisz program wczytujący ze standardowego wejścia trzy dodatnie liczby całkowite n , m i k , i wypisujący w kolejnych wierszach wszystkie wielokrotności n większe od m i mniejsze od k .
- 1.4.4 (**r**) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu liczbę $n!$.
- 1.4.5 Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu sumę kwadratów liczb od 0 do n , czyli wartość $0^2 + 1^2 + 3^2 + \dots + n^2$.
- 1.4.6 Napisz program, który wczytuje ze standardowego wejścia liczbę całkowitą n ($n > 2$) i wypisuje na standardowym wyjściu iloczyn liczb parzystych z zakresu od 2 do n (czyli $2 * 4 * \dots * n$ dla n parzystych i $2 * 4 * \dots * (n - 1)$ w przeciwnym wypadku).
- 1.4.7 Napisz program, który wczytuje ze standardowego wejścia dwie liczby całkowite n i m (zakładamy, że $n < m$) i wypisuje na standardowym wyjściu liczbę $n * \dots * m$.
- 1.4.8 (*,r) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu element ciągu Fibonacciego o indeksie n .
- 1.4.9 (**r,!**) Napisz program, który wczytuje ze standardowego wejścia dodatnie liczby całkowite n i m , i wypisuje na standardowym wyjściu największy wspólny dzielnik tych liczb.
- 1.4.10 (**r,!**) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę n i wypisuje na standardowym wyjściu wartość $\lfloor \sqrt{n} \rfloor$

(wartość \sqrt{n} zaokrągloną w dół do najbliższej wartości całkowitoliczbowej). Program napisz bez użycia funkcji z biblioteki matematycznej.

- 1.4.11 Napisz program, który wczytuje ze standardowego wejścia liczby a , b , c , d i:
- wypisuje na standardowe wyjście najmniejszą nieujemną liczbę całkowitą x taką, że $|a| * x^2 + b * x + c > d$.
 - wypisuje na standardowe wyjście największą nieujemną liczbę całkowitą x taką, że $5*x^2+a*x+b < c$. Zakładamy, że taka nieujemna całkowita liczba x istnieje.
 - wypisuje na standardowe wyjście największą nieujemną liczbę całkowitą x taką, że $5*x^2+a*x+b \leq c$. Zakładamy, że taka nieujemna całkowita liczba x istnieje.
- 1.4.12 (*,r) Napisz program, który wczytuje ze standardowego wejścia dodatnią liczbę n i wypisuje na standardowym wyjściu sumę wszystkich liczb mniejszych od n , względnie pierwszych z n .
- 1.4.13 (*, r, !) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu wartość $0! + 1! + \dots + n!$.
- 1.4.14 (*) Napisz program, który wczytuje ze standardowego wejścia liczbę n i wypisuje na standardowym wyjściu wszystkie trójkę pitagorejskie (tj. trójkę liczb całkowitych a , b , c takich, że $a^2 + b^2 = c^2$), składające się z liczb mniejszych od n .

ROZDZIAŁ 2

FUNKCJE

2.1.	Wprowadzenie	8
2.2.	Zadania	8

2.1. Wprowadzenie

Funkcje są ważnym elementem większości języków imperatywnych. W niniejszym rozdziale znajdują się zadania pozwalające przećwiczyć różne aspekty pracy z funkcjami, od pisania prostych funkcji po przeciążanie funkcji i pisanie funkcji z domyślnymi wartościami argumentów (dwie ostatnie możliwości udostępnia nam tylko język C++). Czytelnik znajdzie w tym rozdziale także grupę zadań, w których należy napisać funkcję rekurencyjną. Pisanie funkcji rekurencyjnych wymaga szczególnej ostrożności, gdyż w ich przypadku szukanie błędów jest wyjątkowo trudne. Pomimo tego warto przećwiczyć pisanie funkcji rekurencyjnych, ponieważ w wielu przypadkach ich użycie pozwala znacznie uprościć kod programu.

2.2. Zadania

- 2.2.1 (r) Napisz program, który wczytuje ze standardowego wejścia liczbę całkowitą n i wypisuje na standardowe wyjście wartość bezwzględną z n . Do rozwiązania zadania nie używaj funkcji bibliotecznych za wyjątkiem operacji wejścia/wyjścia. W programie użyj samodzielnie zaimplementowanej funkcji liczącej wartość bezwzględną.
- 2.2.2 (r) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu liczbę $n!$. W programie użyj samodzielnie zaimplementowanej funkcji liczącej wartość silni.
- 2.2.3 Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n ($n > 2$) i wypisuje na standardowym wyjściu największą liczbę k taką, że k dzieli n i $k < n$. Algorytm wyszukiwania liczby k spełniającej powyższe warunki umieść w oddzielnej funkcji.
- 2.2.4 Napisz funkcję, która dostaje jako argument nieujemną liczbę całkowitą n i zwraca jako wartość liczbę 2^n .
- 2.2.5 Napisz funkcję, która dostaje jako argument liczbę całkowitą n i zwraca jako wartość liczbę 2^n .
- 2.2.6 Napisz funkcję, która dostaje jako argumenty nieujemne liczby całkowite n i m , z których co najmniej jedna jest różna od zera i zwraca jako wartość n^m .
- 2.2.7 Napisz funkcję, która dostaje jako argumenty liczby całkowite n i m , z których co najmniej jedna jest różna od zera i zwraca jako wartość n^m .
- 2.2.8 Napisz funkcję, która dostaje jako argumenty liczbę dodatnią n i zwraca jako wartość $\lfloor \sqrt{n} \rfloor$. Rozwiąż zadanie nie wykorzystując funkcji bibliotecznych.

- 2.2.9 (*) Napisz funkcję, która dostaje jako argumenty liczbę całkowitą m ($m > 1$) oraz nieujemną liczbę n i zwraca jako wartość $\lfloor \sqrt[m]{n} \rfloor$. Rozwiąż zadanie nie wykorzystując funkcji bibliotecznych.
- 2.2.10 (r) Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu sumę liczb mniejszych od n i zarazem względnie pierwszych z n . Algorytm wyliczania sumy podziel na dwie funkcje.
- 2.2.11 Napisz program, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowym wyjściu następującą sumę $\lfloor \sqrt{0} \rfloor + \lfloor \sqrt{1} \rfloor + \dots + \lfloor \sqrt{n} \rfloor$. Algorytm wyliczania sumy podziel na dwie funkcje.
- 2.2.12 Napisz program, który wczytuje ze standardowego wejścia nieujemne liczby całkowitą n i m ($m > 1$), i wypisuje na standardowym wyjściu następującą sumę $\lfloor \sqrt[m]{0} \rfloor + \lfloor \sqrt[m]{1} \rfloor + \dots + \lfloor \sqrt[m]{n} \rfloor$. Algorytm wyliczania sumy podziel na dwie funkcje.
- 2.2.13 (*,r) Napisz funkcję, która dostaje jako argument dodatnią liczbę całkowitą n i wypisuje na standardowym wyjściu wszystkie możliwe rozkłady liczby n na sumy dwóch kwadratów dodatnich liczb całkowitych. Rozważ dwa przypadki:
- (a) gdy „ $a^2 + b^2$ ” i „ $b^2 + a^2$ ” dla $a \neq b$ traktujemy jako dwa różne rozkłady,
 - (b) gdy „ $a^2 + b^2$ ” i „ $b^2 + a^2$ ” traktujemy jako ten sam rozkład i wypisujemy tylko jedne z nich.
- Jeżeli zajdzie taka potrzeba, możesz w rozwiązaniu używać funkcji pomocniczych.
- 2.2.14 (*) Napisz funkcję, która dostaje jako argument dodatnią liczbę całkowitą n i wypisuje na standardowym wyjściu wszystkie możliwe rozkłady liczby n na sumy kwadratów dodatnich liczb całkowitych. Rozważ dwa przypadki analogiczne do tych z zadania 2.2.13. Jeżeli zajdzie taka potrzeba możesz w rozwiązaniu używać funkcji pomocniczych.
- 2.2.15 (*) Napisz funkcję, która dostaje jako argumenty dodatnie liczby całkowite n i m , i wypisuje na standardowym wyjściu wszystkie możliwe rozkłady liczby n na sumę dwóch dodatnich liczb całkowitych podniesionych do potęgi m . Rozważ dwa przypadki analogiczne do tych z zadania 2.2.13. Jeżeli zajdzie taka potrzeba możesz w rozwiązaniu używać funkcji pomocniczych.
- 2.2.16 (*) Napisz funkcję, która dostaje jako argumenty dodatnie liczby całkowite n i m , i wypisuje na standardowym wyjściu wszystkie możliwe rozkłady liczby n na sumę dodatnich liczb całkowitych podniesionych do potęgi m . Rozważ dwa przypadki analogiczne do tych z zadania 2.2.13. Jeżeli zajdzie taka potrzeba możesz w rozwiązaniu używać funkcji pomocniczych.

- 2.2.17 (*,r,!) Napisz funkcję, która zlicza i wypisuje na standardowym wyjściu liczbę swoich wywołań.
- 2.2.18 (*) Napisz funkcję generującą liczby pseudolosowe. Pierwszą wartością funkcji powinna być dowolna liczba z przedziału $(0, 1)$. Kolejne wartości powinny być wyliczane ze wzoru $x_n = 1 - x_{n-1}^2$, gdzie x_n to aktualna, a x_{n-1} to poprzednia wartość funkcji.
- 2.2.19 (*) Napisz funkcję, która wczytuje ze standardowego wejścia liczbę całkowitą i zwraca ją jako swoją wartość. Dodatkowo funkcja powinna sumować wszystkie dotychczas wczytane wartości i przy każdym swoim wywołaniu wypisywać na standardowym wyjściu ich aktualną sumę .
- 2.2.20 (r,!) Napisz rekurencyjną funkcję, która dla otrzymanej w argumencie nieujemnej całkowitej liczby n zwraca jako wartość $n!$.
- 2.2.21 Napisz rekurencyjną funkcję zwracającą dla otrzymanej w argumencie nieujemnej liczby całkowitej n wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób:

$$a_0 = 1$$

$$a_n = 2 * a_{n-1} + 5 \text{ dla } n > 0.$$

- 2.2.22 Napisz rekurencyjną funkcję zwracającą dla otrzymanej w argumencie nieujemnej liczby całkowitej n wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób:

$$a_0 = a_1 = 1$$

$$a_n = a_{n-1} + 2 * a_{n-2} + 3 \text{ dla } n > 1$$

- 2.2.23 (r,!) Napisz rekurencyjną funkcję zwracającą dla otrzymanej w argumencie nieujemnej liczby całkowitej n wartość elementu ciągu Fibonacciego o indeksie n .
- 2.2.24 (*) Napisz rekurencyjną funkcję zwracającą dla otrzymanej w argumencie nieujemnej liczby całkowitej n wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób

$$a_0 = a_1 = 1$$

$$a_n = a_0 + a_1 + \dots + a_{n-1} \text{ dla } n > 1$$

- 2.2.25 (*) Napisz funkcję rekurencyjną, która dla otrzymanej w argumencie nieujemnej liczby całkowitej n zwraca wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób

$$a_0 = a_1 = 1$$

$$a_n = a_{n-1} + n \text{ dla } n \text{ parzystych}$$

$$a_n = a_{n-1} * n \text{ dla } n \text{ nieparzystych.}$$

- 2.2.26 (*,r,!) Napisz funkcję rekurencyjną, która dla otrzymanej w argumentie nieujemnej liczby całkowitej n zwraca wartość elementu o indeksie n ciągu zdefiniowanego w następujący sposób

$$a_0 = a_1 = a_2 = 1$$

oraz dla $k > 2$

$$a_{3 \cdot k} = a_{3 \cdot k - 1} + a_{3 \cdot k - 2}$$

$$a_{3 \cdot k + 1} = 5 * a_{3 \cdot k} + 4$$

$$a_{3 \cdot k + 2} = a_{3 \cdot k + 1}.$$

- 2.2.27 (r) Napisz funkcję rekurencyjną, która dla otrzymanej w argumentach pary nieujemnych liczb całkowitych n i m zwraca wartość $f(n, m)$ gdzie funkcja f jest zdefiniowana w następujący sposób:

$$f(n, 0) = n$$

$$f(0, m) = m$$

$$f(n, m) = f(n - 1, m) + f(n, m - 1) + f(n - 1, m - 1) \text{ dla } n, m > 0.$$

- 2.2.28 Napisz funkcję rekurencyjną, która dla otrzymanej w argumentach pary nieujemnych liczb całkowitych n i m zwraca wartość $f(n, m)$ gdzie funkcja f jest zdefiniowana w następujący sposób:

$$f(n, 0) = n$$

$$f(n, m) = f(m, n)$$

$$f(n, m) = n - m + f(n - 1, m) + f(n, m - 1) \text{ dla } n \geq m > 0.$$

- 2.2.29 (r,!) Napisz rekurencyjną funkcję, która dostaje jako argumenty dwie dodatnie liczby całkowite n i m , i zwraca jako wartość największy wspólny dzielnik tych liczb obliczony algorytmem Euklidesa.

- 2.2.30 (C++,r,!) Napisz funkcję, która dostaje jako argumenty nieujemne liczby całkowite n i m , z których co najmniej jedna jest różna od zera, i zwraca jako wartość n^m . Jeżeli drugi z argumentów nie zostanie podany, funkcja powinna zwrócić wartość n^2 .

- 2.2.31 (C++) Napisz funkcję, która dostaje jako argumenty liczbę całkowitą m ($m > 1$) oraz nieujemną liczbę n i zwraca jako wartość $\lfloor \sqrt[m]{n} \rfloor$. Rozwiąż to zadanie nie wykorzystując funkcji bibliotecznych. W przypadku podania tylko pierwszego argumentu funkcja powinna zwracać $\lfloor \sqrt{n} \rfloor$.

- 2.2.32 (C++) Napisz funkcję, która dostaje jako argumenty pięć liczb całkowitych typu `unsigned int` i zwraca jako wartość maksimum z podanych liczb. Funkcję napisz w taki sposób, żeby można było jej podać także mniejszą liczbę argumentów.
- 2.2.33 (C++) Napisz funkcję, która dostaje jako argumenty pięć liczb typu `unsigned int` i zwraca jako wartość sumę podanych liczb. Funkcję napisz w taki sposób, żeby liczyła sumę także dwóch, trzech i czterech argumentów.
- 2.2.34 (C++) Napisz funkcję, która dostaje jako argumenty pięć liczb typu `int` i zwraca jako wartość iloczyn podanych liczb. Funkcję napisz w taki sposób, żeby liczyła iloczyn także dwóch, trzech i czterech argumentów.
- 2.2.35 (C++) Napisz funkcję, która dla dwóch dodatnich całkowitoliczbowych argumentów m i n zwraca wartość `true` jeżeli n dzieli m oraz `false` w przeciwnym wypadku. W przypadku podania jednego argumentu funkcja powinna sprawdzać czy podana liczba jest parzysta.
- 2.2.36 (C++,r,!) Napisz rodzinę dwuargumentowych funkcji `pot`, z których każda jako argumenty otrzymuje liczbę n i nieujemną liczbę całkowitą m typu `unsigned int` (zakładamy, że co najmniej jeden z argumentów jest różny od zera) i zwraca jako wartość n^m . Przeciąż funkcję `pot` dla n o typach: `double`, `int`, `unsigned int`. Wynik zwrócony przez każdą z funkcji `pot` powinien być tego samego typu co n .
- 2.2.37 (C++) Rodzinę funkcji `pot` z zadania 2.2.36 rozszerz o funkcje, w których m jest typu `int`. Funkcje te powinny zwracać wartości typu `double`.
- 2.2.38 (C++) Napisz rodzinę funkcji `maksimum` zwracających maksimum z dwóch liczb otrzymanych w argumentach. Typem zwracanym powinien być bardziej pojemny z typów argumentów (tak jak to ma miejsce w przypadku operacji arytmetycznych). Przykładowo dla jednego argumentu typu `int` a drugiego typu `double` zwrócony powinien zostać wynik o typie `doble`.
- 2.2.39 (C++) Napisz rodzinę funkcji `maksimum` zwracających maksimum od dwóch do pięciu wartości otrzymanych w argumentach. Argumenty funkcji oraz wartość zwracana przez funkcję powinny być typu `int`.
- 2.2.40 (C++) Napisz rodzinę funkcji `srednia` zwracających średnią arytmetyczną z dwóch lub trzech wartości podanych przez użytkownika. Typem wyniku każdej z funkcji powinien być najbardziej pojemny z typów argumentów.

ROZDZIAŁ 3

WSKAŹNIKI I REFERENCJE

3.1.	Wprowadzenie	14
3.2.	Zadania	14

3.1. Wprowadzenie

Autor niniejszego zbioru jest zwolennikiem tego, aby w procesie nauczania języka C jako pierwszego języka programowania, najpierw mówić o tablicach jednowymiarowych, a dopiero potem mówić o wskaźnikach i ich powiązaniach z tablicami. Od lat w takiej właśnie kolejności był prezentowany materiał na kursie „Programowanie w języku C” na kierunku informatyka na UMCS. Takie ułożenie materiału pozwala studentom oswoić się z podstawami programowania zanim zaczną poznawać trudne zagadnienia związane ze wskaźnikami. Niniejszy zbiór zadań odszedł od takiego ułożenia materiału ze względu na jednoczesną prezentację rozwiązań w językach C i C++. O ile bowiem w języku C możemy używać jednowymiarowych tablic nie wiedząc nic o wskaźnikach ani dynamicznym zarządzaniu pamięcią, o tyle w języku C++ jest to możliwe tylko wtedy, gdy używamy tablic o z góry (t.j. w momencie komplikacji) znanych rozmiarach.

Ci spośród czytelników, którzy uczą się języka C i nie chcą jeszcze poznawać wskaźników mogą przejść od razu do rozdziału 4. Większość z prezentowanych tam zadań nie będzie wymagała od nich znajomości wskaźników.

3.2. Zadania

- 3.2.1 (r) Napisz funkcję otrzymującą jako argumenty wskaźniki do dwóch zmiennych typu `int`, która zwraca jako wartość mniejszą z liczb wskazywanych przez argumenty.
- 3.2.2 (r) Napisz funkcję otrzymującą jako argumenty wskaźniki do dwóch zmiennych typu `int`, która zwraca jako wartość wskaźnik na zmienną przechowującą mniejszą z liczb wskazywanych przez argumenty.
- 3.2.3 (r) Napisz funkcję otrzymującą jako argumenty wskaźniki do dwóch zmiennych typu `int`, która zamienia ze sobą wartości wskazywanych zmiennych.
- 3.2.4 Napisz funkcję otrzymującą jako argumenty wskaźniki do dwóch zmiennych typu `int`, która zamienia ze sobą wartości wskazywanych zmiennych tylko wtedy, gdy wskazywana przez drugi argument zmienna jest mniejsza od zmiennej wskazywanej przez pierwszy argument.
- 3.2.5 Napisz funkcję, której argumentami są dwa wskaźniki do stałych typu `int`, zaś zwracaną wartością jest suma wartości zmiennych wskazywanych przez argumenty.
- 3.2.6 Napisz funkcję, której argumentami są `n` typu `int` oraz `w` wskaźnik do `int`, która przepisuje wartość `n` do zmiennej wskazywanej przez `w`.

- 3.2.7 (**C++,r**) Napisz funkcję otrzymującą jako argumenty referencje do dwóch zmiennych typu **int**, która zamienia ze sobą wartości zmiennych, do których referencje dostaliśmy w argumentach.
- 3.2.8 (**C++**) Napisz funkcję otrzymującą dwa argumenty referencję **a** oraz wskaźnik **b** do zmiennych typu **int**, która zamienia ze sobą wartości zmiennych, do których wskaźnik i referencję dostała w argumentach.
- 3.2.9 (**r,róż**) Napisz bezargumentową funkcję, która rezerwuje pamięć dla pojedynczej zmiennej typu **int** i zwraca jako wartość wskaźnik do niej.
- 3.2.10 Napisz bezargumentową funkcję, która rezerwuje pamięć dla pojedynczej zmiennej typu **double** i zwraca jako wartość wskaźnik do niej.
- 3.2.11 (**r,róż**) Napisz funkcję, która dostaje jako argument dodatnią liczbę całkowitą **n**, rezerwuje w pamięci blok **n** zmiennych typu **int** i zwraca jako wartość wskaźnik do początku zarezerwowanego bloku pamięci.
- 3.2.12 Napisz funkcję, która dostaje jako argument dodatnią liczbę całkowitą **n**, rezerwuje w pamięci blok **n** zmiennych typu **double** i zwraca jako wartość wskaźnik do początku zarezerwowanego bloku pamięci.
- 3.2.13 (*,r,!) Napisz funkcję o dwóch argumentach:
- wskaźnik na funkcję o jednym argumencie typu **int** zwracającą wartość typu **double**,
 - wartość typu **int**,
- która zwraca wartość funkcji otrzymanej w pierwszym argumencie na liczbie całkowitej podanej w drugim argumencie.
- 3.2.14 (*) Napisz funkcję, która otrzymuje trzy argumenty:
- dwa wskaźniki na funkcje o jednym argumencie typu **int** zwracające wartość typu **int**,
 - wartość **n** typu **unsigned int**,
- i zwraca **true**, jeżeli otrzymane w argumentach funkcje są równe dla wartości od 0 do **n** i **false** w przeciwnym wypadku.
- 3.2.15 (**r,!**) Napisz funkcję, która dostaje dwa argumenty: wskaźnik na stałą typu **int** i wskaźnik na zmienną typu **int**, i przepisuje zawartość stałej wskazywanej przez pierwszy argument do zmiennej wskazywanej przez drugi argument.
- 3.2.16 (**r,!**) Napisz funkcję, która dostaje dwa argumenty: wskaźnik na stałą typu **int** i stały wskaźnik na zmienną typu **int**. I przepisuje zawartość stałej wskazywanej przez pierwszy argument do zmiennej wskazywanej przez drugi argument.

ROZDZIAŁ 4

TABLICE JEDNOWYMIAROWE

4.1.	Wprowadzenie	18
4.2.	Zadania	18

4.1. Wprowadzenie

Większość zadań prezentowanych w tym rozdziale nie wymaga od czytelnika uczącego się języka C znajomości wskaźników. Wystarczy umiejętność deklarowania tablic automatycznych. Wszystkie zadania wymagające operowania na wskaźnikach zostały oznaczone gwiazdką.

W języku C++ nie można deklarować tablic automatycznych o rozmiarze podanym przez zmienną. Jedynym sposobem w języku C++ tworzenia tablic o rozmiarze nieznanym w momencie komilacji jest ręczne tworzenie tablic dynamicznych (na przykład za pomocą operatora `new`).

4.2. Zadania

- 4.2.1 Napisz funkcję, która otrzymuje dwa argumenty: nieujemną liczbę całkowitą n oraz n -elementową tablicę `tab` elementów typu `int` i:
- (r) nadaje wartość zero wszystkim elementom tablicy `tab`,
 - (r) zapisuje do kolejnych elementów tablicy wartości równe ich indeksom (do komórki o indeksie i funkcja ma zapisywać wartość i),
 - podwaja wartość wszystkich elementów w tablicy `tab`,
 - do wszystkich komórek tablicy `tab` wstawia wartości bezwzględne ich pierwotnych wartości.
- 4.2.2 Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `int` i zwraca jako wartość:
- (r) średnią arytmetyczną elementów tablicy `tab`.
 - sumę elementów tablicy `tab`,
 - sumę kwadratów elementów tablicy `tab`.
- 4.2.3 (r) Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `const int` i zwraca jako wartość średnią arytmetyczną elementów tablicy `tab`.
- 4.2.4 (*) Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `unsigned int` i zwraca jako wartość średnią geometryczną elementów tablicy `tab`.
- 4.2.5 (*,r!,róż) Napisz funkcję, która otrzymuje jako argument liczbę całkowitą n ($n \geq 3$) i zwraca jako wartość największą liczbę pierwszą mniejszą od n (do wyznaczenia wyniku użyj algorytmu sita Eratostenesa).

- 4.2.6 Napisz funkcję, która otrzymuje trzy argumenty: dodatnią liczbę całkowitą n oraz dwie n -elementowe tablice `tab1`, `tab2` o elementach typu `int` i:
- (r) przepisuje zawartość tablicy `tab1` do tablicy `tab2`,
 - przepisuje zawartość tablicy `tab1` do tablicy `tab2` w odwrotnej kolejności (czyli element `tab1[0]` ma zostać zapisany do komórki tablicy `tab2` o indeksie $n - 1$).
- 4.2.7 Napisz funkcję, która otrzymuje cztery argumenty: dodatnią liczbę całkowitą n oraz trzy n -elementowe tablice `tab1`, `tab2` i `tab3` o elementach typu `int`, i:
- przypisuje elementom tablicy `tab3` sumę odpowiadających im elementów tablic `tab1` i `tab2` (do komórki tablicy `tab3` o indeksie i powinna trafić suma elementów `tab1[i]` i `tab2[i]`),
 - przypisuje elementom tablicy `tab3` większy spośród odpowiadających im elementów tablic `tab1` i `tab2` (do komórki tablicy `tab3` o indeksie i powinien trafić większy spośród elementów `tab1[i]` i `tab2[i]`),
 - przypisuje zawartość tablicy `tab1` do tablicy `tab2`, zawartość tablicy `tab2` do tablicy `tab3` oraz zawartość tablicy `tab3` do tablicy `tab1`.
- 4.2.8 Napisz funkcję, która otrzymuje cztery argumenty: dodatnią liczbę całkowitą n , n -elementowe tablice `tab1` i `tab2` oraz $2 \cdot n$ -elementową tablicę `tab3` o elementach typu `double`.
- Funkcja powinna przepisywać zawartość tablic `tab1` i `tab2` do tablicy `tab3` w taki sposób, że na początku tablicy `tab3` powinny się znaleźć elementy tablicy `tab1`, a po nich elementy tablicy `tab2`.
 - Funkcja powinna przepisywać zawartość tablic `tab1` i `tab2` do tablicy `tab3` w taki sposób, że w komórkach tablicy `tab3` o nieparzystych indeksach powinny się znaleźć elementy tablicy `tab1`, a w komórkach tablicy `tab3` o parzystych indeksach elementy tablicy `tab2`.
- 4.2.9 Napisz funkcję, która otrzymuje cztery argumenty: dodatnią liczbę całkowitą n oraz trzy n -elementowe tablice `tab1`, `tab2` i `tab3` o elementach typu `int` i zamienia zawartości komórek otrzymanych w argumentach tablic w następujący sposób:
- dla dowolnego i komórka `tab1[i]` powinna zawierać największą spośród pierwotnych wartości komórek `tab1[i]`, `tab2[i]` oraz `tab3[i]`,
 - dla dowolnego i komórka `tab2[i]` powinna zawierać drugą co do wielkości spośród pierwotnych wartości komórek `tab1[i]`, `tab2[i]` oraz `tab3[i]`,
 - dla dowolnego i komórka `tab3[i]` powinna zawierać najmniejszą

spośród pierwotnych wartości komórek `tab1[i]`, `tab2[i]` oraz `tab3[i]`.

- 4.2.10 Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `int` i:
- a) `(r,!)` zwraca największą wartość przechowywaną w tablicy `tab`,
 - b) zwraca najmniejszą wartość przechowywaną w tablicy `tab`,
 - c) `(r,!)` zwraca indeks elementu tablicy `tab` o największej wartości,
 - d) zwraca indeks elementu tablicy `tab` o najmniejszej wartości,
 - e) zwraca największą spośród wartości bezwzględnych elementów przechowywanych w tablicy `tab`,
 - f) zwraca indeks elementu tablicy `tab` o największej wartości bezwzględnej.
- 4.2.11 Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz dwie n -elementowe tablice `tab` o elementach typu `double` przechowujące n -wymiarowe wektory i zwraca jako wartość iloczyn skalarny wektorów otrzymanych w argumentach.
- 4.2.12 Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `int` i:
- a) `(r)` odwraca kolejność elementów tablicy `tab`.
 - b) `(r)` przesuwa o jeden w lewo wszystkie elementy tablicy (tak, żeby wartość elementu o indeksie $n - 1$ znalazła się w elemencie o indeksie $n - 2$, wartość elementu o indeksie $n - 2$ znalazła się w elemencie o indeksie $n - 3$, zaś wartość elementu o indeksie 0 w elemencie o indeksie $n - 1$),
 - c) `(r,!)` przesuwa o jeden w prawo wszystkie elementy tablicy (tak, żeby wartość elementu o indeksie 0 znalazła się w elemencie o indeksie 1, wartość elementu o indeksie 1 znalazła się w elemencie o indeksie 2, zaś wartość elementu o indeksie $n - 1$ w elemencie o indeksie 0),
 - d) `(*,r,!)` sortuje rosnąco elementy tablicy `tab` (porządkuje elementy przechowywane w tablicy w taki sposób aby ciąg `tab[0], tab[1], ..., tab[n-1]` był ciągiem niemalejącym),
 - e) sortuje malejąco elementy tablicy `tab`.
- 4.2.13 `(*,r,!)` Napisz funkcję, która otrzymuje jako argument dodatnią liczbę całkowitą n , a następnie tworzy dynamiczną n -elementową tablicę o elementach typu `int` i zwraca jako wartość wskaźnik do jej pierwszego elementu.
- 4.2.14 `(*)` Napisz funkcję, która otrzymuje jako argument dodatnią liczbę całkowitą n , a następnie tworzy dynamiczną n -elementową tablicę o elementach typu `double` i zwraca jako wartość wskaźnik do jej pierwszego elementu.
- 4.2.15 `(*,r,!)` Napisz funkcję, która dostaje jako argument wskaźnik do jed-

nowymiarowej dynamicznej tablicy o elementach typu `int` i zwalnia pamięć zajmowaną przez przekazaną w argumencie tablicę.

- 4.2.16 (*) Napisz funkcję, która dostaje jako argument wskaźnik do jednowymiarowej dynamicznej tablicy o elementach typu `double` i zwalnia pamięć zajmowaną przez przekazaną w argumencie tablicę.
- 4.2.17 (*) Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n -elementową tablicę `tab` o elementach typu `double` a następnie tworzy kopię tablicy `tab` i zwraca jako wartość wskaźnik do nowo utworzonej kopii.
- 4.2.18 (*) Napisz funkcję, która otrzymuje trzy argumenty: dodatnią liczbę całkowitą n oraz dwie tablice n -elementowe o elementach typu `int` przechowujące współrzędne wektorów i zwraca jako wartość wskaźnik do pierwszego elementu nowo utworzonej tablicy przechowującej sumę wektorów otrzymanych w argumentach.
- 4.2.19 (*) Napisz funkcję, która dostaje w argumentach dodatnią liczbę całkowitą n oraz n -elementową tablicę liczb całkowitych `tab1` o elementach typu `int` i przepisuje do nowo utworzonej tablicy `tab2` elementy tablicy `tab1` o wartości różnej od zera. Rozmiar tablicy `tab2` powinien być równy liczbie niezerowych elementów tablicy `tab1`. Jako wartość funkcja powinna zwrócić wskaźnik na pierwszy element tablicy `tab2`.

ROZDZIAŁ 5

NAPISY

5.1.	Wprowadzenie	24
5.2.	Zadania	25

5.1. Wprowadzenie

Operacje na napisach, nazywanych też łańcuchami lub z języka angielskiego stringami, to jedna z najslabszych stron języka C. Operacje te wymagają od programisty szczególnej ostrożności. Z tego powodu, pomimo iż napisy są zwykłymi tablicami jednowymiarowymi o elementach typów znakowych,

to poświęcamy im oddzielny rozdział. W języku C++ można używać napisów takich jak w C, jednak w C++ istnieją specjalne klasy służące do ich przechowywania: **string** i **wstring**. Klasy te są znacznie wygodniejsze w użyciu od tablic znaków.

Na początku podrozdziału 5.2 są zadania, które należy rozwiązać bez używania funkcji bibliotecznych. Tą część uczący się języka C++ mogą pominąć, mogą ją też potraktować jako ćwiczenia z operacjami na tablicach jednowymiarowych. W drugiej części podrozdziału 5.2 są zadania, w których można używać funkcji bibliotecznych. Znajdują się tam także zadania, przeznaczone specjalnie dla uczących się C++, w których należy użyć typów **string** i **wstring**.

Napisy w języku C są przechowywane w jednowymiarowych tablicach o elementach typów znakowych. W języku C są dwa podstawowe typy znakowe **char** i **wchar_t** oraz modyfikacje typu **char**: **unsigned char** i **signed char**. Standard języka C nie określa długości zmiennych o typach **char** i **wchar_t**. Typ **char** jest określony jako typ wystarczający do przechowywania podstawowego zestawu znaków na danym komputerze (w praktyce **char** jest typem jednobajtowym), zaś typ **wchart_t** powinien wystarczyć do przechowywania pełnego zestawu znaków dostępnego na danym komputerze.

Na końcu poprawnego napisu w języku C (niezależnie od typu znaków z których się składa) znajduje się znak o kodzie 0. Służy on do zaznaczenia końca napisu. Tablica przechowująca n -znakowy napis musi mieć co najmniej $n + 1$ elementów (może mieć więcej), aby móc przechować kończący napis znak o kodzie 0. Konieczność dbania o to, żeby na końcu napisu zawsze był znak o kodzie 0, jest jedną z dwóch głównych niedogodności przy operowaniu na napisach w języku C.

Operacje na napisach o elementach typu **char** można znaleźć w bibliotece **string**, zaś niektóre operacje na znakach tego typu także w bibliotece **ctype**. Typ **wchar_t** oraz funkcje operujące na zmiennych tego typu, odpowiadające operacjom z innych bibliotek standardowych (w tym z biblioteki **string**) znajdują się w bibliotece **wchar**. Odpowiedniki funkcji z bibliotek **ctype** operujące na typie **wchar_t** znajdują się w bibliotece **wctype**. Opisy funkcji z powyższych bibliotek czytelnik znajdzie w literaturze. Funkcje z bibliotek standardowych wymagają dbania o to, żeby używane tablice znaków

były wystarczających rozmiarów. Jest to druga ze wspomnianych wcześniej dwóch najważniejszych niedogodności przy operowaniu na napisach w języku C.

W języku C++ do przechowywania napisów służą klasy `string` i `wstring`. Jedyną różnicą pomiędzy tymi klasami jest typ znaków, z których składają się napisy. W obiektach klasy `string` znaki są typu `char`, natomiast napisy przechowywane w obiektach klasy `wstring` składają się ze znaków typu `wchart_t`. Klasy `string` i `wstring` są zdefiniowane w bibliotece `string` języka C++.

Uwaga! Biblioteka `string` języka C++ i biblioteka o tej samej nazwie z języka C to dwie różne biblioteki. Biblioteka `string` języka C jest dostępna w języku C++ pod nazwą `cstring`.

Używanie klas `string` i `wstring` znacznie ułatwia operowanie na napisach. W szczególności operowanie na tych klasach uwalnia nas od wspomnianych wcześniej dwóch głównych mankamentów operacji na napisach w języku C, czyli konieczności dbania o obecność znaku o kodzie 0 na końcu napisu i o odpowiedni rozmiar używanych tablic. Napisy przechowywane w obiektach klas `string` i `wstring` mogą zawierać znaki o kodzie 0 jako normalne znaki w napisie, zaś metody zdefiniowane dla tych klas dbają za nas o przydział pamięci dla przechowywanych napisów. O różnicy w łatwości operowania na różnych rodzajach napisów można się przekonać analizując rozwiązania zadań z następnego podrozdziału. Opis klas `string` i `wstring` czytelnik znajdzie w literaturze.

5.2. Zadania

Następujące zadania rozwiąż nie używając funkcji bibliotecznych:

- 5.2.1 (r) Napisz funkcję `wyczysc`, która usuwa z tablicy przechowywany w niej napis (w sensie: umieszcza w niej poprawny napis o długości 0). Napisz dwie wersje funkcji `wyczysc` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.2 (r) Napisz funkcję `dlugosc`, która jako argument otrzymuje napis i zwraca jako wartość jego długość. Napisz dwie wersje funkcji `dlugosc` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.3 Napisz funkcję `porownaj`, która jako argumenty otrzymuje dwa napisy i zwraca 1 gdy napisy są równe i 0 w przeciwnym przypadku. Napisz dwie wersje funkcji `porownaj` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.4 (r,!) Napisz funkcję, która jako argumenty otrzymuje dwa napisy i zwraca wartość 1, gdy pierwszy napis jest wcześniejszy w kolejności leksykograficznej i 0 w przeciwnym przypadku.

Zakładamy, że oba napisy składają się ze znaków typu `char`, zawierają wyłącznie małe litery alfabetu łacińskiego, a system, na którym jest komplikowany i uruchamiany program, używa standardowego kodowania ASCII.

- 5.2.5 Napisz funkcję `przepisz`, która otrzymuje dwie tablice znaków i przepisuje串nypis znajdujący się w pierwszej tablicy do drugiej tablicy. Zakładamy, że w drugiej tablicy jest wystarczająco dużo miejsca. Napisz dwie wersje funkcji `przepisz` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.6 Napisz funkcję `kopiujn`, która dostaje w argumentach dwie tablice znaków `nap1`, `nap2` oraz liczbę n i przekopiowuje n pierwszych znaków z napisu przechowywanego w tablicy `nap1` do tablicy `nap2`. W przypadku gdy串nypis w tablicy `nap1` jest krótszy niż n znaków, funkcja powinna po prostu przepisać串nypis. Funkcja powinna zadbać o to, żeby na końcu napisu w tablicy `nap2` znalazła się znak o kodzie 0.
Zakładamy, że w docelowej tablicy jest wystarczająco dużo miejsca. Napisz dwie wersje funkcji: dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.7 (r) Napisz funkcję `sklej` otrzymującą jako argumenty trzy tablice znaków i zapisującą do trzeciej tablicy konkatenację napisów znajdujących się w dwóch pierwszych tablicach (czyli dla napisów "Ala m" i "a kota" znajdujących się w pierwszych dwóch argumentach do trzeciej tablicy powinien zostać zapisany串nypis "Ala ma kota"). Zakładamy, że w trzeciej tablicy jest wystarczająco dużo miejsca.
Napisz dwie wersje funkcji `sklej` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.8 (r,!) Napisz funkcję, która otrzymuje w argumencie串nypis i podmienia w nim wszystkie małe litery na duże litery.
Zakładamy, że napis przechowywany jest w tablicy o elementach typu `char`, składa się wyłącznie z liter łacińskich i białych znaków, oraz że system, na którym jest komplikowany i uruchamiany program, używa standardowego kodowania ASCII.
- 5.2.9 (r) Napisz funkcję `wytnij`, która dostaje jako argumenty串nypis oraz dwie liczby całkowite n i m , i wycina z otrzymanego napisu znaki o indeksach od n do m ($n \leq m$). Otrzymany w argumencie串nypis może mieć dowolną liczbę znaków (w tym mniejszą od n lub m)
Napisz dwie wersje funkcji `wytnij` dla napisów składających się ze znaków typu `char` i `wchar_t`.
- 5.2.10 (*,r) Napisz funkcję `wytnij2`, która dostaje jako argument dwa napisy `nap1` i `nap2`, i wycina z napisu `nap1` pierwsze wystąpienie w nim napisu `nap2`. Napisz dwie wersje funkcji `wytnij2` dla napisów składających się ze znaków typu `char` i `wchar_t`.

- 5.2.11 **(*,r)** Napisz funkcję **wytnijzw**, która dostaje jako argument dwa napisy **nap1** i **nap2**, i wycina z napisu **nap1** wszystkie znaki występujące także w napisie **nap2**. Napisz dwie wersje funkcji **wytnijz** dla napisów składających się ze znaków typu **char** i **wchar_t**.
- 5.2.12 **(*)** Napisz funkcję **wytnijzn**, która dostaje jako argument dwa napisy **nap1** i **nap2**, i wycina z napisu **nap1** wszystkie znaki niewystępujące w napisie **nap2**. Napisz dwie wersje funkcji **wytnijzn** dla napisów składających się ze znaków typu **char** i **wchar_t**.
- 5.2.13 **(*,r)** Napisz funkcję **wytnijtm**, która dostaje jako argument dwa napisy **nap1** i **nap2** o一样的 długości i wycina z napisu **nap1** znaki równe znakom występującym na tym samym miejscu w napisie **nap2** (znak o indeksie i usuwamy wtedy i tylko wtedy, gdy **nap1[i]=nap2[i]**). Napisz dwie wersje funkcji **wytnijtm** dla napisów składających się ze znaków typu **char** i **wchar_t**.

Dalsze zadania rozwiąż z użyciem funkcji bibliotecznych:

- 5.2.14 **(r)** Napisz funkcję, która wypisuje na standardowym wyjściu otrzymany w argumencie napis. Napisz dwie wersje funkcji: dla napisów składających się ze znaków typu **char** i **wchar_t**.
- 5.2.15 **(C++,r)** Napisz funkcję, która wypisuje na standardowym wyjściu otrzymany w argumencie napis. Napisz dwie wersje funkcji: dla napisów typu **string** i **wstring**.
- 5.2.16 **(r,!)** Napisz funkcję, która dostaje jako argument tablicę znaków i wczytuje do niej napis ze standardowego wejścia. Napisz dwie wersje funkcji: dla tablicy składających się ze znaków typu **char** i **wchar_t**.
- 5.2.17 **(C++,r)** Napisz funkcję, która dostaje w argumentach referencję do zmiennej typu **string** i wczytuje do niej napis ze standardowego wejścia. Napisz drugą wersję funkcji dla napisu typu **wstring**.
- 5.2.18 **(*,r)** Napisz funkcję, która otrzymuje w argumencie tablicę napisów (tablicę tablic a więc typ **char**** lub **wchar_t****) oraz jej rozmiar i zwraca jako wartość pierwszy leksykograficznie spośród przechowywanych w tablicy napisów (funkcja powinna zwrócić kopię znajdującą się w tablicy napisu).
- Zakładamy, że napisy zawierają wyłącznie małe litery łacińskie. Napisz dwie wersje funkcji: dla napisów składających się ze znaków typu **char** i **wchar_t**.
- 5.2.19 **(C++,r,!)** Napisz funkcję, która otrzymuje w argumencie tablicę napisów (tablicę o elementach typu **string** lub **wstring**) oraz jej rozmiar i zwraca jako wartość pierwszy leksykograficznie spośród przechowywanych w tablicy napisów (funkcja powinna zwrócić kopię znajdującą się w tablicy napisu).

Zakładamy, że napisy zawierają wyłącznie małe litery łacińskie. Napisz dwie wersje funkcji: dla napisów typu **string** i **wstring**.

- 5.2.20 **(r)** Napisz funkcję **godzina**, która dostaje w argumentach trzy liczby całkowite **godz**, **min** i **sek**, zawierające odpowiednio godziny, minuty oraz sekundy, i zwraca jako wartość napis zawierający godzinę w formacie **godz:min:sek**, w którym wartości poszczególnych pól pochodzą ze zmiennych podanych w argumentach.

Napisz dwie wersje funkcji **godzina**: zwracające napisy będące tablicami znaków typu **char** i typu **wchar_t**.

- 5.2.21 **(C++,r)** Napisz funkcję **godzina**, która dostaje w argumentach trzy liczby całkowite **godz**, **min** i **sek**, zawierające odpowiednio godziny, minuty oraz sekundy, i zwraca jako wartość napis zawierający godzinę w formacie **godz:min:sek**, w którym wartości poszczególnych pól pochodzą ze zmiennych podanych w argumentach .

Napisz dwie wersje funkcji **godzina**: zwracające napisy typu **string** i typu **wstring**.

- 5.2.22 **(r)** Napisz funkcję **sklej**, która dostaje w argumentach trzy napisy i zwraca jako wartość napis powstały ze sklejenia napisów otrzymanych w argumentach.

Napisz dwie wersje funkcji **sklej** operujące na napisach składających się ze znaków typu **char** i typu **wchar_t**.

- 5.2.23 **(C++,r,!)** Napisz funkcję **sklej**, która dostaje w argumentach trzy napisy i zwraca jako wartość napis powstały ze sklejenia napisów otrzymanych w argumentach.

Napisz dwie wersje funkcji **sklej** operujące na napisach typu **string** i typu **wstring**.

- 5.2.24 Napisz funkcję **kopiuj**, która dostaje jako argumenty napis oraz tablicę znaków i przepisuje napis do otrzymanej w argumencie tablicy znaków.

Napisz dwie wersje funkcji **kopiuj** operujące na napisach składających się ze znaków typu **char** i typu **wchar_t**.

- 5.2.25 Napisz funkcję **kopiuj**, która dostaje jako argumenty napis oraz wskaźnik do napisu (czyli wskaźnik do wskaźnika), tworzy nową tablicę znaków, kopiuje do niej napis zawarty w pierwszym argumencie, i przypisuje wskaźnik do nowo utworzonej tablicy do zmiennej wskazywanej przez drugi argument.

Napisz dwie wersje funkcji **kopiuj** operujące na napisach składających się ze znaków typu **char** i typu **wchar_t**.

- 5.2.26 **(r)** Napisz funkcję, która dostaje w argumencie napis i zamienia wszystkie występujące w nim małe litery na odpowiadające im duże litery.

Napisz dwie wersje funkcji operujące na napisach składających się ze znaków typu **char** i typu **wchar_t**.

5.2.27 (**C++,r**) Napisz funkcje, która dostaje w argumencie referencję do napisu i zamienia wszystkie występujące w nim małe litery na odpowiadające im duże litery.

Napisz dwie wersje funkcji operujące na napisach typów `string` i `wstring`.

ROZDZIAŁ 6

TABLICE WIELOWYMIAROWE

6.1.	Wprowadzenie	32
6.2.	Zadania	32

6.1. Wprowadzenie

W językach C i C++ są dwa rodzaje tablic dwuwymiarowych. Jeden rodzaj to tablice których elementami są tablice jednowymiarowe zaś drugi to tablice wskaźników do tablic jednowymiarowych. Analogicznie możemy stworzyć w C i C++ cztery rodzaje tablic trójwymiarowych różniące się sposobem utworzenia poszczególnych wymiarów. Nie istnieje ustalone polskie nazewnictwo rozróżniające różne rodzaje dynamicznych tablic wielowymiarowych w C i C++. W niniejszym zbiorze na określenie dwóch głównych typów tablic wielowymiarowych używa się określeń tablice tablic (na przykład dla typów `int**` czy `int **`) i tablice wielowymiarowe (na przykład dla typów `int [] [n]` lub `int [] [n] [m]`).

W język C++ nie ma możliwości używania wielowymiarowych tablic o wymiarach nieznanych w czasie komilacji. W związku z tym wiele zadań w podrozdziale 6.2 przeznaczonych jest tylko dla uczących się języka C. Powyższy problem nie dotyczy wielowymiarowych tablic tablic.

6.2. Zadania

- 6.2.1 **(r,!)** Napisz funkcję, która dostaje jako argument dodatnie liczby całkowite n i m , tworzy dynamiczną dwuwymiarową tablicę tablic elementów typu `int` o wymiarach n na m , i zwraca jako wartość wskaźnik do niej.
- 6.2.2 **(C,r,!)** Napisz funkcję, która dostaje jako argument dodatnie liczby całkowite n i m , tworzy dynamiczną dwuwymiarową tablicę elementów typu `int` o wymiarach n na m i zwraca jako wartość wskaźnik do niej.
- 6.2.3 **(r,!)** Napisz funkcję, która dostaje jako argumenty wskaźnik do dwuwymiarowej tablicy tablic elementów typu `int` oraz jej wymiary: dodatnie liczby całkowite n i m , i usuwa z pamięci otrzymaną tablicę.
- 6.2.4 **(C,r)** Napisz funkcję, która dostaje jako argumenty wskaźnik do tablicy dwuwymiarowej elementów typu `int` oraz jej wymiary wymiary n i m , i usuwa z pamięci otrzymaną tablicę.
- 6.2.5 Rozwiąż zadania 6.2.1 i 6.2.3 w wersji z trójwymiarowymi tablicami tablic.
- 6.2.6 **(C)** Rozwiąż zadania 6.2.2 i 6.2.4 w wersji z tablicami trójwymiarowymi.
- 6.2.7 **(r)** Napisz funkcję, która dostaje jako argument dodatnią liczbę całkowitą n , tworzy dynamiczną dwuwymiarową trójkątną tablicę tablic elementów typu `int` o wymiarach n na n i zwraca jako wartość wskaźnik do niej.

- 6.2.8 (r) Napisz funkcję, która dostaje w argumentach dwuwymiarową tablicę elementów typu `int`, o pierwszym wymiarze podanym jako drugi argument funkcji oraz drugim wymiarze równym 100 i wypełnia ją zerami.
- 6.2.9 (r) Napisz funkcję, która dostaje w argumentach dwuwymiarową tablicę tablic elementów typu `int` oraz jej wymiary n i m , i wypełnia ją zerami.
- 6.2.10 (C,r) Napisz funkcję, która dostaje w argumentach tablicę dwuwymiarową elementów typu `int` oraz jej wymiary n i m , i wypełnia ją zerami.
- 6.2.11 Napisz funkcję, która dostaje w argumentach tablicę dwuwymiarową elementów typu `int`, o pierwszym wymiarze podanym jako drugi argument funkcji oraz drugim wymiarze równym 100, która to funkcja zwraca jako wartość sumę wartości elementów tablicy.
- 6.2.12 Napisz funkcję, która dostaje w argumentach dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary n i m , i zwraca jako wartość sumę wartości elementów tablicy.
- 6.2.13 (C) Napisz funkcję, która dostaje w argumentach tablicę dwuwymiarową o elementach typu `int` oraz jej wymiary n i m , i zwraca jako wartość sumę wartości elementów tablicy.
- 6.2.14 (r) Napisz funkcję, która dostaje w argumentach tablicę trójwymiarową o elementach typu `int` o wymiarach $100 \times 100 \times 100$, i zwraca jako wartość sumę wartości elementów tablicy.
- 6.2.15 Napisz funkcję, która dostaje w argumentach dodatnią liczbę całkowitą n oraz tablicę trójwymiarową o elementach typu `int` o wymiarach $n \times 100 \times 100$, i zwraca jako wartość sumę wartości elementów otrzymanej tablicy.
- 6.2.16 (r) Napisz funkcję, która dostaje jako argumenty dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i zwraca jako wartość indeks wiersza o największej średniej wartości elementów. Przyjmujemy, że dwa elementy leżą w tym samym wierszu, jeżeli mają taki sam pierwszy indeks.
- 6.2.17 (r) Napisz funkcję, która dostaje jako argumenty dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i zwraca największą spośród średnich wartości elementów poszczególnych wierszy. Przyjmujemy, że dwa elementy leżą w tym samym wierszu, jeżeli mają taki sam pierwszy indeks.
- 6.2.18 Napisz funkcję, która dostaje jako argument dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i wypisuje jej zawartość na standardowym wyjściu w taki sposób, żeby kolejne wiersze tablicy zostały wypisane w oddzielnych wierszach standardowego wyjścia.

Przyjmujemy, że dwa elementy leżą w tym samym wierszu, jeżeli mają taki sam drugi indeks.

- 6.2.19 **(r)** Napisz funkcję, która dostaje jako argumenty dwie dwuwymiarowe tablice tablic o elementach typu `int` oraz ich wymiary, i przepisuje zawartość pierwszej tablicy do drugiej tablicy.
- 6.2.20 Napisz funkcję, która dostaje jako argumenty dwie dwuwymiarowe tablice tablic o elementach typu `int` oraz ich wymiary, i zamienia zawartości obu tablic.
- 6.2.21 **(r)** Napisz funkcję, która dostaje jako argumenty dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i odwraca kolejność elementów we wszystkich wierszach otrzymanej tablicy (przyjmujemy, że dwa elementy tablicy leżą w tym samym wierszu, jeżeli mają taką samą pierwszą współrzędną).
- 6.2.22 **(C, r)** Napisz funkcję, która dostaje jako argumenty tablicę dwuwymiarową o elementach typu `int` oraz jej wymiary, i odwraca kolejność elementów we wszystkich wierszach otrzymanej tablicy (przyjmujemy, że dwa elementy tablicy leżą w tym samym wierszu, jeżeli mają taką samą pierwszą współrzędną).
- 6.2.23 **(r, !)** Napisz funkcję, która dostaje jako argumenty dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i zmienia kolejność wierszy w tablicy w taki sposób, że wiersz pierwszy ma się znaleźć na miejscu drugiego, wiersz drugi ma się znaleźć na miejscu trzeciego itd., natomiast ostatni wiersz ma się znaleźć na miejscu pierwszego (przyjmujemy, że dwa elementy tablicy leżą w tym samym wierszu jeżeli mają taką samą pierwszą współrzędną).
- 6.2.24 Napisz funkcję, która dostaje jako argumenty dwuwymiarową tablicę tablic o elementach typu `int` oraz jej wymiary, i zmienia kolejność kolumn w tablicy w taki sposób, że kolumna pierwsza ma się znaleźć na miejscu drugiej, kolumna druga ma się znaleźć na miejscu trzeciej itd., natomiast ostatnia kolumna ma się znaleźć na miejscu pierwszej (przyjmujemy, że dwa elementy tablicy leżą w tej samej kolumnie, jeżeli mają taką samą drugą współrzędną).
- 6.2.25 Napisz funkcję, która dostaje jako argumenty dwuwymiarową kwadratową tablicę tablic `tab` o elementach typu `int` oraz jej wymiar, i zmienia kolejność elementów w otrzymanej tablicy w następujący sposób: dla dowolnych `k` i `j` element `tab[k][j]` ma zostać zamieniony miejscami z elementem `tab[j][k]`.
- 6.2.26 Napisz funkcję, która dostaje jako argumenty dwuwymiarową prostokątną tablicę tablic `tab1` o wymiarach $n \times m$ i elementach typu `int` oraz jej wymiary, i zwraca jako wartość wskaźnik do nowo utworzonej dwuwymiarowej tablicy `tab2` o wymiarach $m \times n$ zawierającej

transponowaną macierz przechowywaną w tablicy `tab1` (czyli dla dowolnych k i j zachodzi `tab1[k][j]=tab2[j][k]`).

- 6.2.27 (***,r**) Napisz funkcję, która dostaje jako argumenty dodatnią liczbę całkowitą n oraz trójwymiarową tablicę `tab` elementów typu `int` o wymiarach $n \times n \times n$, i zamienia elementy tablicy w taki sposób, że dla dowolnych i, j, k z zakresu od 0 do $n - 1$, wartość z komórki `tab[i][j][k]` po wykonaniu funkcji ma się znajdować w komórce `tab[k][i][j]`.
- 6.2.28 (**C,***) Napisz funkcję, która dostaje jako argumenty dodatnią liczbę całkowitą n oraz trójwymiarową tablicę `tab` elementów typu `int` o wymiarach $n \times n \times n$, i zamienia elementy tablicy w taki sposób, że dla dowolnych i, j, k z zakresu od 0 do $n - 1$, takich że $i \leq j \leq k$, wartość z komórki `tab[i][j][k]` po wykonaniu funkcji ma się znajdować w komórce `tab[k][i][j]`.
- 6.2.29 (*) Napisz funkcję, która dostaje jako argument trójwymiarową tablicę `tab` elementów typu `int` o wymiarach $100 \times 100 \times 100$, i zamienia elementy tablicy w taki sposób, że dla dowolnych i, j, k z zakresu od 0 do 99, takich że $i \leq j \leq k$, wartość z komórki `tab[i][j][k]` po wykonaniu funkcji ma się znajdować w komórce `tab[k][i][j]`.
- 6.2.30 Napisz funkcję, która otrzymuje w argumentach dwie kwadratowe tablice tablic elementów typu `int` oraz ich wspólny wymiar, i zwraca jako wartość wynik dodawania macierzy przechowywanych w przekazanych argumentach. Wynik powinien zostać zwrócony w nowo utworzonej tablicy tablic.
- 6.2.31 (**C**) Napisz funkcję, która otrzymuje w argumentach dwie kwadratowe tablice dwuwymiarowe elementów typu `int` oraz ich wspólny wymiar, i zwraca jako wartość wynik dodawania macierzy przechowywanych w przekazanych argumentach. Wynik powinien zostać zwrócony w nowo utworzonej tablicy dwuwymiarowej.
- 6.2.32 (***,r**) Napisz funkcję, która otrzymuje w argumentach dwie kwadratowe tablice tablic elementów typu `int` oraz ich wspólny wymiar, i zwraca jako wartość wynik mnożenia macierzy przechowywanych w przekazanych argumentach. Wynik powinien zostać zwrócony w nowo utworzonej tablicy tablic.
- 6.2.33 Napisz funkcję, która otrzymuje w argumentach trzy kwadratowe tablice tablic elementów typu `int` oraz ich wspólny wymiar, i zapisuje do trzeciej tablicy wynik mnożenia macierzy przechowywanych w dwóch pierwszych tablicach.
- 6.2.34 (**C,***) Napisz funkcję, która otrzymuje w argumentach dwie kwadratowe tablice dwuwymiarowe elementów typu `int` oraz ich wspólny wymiar, i zwraca jako wartość wynik mnożenia macierzy przechowywają-

nych w przekazanych argumentach. Wynik powinien zostać zwrócony w nowo utworzonej tablicy dwuwymiarowej.

- 6.2.35 (*) Napisz funkcję, która otrzymuje w argumentach dwie prostokątne dwuwymiarowe tablice tablic elementów typu `int` o wymiarach odpowiednio $n \times m$ i $m \times k$ oraz ich wymiary, i zwraca jako wartość wynik mnożenia macierzy przechowywanych w przekazanych argumentach. Wynik powinien zostać zwrócony w nowo utworzonej tablicy tablic.
- 6.2.36 (*,r,!) Napisz funkcję otrzymującą dwa argumenty: dodatnią liczbę całkowitą n i dwuwymiarową kwadratową tablicę tablic elementów typu `int` o wymiarach $n \times n$, i zwraca jako wartość wyznacznik macierzy przechowywanej w otrzymanej w argumencie tablicy.
- 6.2.37 (C,*,r,!) Napisz funkcję otrzymującą dwa argumenty: dodatnią liczbę całkowitą n i kwadratową tablicę dwuwymiarową elementów typu `int` o wymiarach $n \times n$, i zwraca jako wartość wyznacznik macierzy przechowywanej w otrzymanej w argumencie tablicy.

ROZDZIAŁ 7

ZŁOŻONE TYPY DANYCH, LISTY WSKAŹNIKOWE

7.1.	Wprowadzenie	38
7.2.	Złożone typy danych	38
7.3.	Listy jednokierunkowe	41

7.1. Wprowadzenie

W tym rozdziale czytelnik znajdzie zadania związane ze złożonymi typami danych dostępnymi w C, a więc: strukturami, uniami i typami wyliczeniowymi. Duży nacisk został położony na zadania dotyczące list wskaźnikowych, jednego z popularnych zastosowań struktur. Zadania dotyczące list wskaźnikowych to okazja do przećwiczenia operacji na wskaźnikach oraz zarządzania pamięcią.

Istnieje wiele rodzajów list wskaźnikowych. Jednak ze względu na to, że nie jest to zbiór zadań ze struktur danych, w zadaniach pojawiają się wyłącznie listy jednokierunkowe. Zadania dotyczą zarówno list bez głowy, czyli takich, w których na początku listy znajduje się jej pierwszy element, jak i list z głową, czyli takich, w których na początku listy znajduje się sztuczny element nazywany głową. Dzięki dodaniu do listy głowy wiele operacji się upraszcza.

W języku C++ struktury to prawie to samo, co klasy. Różnią się tylko tym, że w przeciwieństwie do klas, pola i metody w strukturach są domyślnie publiczne. W tym rozdziale traktujemy jednak struktury wyłącznie jako tradycyjne złożone typy danych, tak jak jest to w języku C.

W podrozdziale 7.2 czytelnik znajdzie zadania mające na celu przećwiczenie definiowania złożonych typów danych oraz wykonywania prostych operacji na nich. W podrozdziale 7.3 znajduje się urozmaicony zestaw zadań, umożliwiający przećwiczenie używania list jednokierunkowych z głową i bez głowy. Na początku tego rozdziału znajdują się zadania wymagające implementacji podstawowych operacji na listach. Ze względu na problemy, jakie operowanie na listach sprawia początkującym programistom, wszystkie zadania z tej części podrozdziału 7.3 zostały rozwiązane.

7.2. Złożone typy danych

- 7.2.1 (**r,róż**) Zdefiniuj strukturę **trojkat** przechowującą długości boków trójkąta. Napisz funkcję, która otrzymuje jako argument zmienią typu **struct trojkat**, i zwraca jako wartość obwód trójkąta przekazanego w argumencie.
- 7.2.2 (**r,!**) Napisz funkcję, która otrzymuje jako argumenty zmiennej **troj1** typu **struct trojkat** zdefiniowanego w zadaniu 7.2.1 oraz zmiennej **troj2** wskaźnik na zmiennej typu **struct trojkat**, i przepisuje wartość zmiennej **troj1** do zmiennej wskazywanej przez **troj2**.
- 7.2.3 (**r**) Zdefiniuj strukturę **punkt** służącą do przechowywania współrzędnych punktów w trójwymiarowej przestrzeni kartezjańskiej.

Napisz funkcję, która otrzymuje jako argumenty tablicę `tab` o argumentach typu `struct punkt` oraz jej rozmiar, i zwraca jako wartość najmniejszą spośród odległości pomiędzy punktami przechowywanymi w tablicy `tab`. Zakładamy, że otrzymana w argumencie tablica ma co najmniej dwa argumenty.

7.2.4 (r,!) Napisz funkcję, która otrzymuje jako argumenty tablice `tab1` i `tab2` o argumentach typu `struct punkt` zdefiniowanego w rozwiązaniu zadania 7.2.3 oraz ich rozmiar, i przepisuje zwartość tablicy `tab1` do tablicy `tab2`.

7.2.5 (r,!) Zdefiniuj strukturę `punkt10` służącą do przechowywania współrzędnych punktów w dziesięciowymiarowej przestrzeni kartezjańskiej. Do przechowywania poszczególnych wymiarów wykorzystaj tablicę dziesięcioelementową.

Napisz funkcję, która otrzymuje jako argumenty tablice `tab1` i `tab2` typu `struct punkt10` oraz ich wspólny rozmiar, i przepisuje zwartość tablicy `tab1` do tablicy `tab2`.

7.2.6 (r,!) Zdefiniuj strukturę `punktn` służącą do przechowywania współrzędnych punktów w n -wymiarowej przestrzeni kartezjańskiej. Do przechowywania poszczególnych wymiarów wykorzystaj tablicę n -elementową. W strukturze `punktn` przechowuj także ilość wymiarów przestrzeni.

Napisz funkcję, która otrzymuje jako argumenty tablice `tab1` i `tab2` o argumentach typu `struct punktn` oraz ich wspólny rozmiar, i przepisuje zwartość tablicy `tab1` do tablicy `tab2`. Zakładamy, że tablica `tab2` jest pusta (czyli nie musimy się martwić o jej poprzednią zawartość).

7.2.7 Zdefiniuj strukturę `zespolone` służącą do przechowywania liczb zespolonych. Zdefiniowana struktura powinna zawierać pola `im` i `re` typu `double` służące do przechowywania odpowiednio części urojonej i rzeczywistej liczby zespolonej.

Napisz funkcję `dodaj`, która dostaje dwa argumenty typu `zespolone` i zwraca jako wartość ich sumę.

7.2.8 Zdefiniuj strukturę `student` służącą do przechowywania danych osobowych studenta (struktura powinna zawierać takie pola, jak: `imie`, `nazwisko`, `adres`, `pesel`, `kierunek` i `numer legitymacji`).

Napisz funkcję, która otrzymuje jako argument wskaźnik na strukturę `student` i wczytuje dane ze standardowego wejścia do rekordu wskazywanego przez argument funkcji.

7.2.9 (r) Zdefiniuj strukturę `lista` posiadającą dwa pola: jedno typu `int` oraz drugie będące wskaźnikiem do definiowanego typu.

- 7.2.10 (r) Zdefiniuj unię `super_int`, w której będzie można przechowywać zarówno zmienne typu `int`, jak i `unsigned int`.
- 7.2.11 (r) Zdefiniuj unię `Liczba`, która może służyć w zależności od potrzeb do przechowywania liczby wymiernej lub liczby całkowitej. Zdefiniuj strukturę `Dane`, o dwóch polach polu `tp` typu `int` oraz polu `zaw` typu `Liczba`.
Napisz bezargumentową funkcję, która wczytuje ze standardowego wejścia zawartość do struktury `Dane` i zwraca ją jako wartość. Funkcja powinna pytać użytkownika, czy chce wczytać liczbę całkowitą, czy wymierną oraz w zależności od jego wyboru wstawić do pola `tp` wartość 0 lub 1. Następnie funkcja powinna wczytać do pola `zaw` wartość odpowiedniego typu.
- 7.2.12 (*) Zdefiniuj strukturę `zespolone`, która ma służyć do przechowywania liczb zespolonych oraz unię `Liczba` mogącą przechowywać liczby wymierne i całkowite. Części urojona i rzeczywista liczby zespolonej powinny być przechowywane w polach `im` i `re` typu `Liczba`. Struktura `zespolone` powinna mieć dodatkowe pole `tp` przechowujące informację jakiego typu wartości przechowywane są w polach `im` i `re` (zakładamy, że oba są tego samego typu).
Napisz funkcję `dodaj`, która dostaje dwa argumenty typu `zespolone` i zwraca jako wartość ich sumę. Zwróć uwagę na zgodność typów składników i zwracanej wartości (suma dwóch liczb całkowitych jest liczbą całkowitą, natomiast jeżeli którykolwiek ze składników jest wymierny to i suma jest wymierna).
- 7.2.13 (r) Zdefiniuj strukturę `figura` przechowującą wymiary figur geometrycznych niezbędne do obliczenia pola. Struktura powinna mieć możliwość przechowywania wymiarów takich figur, jak: trójkąt, prostokąt, równoległobok i trapez. Rodzaj przechowywanej figury powinien być zakodowany w wartości pola `fig` typu `int`. Definiując strukturę, staraj się zużyć jak najmniej pamięci.
Napisz funkcję `pole`, która dostaje jako argument zmienną `f` typu `struct figura` i zwraca jako wartość pole figury której wymiary przechowuje zmienna `f`.
- 7.2.14 (r) Zdefiniuj typ wyliczeniowy `czworokat`, mogący przyjmować wartości odpowiadające nazwom różnych czworokątów.
- 7.2.15 Zdefiniuj typ wyliczeniowy `zwierzak`, mogący przyjmować wartości odpowiadające nazwom różnych zwierząt domowych.
- 7.2.16 Zdefiniuj typ wyliczeniowy, służący do przechowywania informacji o stanie połączenia internetowego, o trzech wartościach odpowiadających trzem stanom: połączenie nawiązane, połączenie nienawiązane i połączenie w trakcie nawiązywania. Następnie zdefiniuj strukturę `komputer`

przechowującą stan połączenia, IP podłączonego komputera (jako napis) oraz nazwę jego właściciela.

Napisz funkcję, która jako argument otrzymuje strukturę komputer i wyświetla na standardowym wyjściu jej zawartość w sposób przyjazny dla użytkownika.

- 7.2.17 (**r,!róż**) Zdefiniuj strukturę dane osobowe zawierającą pola: imie, nazwisko, plec, stan_cywילny. W zależności od płci pole stan_cywילny powinno móc mieć jedną z dwóch wartości wolny lub zonaty dla mężczyzn i wolna lub mezatka dla kobiet.

Napisz funkcję wczytaj o dwóch argumentach: tablicy tab o elementów typu stan_cywילny i jej rozmiarze. Funkcja powinna wczytywać do komórek tablicy tab wartości podane na standardowym wejściu.

- 7.2.18 (**r,!**) Zdefiniuj złożony typ danych dzięki któremu będzie można odnosić się do kolejnych bajtów zmiennej typu unsigned int jak do kolejnych elementów tablicy.

- 7.2.19 (**r**) Wykorzystując typ danych zdefiniowany w rozwiązaniu zadania 18 napisz funkcję, która dostaje w argumentach dwie nieujemne liczby całkowite typu unsigned int i zwraca jako wartość nieujemną liczbę całkowitą, której kolejne bajty są iloczynami modulo 256 odpowiadających sobie bajtów liczb podanych w argumentach.

7.3. Listy jednokierunkowe

Jednokierunkowe listy bez głowy

Listing 7.1. Definicja struktury element

```
struct element{
    int i;
    struct element * next;
};
```

Struktura element może być wykorzystana w implementacji jednokierunkowej listy wskaźnikowej. Najpierw zostaną przedstawione zadania umożliwiające przećwiczenie operacji na liście jednokierunkowej bez głowy (czyli takiej, która jest reprezentowana przez wskaźnik na pierwszy element). Listę pustą reprezentuje wskaźnik o wartości NULL. Pamiętaj, że pole next ostatniego elementu listy powinno mieć wartość NULL. Dzięki temu będzie możliwe rozpoznanie końca listy.

- 7.3.1 (**r**) Napisz funkcję utworz zwracającą wskaźnik do pustej listy bez głowy o elementach typu element.

- 7.3.2 (r) Napisz funkcję `wyczysc`, która dostaje jako argument wskaźnik do pierwszego elementu listy wskaźnikowej bezgłowy o elementach typu `element` i usuwa wszystkie elementy listy.
- 7.3.3 (r) Napisz funkcję `dodaj` o dwóch argumentach `Lista` typu `element*` oraz `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna dodawać na początek listy reprezentowanej przez zmienną `Lista` nowy element o wartości `a` pola `i` oraz zwracać wskaźnik do pierwszego elementu tak powiększonej listy.
- 7.3.4 (r) Napisz funkcję `dodajk` o dwóch argumentach `Lista` typu `element*` i `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna dodawać na koniec listy reprezentowanej przez zmienną `Lista` nowy element o wartości `a` pola `i` oraz zwracać wskaźnik do pierwszego elementu tak powiększonej listy.
- 7.3.5 (r) Napisz funkcję `dodajw` o trzech argumentach `Lista` i `elem` typu `element*` oraz `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna dodawać element o wartości `a` pola `i` do listy reprezentowanej przez zmienną `Lista` na miejscu tuż za elementem wskazywanym przez `elem`. W przypadku, gdy `elem` jest równy `NULL` funkcja powinna wstawić nowy element na początek listy. Funkcja powinna zwrócić jako wartość wskaźnik do pierwszego elementu powiększonej listy.
- 7.3.6 (r,!) Napisz funkcję `znajdz` o dwóch argumentach `Lista` typu `element*` i `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna sprawdzać, czy na liście reprezentowanej przez zmienną `Lista` znajduje się element o polu `i` równym `a`. Jeżeli tak, to funkcja powinna zwrócić wskaźnik do tego elementu. W przeciwnym wypadku funkcja powinna zwrócić wartość `NULL`.
- 7.3.7 (r) Napisz funkcję `usun` o dwóch argumentach `Lista` typu `element*` i `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna usuwać z listy reprezentowanej przez zmienną `Lista` element o wartości `a` pola `i` (o ile taki element znajduje się na liście) oraz zwracać wskaźnik do pierwszego elementu zmodyfikowanej listy (jeżeli po usunięciu elementu lista będzie pusta, to funkcja powinna zwrócić wartość `NULL`).
- 7.3.8 (r,!) Napisz funkcję `usunw` o dwóch argumentach `Lista` i `elem` typu `element*` i zwracającą wskaźnik do typu `element`. Funkcja powinna usuwać z listy reprezentowanej przez zmienną `Lista` element wskazywany przez `elem` oraz zwracać wskaźnik do pierwszego elementu zmodyfikowanej listy (jeżeli po usunięciu elementu lista będzie pusta, to funkcja powinna zwrócić wartość `NULL`). Dla `elem` równego `NULL` funkcja `usunw` nie powinna nic robić.
- 7.3.9 (r,!) Napisz funkcję `usunw2` o dwóch argumentach `Lista` i `elem` typu

`element*` i zwracającą wskaźnik do typu `element`. Funkcja powinna usuwać z listy reprezentowanej przez zmienną `Lista` element wskażywany przez `elem->next` oraz zwracać wskaźnik do pierwszego elementu zmodyfikowanej listy (jeżeli po usunięciu elementu lista będzie pusta, to funkcja powinna zwrócić wartość `NULL`). Dla `elem` równego `NULL` funkcja powinna usunąć pierwszy element listy (o ile taki istnieje). Dla `elem` różnego od `NULL` i `elem->next` równego `NULL` funkcja `usunw2` nie powinna nic robić.

Jednokierunkowe listy z głową

W zadaniach dotyczących jednokierunkowych list wskaźnikowych z głową (czyli takich, na początku których znajduje się „sztuczny” pusty element, nazywany głową) zostanie wykorzystana struktury `element` zdefiniowaną w Listingu 7.1 . Pamiętaj, że pole `next` ostatniego elementu listy powinno mieć wartość `NULL`, w ten sposób będzie możliwe rozpoznanie końca listy.

- 7.3.10 (r) Napisz funkcję `utworz` tworzącą pustą listę z głową o elementach typu `element` i zwracającą jako wartość wskaźnik do głowy utworzonej listy.
- 7.3.11 (r) Napisz funkcję `wyczysc`, która dostaje jako argument wskaźnik do listy wskaźnikowej z głową o elementach typu `element` i usuwa wszystkie elementy listy (razem z głową).
- 7.3.12 (r) Napisz funkcję `dodaj` o dwóch argumentach `Lista` typu `element*` i `a` typu `int`. Funkcja powinna dodawać na początek listy reprezentowanej przez zmienną `Lista` nowy element o wartości `a` pola `i`.
- 7.3.13 (r) Napisz funkcję `dodajk` o dwóch argumentach `Lista` typu `element*` i `a` typu `int`. Funkcja powinna dodawać na koniec listy reprezentowanej przez zmienną `Lista` nowy element o wartości `a` pola `i`.
- 7.3.14 (r) Napisz funkcję `dodajw` o trzech argumentach `Lista` oraz `elem` typu `element*` i `a` typu `int`. Funkcja powinna dodawać element o wartości `a` pola `i` do listy reprezentowanej przez zmienną `Lista` na miejscu tuż za elementem wskażowanym przez `elem`.
- 7.3.15 (r) Napisz funkcję `znajdz` o dwóch argumentach `Lista` typu `element*` i `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna sprawdzać, czy na liście reprezentowanej przez zmienną `Lista`, znajduje się element o polu `i` równym `a`. Jeżeli tak, to funkcja powinna zwrócić wskaźnik do tego elementu. W przeciwnym wypadku funkcja powinna zwrócić wartość `NULL`.
- 7.3.16 (r) Napisz funkcję `znajdzp` o dwóch argumentach `Lista` typu `element*` i `a` typu `int` zwracającą wskaźnik do typu `element`. Funkcja powinna sprawdzać, czy na liście reprezentowanej przez zmienną `Lista`, znajduje się element o polu `i` równym `a`. Jeżeli tak, to funkcja powinna zwrócić wskaźnik do elementu go poprzedzającego. W przeciwnym

wypadku funkcja powinna zwrócić wskaźnik do ostatniego elementu listy.

- 7.3.17 (r) Napisz funkcję **usun** o dwóch argumentach **Lista** typu **element*** i **a** typu **int**. Funkcja powinna usuwać z listy reprezentowanej przez zmienną **Lista** element o wartości **a** pola **i** (o ile taki element znajduje się na liście).
- 7.3.18 (r) Napisz funkcję **usunw** o dwóch argumentach **Lista** i **elem** typu **element***. Funkcja powinna usuwać z listy reprezentowanej przez zmienną **Lista** element wskazywany przez zmienną **elem**.
- 7.3.19 (r) Napisz funkcję **usunw2** o dwóch argumentach **Lista** i **elem** typu **element***. Funkcja powinna usuwać z listy reprezentowanej przez zmienną **Lista** element wskazywany przez **elem->next**.

Pozostałe zadania z list jednokierunkowych

- 7.3.20 (r) Napisz funkcję **zeruj**, która dostaje jako argument listę wskaźnikową o elementach typu **element** i nadaje wartość 0 polom **i** we wszystkich elementach listy. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.21 Napisz funkcję **bezwzględna**, która dostaje jako argument listę wskaźnikową o elementach typu **element** i zapisuje do pól **i** wszystkich elementów listy wartość bezwzględną ich pierwotnej wartości. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.22 Zdefiniuj strukturę **trojkat** mogącą służyć jako typ elementów listy jednokierunkowej. Struktura **trojkat** powinna posiadać pola służące do przechowywania wszystkich boków trójkąta oraz jego pola.
Napisz funkcję **pole**, która otrzymuje w argumencie listę wskaźnikową o elementach typu **trojkat** i we wszystkich elementach listy do odpowiedniego pola wstawia wartość pola trójkąta o bokach, których długość przechowuje dana struktura. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.23 (r) Zdefiniuj strukturę **trojka** mającą służyć jako typ elementu jednokierunkowej listy wskaźnikowej przechowującej trójkę dodatnich liczb całkowitych **a, b, c**.
Napisz funkcję **pitagoras**, która dostaje w argumencie listę wskaźnikową o elementach typu **trojka** i usuwa z otrzymanej listy wszystkie elementy nieprzechowujące trójkę pitagorejskich (czyli takich, że $a^2 + b^2 = c^2$). Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy. W wersji dla list bez głowy funkcja powinna zwracać wskaźnik do pierwszego elementu przekształconej listy. Jeżeli wynikowa lista bez głowy będzie pusta, funkcja powinna zwrócić **NULL**.
- 7.3.24 (r) Napisz funkcję **suma**, która dostaje jako argument listę wskaźnikową o elementach typu **element** i zwraca jako wartość sumę pól **i**

ze wszystkich elementów listy. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.

- 7.3.25 (r) Napisz funkcję `minimum`, która dostaje jako argument listę wskaźnikową o elementach typu `element` i zwraca jako wartość najmniejszą spośród wartości pól `i` elementów listy. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.26 (r) Napisz funkcję `minimum`, która dostaje jako argument `Lista` listę wskaźnikową o elementach typu `element` i zwraca jako wartość wskaźnik do elementu listy o najmniejszej wartości pola `i`. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.27 (r) Napisz funkcję `minimum`, która dostaje jako argument listę wskaźnikową o elementach typu `element` i zwraca jako wartość wskaźnik do bezpośredniego poprzednika elementu listy o najmniejszej wartości pola `i`. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy. W przypadku listy bez głowy, jeżeli najmniejszą wartość pola `i` ma pierwszy element listy lub gdy otrzymana w argumencie lista jest pusta, funkcja `minimum` powinna zwrócić wartość `NULL`.
- 7.3.28 (r) Napisz funkcję, która dostaje jako argument listę o elementach typu `element` i zwraca jako wartość największą na wartość bezwzględną spośród różnic pomiędzy polami `i` w różnych elementach listy otrzymanej w argumencie. Zakładamy, że otrzymana w argumencie funkcji lista jest co najmniej dwuelementowa. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.29 (r) Napisz funkcję `kopiuj`, która jako argument otrzymuje jednokierunkową listę wskaźnikową o elementach typu `element`, tworzy kopię otrzymanej w argumencie listy i zwraca jako wartość wskaźnik do kopii. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.30 (r) Napisz funkcję `doklej` o dwóch argumentach `Lista1` i `Lista2` typu `element *` wskazujących na dwie jednokierunkowe listy wskaźnikowe bez głowy, która wstawia na koniec listy `Lista1` elementy listy `Lista2` (funkcja nie powinna alokować w pamięci nowych zmiennych typu `element`, ale odpowiednio podpisać już istniejące). Funkcja powinna zwracać wskaźnik do pierwszego elementu połączonej listy.
- 7.3.31 (r,!) Napisz funkcję, która dostaje w argumentach wskaźnik do listy wskaźnikowej o elementach typu `element` i odwraca kolejność elementów listy. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy. W wersji dla list bez głowy funkcja powinna zwracać wskaźnik do pierwszego elementu odwróconej listy.
- 7.3.32 (r) Napisz funkcję która dostaje w argumentach wskaźniki do dwóch list wskaźnikowych bez głowy o elementach typu `element` i równej długości, tworzy listę złożoną z elementów obu list ułożonych naprzemienne (pierwszy ma być pierwszy element pierwszej listy, następ-

nie pierwszy element drugiej listy, drugie element pierwszej listy itd.) i zwraca jako wartość wskaźnik do pierwszego elementu nowo utworzonej listy.

- 7.3.33 (**r,!**) Napisz funkcję, która dostaje jako argument listę wskaźnikową o elementach typu **element** i przesuwa jej elementy w taki sposób, że pierwszy element będzie drugi, drugi element będzie trzeci etc. a na pierwszym miejscu będzie ostatni element pierwotnej listy. Dla listy o długości nie większej niż 1 funkcja nie powinna nic robić. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy. W wersji dla list bez głowy funkcja powinna zwracać wskaźnik do pierwszego elementu przekształconej listy.
- 7.3.34 (**r**) Napisz funkcję, która dostaje jako argumenty dwie listy wskaźnikowe **Lista1** i **Lista2** o elementach typu **element** i tworzy nową listę o elementach tego samego typu przechowującą wartości występujące w polach **i** elementów zarówno listy **Lista1** jak i **Lista2**. W stworzonej liście nie powinny powtarzać się przechowywane w elementach wartości. Funkcja powinna zwracać wskaźnik do nowo utworzonej listy. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy.
- 7.3.35 (***,r,!**) Napisz funkcję sortującą rosnąco podaną w argumencie listę o elementach typu **element**. Napisz dwie wersje funkcji: dla list z głową i dla list bez głowy. W wersji dla listy bez głowy funkcja powinna zwracać wskaźnik do pierwszego elementu posortowanej listy.

ROZDZIAŁ 8

OPERACJE NA PLIKACH

8.1.	Wprowadzenie	48
8.2.	Zadania	48

8.1. Wprowadzenie

W językach C i C++ pliki powiązane są ze strumieniami i pracuje się na nich podobnie jak na innych strumieniach. W języku C do operacji na plikach służą funkcje z biblioteki `stdio`, a wśród nich między innymi: `fopen`, `fclose`, `fwrite`, `fread`, `fprintf`, `fscanf`, `feof` i `fseek`. W języku C++ do operowania na plikach służą klasy `ifstream`, `ofstream` i `fstream` znajdujące się w bibliotece `fstream`. Szczegóły czytelnik znajdzie w literaturze.

Aby zacząć operować na pliku należy go otworzyć, zaś po zakończeniu pracy należy plik zamknąć. Otwierając plik należy określić w jakim celu plik jest otwierany (do czytania, do pisania etc.) oraz czy plik ma być otwarty w trybie binarnym (czyli jako ciąg bajtów) czy tekstowym. Aby zrozumieć różnicę pomiędzy plikiem binarnym a tekstowym można zapisać do obu rodzajów plików jednobajtową bezznakową liczbę całkowitą o wartości 100. W pliku binarnym znajdzie się jeden bajt zawierający binarnie zapisaną liczbę 100, natomiast w pliku tekstowym znajdą się trzy bajty, z których pierwszy będzie zawierał kod znaku '1', a dwa kolejne bajty będą zawierały kod znaku '0'.

Częstym błędem u początkujących programistów C jest niewłaściwe użycie funkcji `feof`. Używając jej należy pamiętać, że ta funkcja zwraca `true` dopiero wtedy, gdy nie powiedzie się próba czytania. Czyli to, że `feof` ma wartość `false`, nie oznacza, że w pliku pozostało jeszcze coś do przeczytania. Podobny problem dotyczy metody `eof` klasy `fstream`.

Inną rzeczą, o której należy pamiętać, to fakt, że strumieni w języku C++ nie można kopiować. Można przekazywać je co najwyżej przez referencję lub wskaźnik.

W treściach zadań wielokrotnie pojawia się pojęcie deskryptora. W systemie UNIX jest to liczba całkowita jednoznacznie identyfikująca otworzony plik. Chcąc pisać do lub czytać z otworzonego pliku należy podać jego deskryptor. W treści zadań dla uproszczenia pojęcie deskryptora używane jest w odniesieniu do wartości typu `FILE *`.

8.2. Zadania

8.2.1 (**r,róż**) Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku, otwiera plik do tekstowego czytania i zwraca jako wartość deskryptor świeżo otwartego pliku (w wersji dla języka C++ funkcja powinna zwrócić wskaźnik do obiektu klasy `fstream`).

8.2.2 (**r,!**) Napisz funkcję, która dostaje jako argument deskryptor do pliku tekstowego otwartego do czytania (w wersji dla języka C++ referencję

- do obiektu klasy `fstream`), wypisuje zawartość pliku na standardowe wyjście i zamyka plik.
- 8.2.3 **(r,!)** Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku tekstowego i wypisuje na standardowym wyjściu zawartość pliku z pominięciem białych znaków.
Napisz dwie wersje funkcji dla znaków typów `char` i `wchar_t`.
- 8.2.4 **(r)** Napisz funkcję, która dostaje jako argumenty ścieżkę dostępu do pliku tekstowego oraz znak `c` i zwraca jako wartość liczbę wystąpień znaku `c` w podanym w argumencie pliku.
Napisz dwie wersje funkcji dla znaków typów `char` i `wchar_t`.
- 8.2.5 Napisz funkcję, która dostaje w argumencie ścieżkę dostępu do pliku tekstowego i wypisuje na standardowym wyjściu statystyki występowania w pliku poszczególnych znaków (zakładamy, że znaki są typu `char`).
8.2.6 Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku tekstowego i zwraca jako wartość najczęściej występujący w pliku znak (zakładamy, że znaki są typu `char`).
8.2.7 Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku tekstowego zawierającego liczby całkowite oddzielone białymi znakami i zwraca jako wartość sumę znajdujących się w pliku liczb.
8.2.8 Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku tekstowego zawierającego liczby całkowite oddzielone białymi znakami i zwraca jako wartość najmniejszą spośród znajdujących się w pliku liczb.
8.2.9 **(r,!)** Napisz funkcję, która dostaje jako argumenty ścieżki dostępu do dwóch plików tekstowych i zwraca jako wartość 1, jeżeli podane pliki mają taką samą zawartość oraz 0 w przeciwnym wypadku.
Napisz dwie wersje funkcji dla znaków typu `char` i `wchar_t`.
- 8.2.10 **(r)** Napisz funkcję, która dostaje jako argumenty ścieżki dostępu do dwóch plików tekstowych i zwraca jako wartość 1, jeżeli podane pliki mają taką samą zawartość z dokładnością do białych znaków oraz 0 w przeciwnym wypadku.
Napisz dwie wersje funkcji dla znaków typu `char` i `wchar_t`.
- 8.2.11 **(r)** Napisz funkcję, która dostaje jako argument ścieżkę dostępu do pliku, otwiera plik do tekstowego pisania z kursorem ustawionym na końcu pliku i zwraca jako wartość deskryptor świeżo otwartego pliku (w wersji dla języka C++ funkcja powinna zwrócić wskaźnik do obiektu klasy `fstream`).
8.2.12 **(r)** Napisz funkcję, która dostaje jako argument deskryptor do pliku tekstowego otwartego do pisania (w wersji dla języka C++ referencję do obiektu klasy `fstream`) oraz liczbę `n`, wczytuje ze standardowego

wejścia n wersów tekstu, zapisuje do pliku wczytany tekst i zamyka plik.

Napisz dwie wersje funkcji dla znaków typów `char` i `wchar_t`.

- 8.2.13 **(r)** Napisz funkcję, która dostaje jako argumenty deskryptory dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego pliku.
- 8.2.14 **(r)** Napisz funkcję, która dostaje jako argumenty ścieżki dostępu do dwóch plików (w wersji dla języka C++ referencje do obiektów klasy `fstream`) i przepisuje zawartość pierwszego pliku do drugiego pliku (stara zawartość drugiego pliku ma zostać skasowana).
- 8.2.15 Napisz funkcję, która dostaje jako argumenty ścieżki dostępu do dwóch plików i dopisuje zawartość pierwszego pliku na koniec drugiego pliku.
- 8.2.16 Napisz funkcję, która dostaje w argumentach jednowymiarową tablicę liczb całkowitych `tab`, jej rozmiar oraz ścieżkę dostępu do pliku tekstowego, i dopisuje w kolejnych wierszach na końcu otrzymanego pliku wartości kolejnych elementów tablicy `tab`.
- 8.2.17 **(*,r,!)** Napisz funkcję, która dostaje jako argumenty nazwę pliku, dwuwymiarową tablicę tablic o elementach typu `int` oraz wymiary tablicy i zapisuje binarnie zawartość tablicy do podanego pliku.
- 8.2.18 **(*,r,!)** Napisz funkcję, która dostaje jako argumenty nazwę pliku, dwuwymiarową tablicę tablic o elementach typu `int` oraz wymiary tablicy i wczytuje binarnie zawartość pliku do tablicy. Napisz funkcję tak, aby była „kompatybilna” z funkcją z zadania 8.2.17.
- 8.2.19 **(*)** Napisz funkcję, która dostaje jako argumenty nazwę pliku, dwuwymiarową tablicę tablic o elementach typu `int` oraz wymiary tablicy i zapisuje binarnie zawartość tablicy oraz jej wymiary do podanego pliku.
- 8.2.20 **(*)** Napisz funkcję, która dostaje jako argumenty nazwę pliku, wczytuje zawartość pliku do nowo utworzonej dwuwymiarowej tablicy tablic. Wymiary tablicy powinny być podane w pliku. Napisz funkcję tak, aby była „kompatybilna” z funkcją z zadania 8.2.19.

ROZDZIAŁ 9

INSTRUKCJE PREPROCESORA, APLIKACJE WIELOPLIKOWE, MAKEFILE.

9.1.	Wprowadzenie	52
9.2.	Makra	52
9.3.	Aplikacje wieloplikowe, makefile	53

9.1. Wprowadzenie

Przed właściwym procesem komplikacji, czyli tłumaczeniem programu napisanego w języku wysokiego poziomu na kod maszynowy, w przypadku języków C i C++ kod programu przetwarzany jest przez preprocesor. W początkach języka C preprocesor odgrywał bardzo dużą rolę. Jednak w wyniku rozwoju tego języka, w tym dodania do niego m. in. takich elementów jak stałe, czy funkcje inline, znaczenia preprocesora zmalało. Nie bez znaczenia był też rozwój debuggerów, które znacznie ułatwiały szukanie błędów w programach, co wcześniej było jedną z dziedzin, w której wykorzystywano dyrektywy preprocesora. Obecnie dyrektywy preprocesora wykorzystuje się niemal wyłącznie dołączania bibliotek i tworzenia aplikacji wieloplikowych.

9.2. Makra

W zadaniach w tym podrozdziale należy napisać makrodefinicje. Makrodefinicje, nazywane krócej makrami, tworzy się przy pomocy dyrektywy `#define`.

- 9.2.1 **(r,!)** Napisz makro, które dostaje trzy argumenty i zwraca ich sumę.
- 9.2.2 Napisz makro, które dla trzech otrzymanych w argumentach liczb zwraca ich średnią.
- 9.2.3 **(r)** Napisz makro, które dostaje jako argumenty dwie liczby całkowite i wypisuje na standardowym wyjściu większą z nich.
- 9.2.4 **(r)** Napisz jednoargumentowe makro, które zwraca wartość 1 jeżeli argumentem jest liczba parzysta i 0 jeżeli argument jest nieparzysty.
- 9.2.5 Napisz makro, które dostaje dwa argumenty i zwraca większy z nich (zakładamy, że otrzymane wartości są porównywalne).
- 9.2.6 **(r,!)** Napisz makro, które dostaje trzy liczby całkowite jako argumenty i wypisuje na standardowym wyjściu największą z otrzymanych wartości.
- 9.2.7 **(r)** Napisz makro, które dostaje trzy argumenty i zwraca największą z otrzymanych wartości (zakładamy, że otrzymane wartości są porównywalne).
- 9.2.8 **(r)** Napisz makro o dwóch argumentach `x` i `n`, które działa jak pętla `for+`, w której zmienna `x` przebiega wartości od 0 do `n-1`.
- 9.2.9 Napisz makro o dwóch argumentach `x` i `n`, które działa jak pętla `for`, w której zmienna `x` przebiega wartości od `n` do 0.
- 9.2.10 **(r)** Napisz makro, które nadaje wartość 0 podanej w argumencie zmiennej.

9.2.11 Napisz makro, które zwiększa o 2 podaną w argumencie zmienną liczbową.

9.3. Aplikacje wieloplikowe, makefile

- 9.3.1 (r) Napisz program, który oblicza miejsca zerowe wielomianu drugiego stopnia o współczynnikach wczytanych ze standardowego wejścia. W programie wykorzystaj samodzielnie napisaną funkcję liczącą pierwiastek z liczby dodatniej. Funkcję liczącą pierwiastek umieść w oddzielnym pliku.
- 9.3.2 Napisz program, który oblicza wartość wielomianu jednej zmiennej o współczynnikach podanych przez użytkownika w punkcie podanym przez niego. W programie wykorzystaj samodzielnie napisaną funkcję liczącą potęgę. Funkcję liczącą potęgę umieść w oddzielnym pliku.
- 9.3.3 (r,!) Napisz program, który oblicza miejsca zerowe wielomianu drugiego stopnia o współczynnikach wczytanych ze standardowego wejścia. W programie wykorzystaj samodzielnie napisaną funkcję liczącą pierwiastek z liczby dodatniej. Program podziel na trzy pliki: plik zawierający główny program, plik zawierający funkcję pierwiastkującą, oraz plik z nagłówkiem funkcji pierwiastkującej.
- 9.3.4 (r,!) Napisz plik makefile pozwalający na komplikację programu z zadania 9.3.3.
- 9.3.5 Napisz program, który oblicza wartość wielomianu jednej zmiennej o współczynnikach podanych przez użytkownika w punkcie podanym przez użytkownika. W programie wykorzystaj samodzielnie napisaną funkcję liczącą potęgę. Program podziel na trzy pliki: plik zawierający główny program, plik zawierający funkcję potęgującą oraz plik z nagłówkiem funkcji potęgującej.
- 9.3.6 Napisz plik makefile pozwalający na komplikację programu z zadania 9.3.5.
- 9.3.7 (r) Napisz program, który wczytuje ze standardowego wejścia ciąg liczb zespolonych i wypisuje na standardowym wyjściu sumę wczytyanych liczb. Program powinien składać się z trzech części:
- programu głównego,
 - biblioteki zawierającej definicje typu `zespolone` oraz funkcji do wczytywania i wypisywania elementów tego typu,
 - biblioteki zawierającej funkcje do dodawania i mnożenia liczb zespolonych.
- Biblioteki powinny się składać z dwóch plików: pliku nagłówkowego oraz pliku z definicjami funkcji.

9.3.8 **(r)** Napisz plik makefile pozwalający na komplikację programu z zadania 9.3.7.

9.3.9 Napisz program, który wczytuje ze standardowego wejścia listę pracowników (imię, nazwisko, wiek), zapisuje ich w tablicy rekordów typu **osoba**, i wypisuje średnią wieku wczytanych pracowników. Program powinien składać się z trzech części:

- programu głównego,
- biblioteki zawierającej definicję typu **osoba** oraz funkcje do wczytywania i wypisywania elementów tego typu,
- biblioteki zawierającej funkcje statystyczne (średnia wieku, minimalny oraz maksymalny wiek) pracujące na tablicach o elementach typu **osoba**. Funkcje zawarte w tej bibliotece powinny otrzymywać w argumentach tablicę elementów typu **osoba** oraz jej rozmiar i zwracać wynik działania jako swoją wartość.

Biblioteki powinny się składać z dwóch plików: pliku nagłówkowego oraz pliku z definicjami funkcji.

9.3.10 Napisz plik makefile pozwalający na komplikację programu z zadania 9.3.9.

9.3.11 **(r, !)** Napisz program, który wczytuje ze standardowego wejścia listę pracowników (imię, nazwisko, wiek), zapisuje ich w tablicy rekordów typu **osoba**, i wypisuje średnią wieku wczytanych pracowników. Program powinien składać się z trzech części:

- programu głównego,
- biblioteki **dane** zawierającej definicję typu **osoba** oraz funkcje do wczytywania i wypisywania elementów tego typu. Wczytane dane osobowe oraz liczba przechowywanych rekordów powinny być przechowywane przez zmienne zadeklarowane w tej bibliotece,
- biblioteki zawierającej funkcje statystyczne (średnia wieku, minimalny oraz maksymalny wiek) pracujące na tablicach o elementach typu **osoba**. Funkcje zawarte w tej bibliotece powinny korzystać z danych przechowywanych przez zmienne zadeklarowane w bibliotece **dane**.

Biblioteki powinny się składać z dwóch plików: pliku nagłówkowego oraz pliku z definicjami funkcji.

9.3.12 **(r)** Napisz plik makefile pozwalający na komplikację programu z zadania 9.3.11.

9.3.13 **(r, !)** Napisz bibliotekę służącą do przechowywania danych osobowych (imię, nazwisko, wiek). Biblioteka powinna zawierać definicję typu **osoba** oraz udostępniać następujące funkcje:

- **wczytaj** – funkcja wczytująca dane jednej osoby ze standardowego wejścia. Dane powinny być wczytywane do tablicy o elementach typu **osoba**.

- **wypisz** – funkcja wypisująca na standardowym wyjściu wszystkie wczytane dane osobowe,
- **ile** – funkcja zwracająca jako wartość liczbę osób, których dane osobowe zostały wczytane.
- **wczytana** – funkcja udostępniająca wczytane dane osobowe. Funkcja dla podanej w argumencie nieujemnej liczby całkowitej **n** powinna zwrócić element typu **osoba** przechowywany pod indeksem **n**.

Dostęp do danych przechowywanych w bibliotece powinien być możliwy wyłącznie za pośrednictwem zdefiniowanych w bibliotece funkcji. Zdefiniowane w bibliotece zmienne powinny być widoczne wyłącznie w tej bibliotece.

9.3.14 Napisz bibliotekę **kalkulator**. Biblioteka ta powinna udostępniać następujące funkcje:

- **wczytaj** – funkcja wczytująca podaną w argumencie wartość do kalkulatora,
- **dodaj** – funkcja dodająca liczbę podaną w argumencie do wartości przechowywanej w kalkulatorze; wynik działania powinien zostać zapisany w kalkulatorze i zwrócony przez funkcję,
- **odejmij** – funkcja odejmująca liczbę podaną w argumencie od wartości przechowywanej w kalkulatorze; wynik działania powinien zostać zapisany w kalkulatorze i zwrócony przez funkcję,
- **pomnoż** – funkcja mnożąca liczbę podaną w argumencie przez wartość przechowywaną w kalkulatorze;; wynik działania powinien zostać zapisany w kalkulatorze i zwrócony przez funkcję,
- **odejmij** – funkcja dzieląca wartość przechowywaną w kalkulatorze przez liczbę podaną w argumencie funkcji; wynik działania powinien zostać zapisany w kalkulatorze i zwrócony przez funkcję.

Zdefiniowane w bibliotece **kalkulator** zmienne powinny być widoczne wyłącznie w tej bibliotece.

9.3.15 (**r**,**!**,**róż**) Napisz bibliotekę **koło** zawierającą:

- jednoargumentową funkcję **pole** zwracającą pole koła o podanym w argumencie promieniu,
- jednoargumentową funkcję **obwód** zwracającą obwód koła o podanym w argumencie promieniu,
- stałą **pi** przechowującą wartość liczby π . Stała ta powinna być dostępna w programach używających biblioteki **koło**.

ROZDZIAŁ 10

ROZWIAZANIA I WSKAZÓWKI

10.1.	Rozwiązania do zadań z rozdziału 1.2	58
10.2.	Rozwiązania do zadań z rozdziału 1.3	62
10.3.	Rozwiązania do zadań z rozdziału 1.4	65
10.4.	Rozwiązania do zadań z rozdziału 2.2	70
10.5.	Rozwiązania do zadań z rozdziału 3.2	77
10.6.	Rozwiązania do zadań z rozdziału 4.2	79
10.7.	Rozwiązania do zadań z rozdziału 5.2	86
10.8.	Rozwiązania do zadań z rozdziału 6.2	97
10.9.	Rozwiązania do zadań z rozdziału 7.2	105
10.10.	Rozwiązania do zadań z rozdziału 7.3	113
10.11.	Rozwiązania do zadań z rozdziału 8.2	129
10.12.	Rozwiązania do zadań z rozdziału 9.2	138
10.13.	Rozwiązania do zadań z rozdziału 9.3	139

10.1. Rozwiązania do zadań z rozdziału 1.2

Zadanie 1.2.1

Listing 10.1. Rozwiązanie zad. 1.2.1 w języku C

```
#include <stdio.h>
2
int main(){
4    int liczba;
    printf("Hello\World");
6    return 0;
}
```

Standardowe wejście i standardowe wyjście to nazwy predefiniowanych w języku C strumieni. To, co one oznaczają, nie zależy od programu napisanego w C, ale od konfiguracji systemu, w którym uruchamiany jest skompilowany program. Zazwyczaj standardowe wejście oznacza klawiaturę, a standardowe wyjście ekran.

Listing 10.2. Rozwiązanie zad. 1.2.1 w języku C++

```
#include <iostream>

int main(){
    int liczba;
    std::cout<<"Hello\World";
    return 0;
}
```

Aby w powyższym rozwiązaniu uniknąć używania przedrostka `std::`, można użyć polecenia `using namespace std;`. Wtedy rozwiązanie wygląda następująco:

Listing 10.3. Rozwiązanie zad. 1.2.1 w języku C++

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hello\World";
    return 0;
}
```

Zadanie 1.2.4

Listing 10.4. Rozwiązanie zad. 1.2.4 w języku C

```
1 #include <stdio.h>
3 int main(){
4     int liczba;
5     scanf("%d", &liczba);
6     printf("%d", liczba);
7     return 0;
}
```

W funkcji `scanf` zmienne, do których wczytywane są wartości, są poprzedzone znakiem „&”. Używając funkcji `scanf` należy uważać, aby nie zapominać o wstawianiu „&”, gdyż jest to błąd niewykrywany przez kompilatory, który ujawnia się dopiero podczas działania programu. Wyjątkiem, w którym nie stosuje się znaku „&” przed nazwą wczytywanej zmiennej jest wczytywanie napisów.

Listing 10.5. Rozwiązanie zad. 1.2.4 w języku C++

```
#include <iostream>
using namespace std;

int main(){
    int liczba;
    cin>>liczba;
    cout<<liczba;
    return 0;
}
```

Zadanie 1.2.5

Listing 10.6. Rozwiązanie zad. 1.2.5 w języku C

```
#include <stdio.h>

int main(){
    double liczba;
    scanf("%lf", &liczba);
    printf("%f", liczba);
    return 0;
}
```

Warto zapamiętać, że w funkcjach `scanf` i `printf` innych flag używa się przy wczytywaniu i wypisywaniu wartości typu `double` (odpowiednio „%lf” i „%f”).

Listing 10.7. Rozwiązanie zad. 1.2.5 w języku C++

```
#include <iostream>

int main(){
    double liczba;
    std::cin>>liczba;
    std::cout<<liczba;
    return 0;
}
```

Jak widać powyższe rozwiązanie różni się od rozwiązania zadania 1.2.4 jedynie typem zmiennej `liczba`. Operatory „`<<`” i „`>>`” w pewnym sensie rozpoznają typ zmiennej `liczba` i w zależności od niego wczytują i wypisują wartość zmiennej `liczba` w odpowiedni sposób.

Zadanie 1.2.8

Listing 10.8. Rozwiązanie zad. 1.2.8 w języku C

```
#include <stdio.h>
2
int main(){
4   int l1, l2, l3;
5   scanf("%d", &l1);
6   scanf("%d", &l2);
7   scanf("%d", &l3);
8   printf("%f", (double)(l1 + l2 + l3) / 3);
9   return 0;
10 }
```

Znaczna część studentów na kolokwium linię 8 powyższego programu napisałaby w następujący sposób `printf("%f", (l1+l2+l3)/3);` co byłoby błędem, gdyż dla $l1=0$, $l2=1$ i $l3=1$ na ekranie wyświetlona zostałaby liczba 0, a nie 0.666667. Operatory arytmetyczne w językach C i C++ zwracają wynik takiego samego typu jak argumenty, a gdy argumenty są różnego typu, to typ wyniku jest „najpojemniejszym” spośród typów argumentów. W powyższym programie wartość pierwszego argumentu dzielenia została zrzutowana na typ —`double`—. Dzięki temu argumentami dzielenia nie są dwie wartości typu `int`, a jedna wartość typu `double` i druga wartość typu `int`. Co za tym idzie, wynik dzielenia wykonanego w programie jest typu `double`, a więc może on wyrazić liczbę 0.6(6) (a raczej najbliższa jej wartość możliwa do zapisania w typie `double`). Innym poprawnym, choć mniej eleganckim, rozwiązaniem jest napisanie linii 8 programu 10.8 w następujący sposób `printf("%f", (l1*1.0+l2+l3)/3);`.

Rozwiązań w języku C++ jest analogiczne.

Zadanie 1.2.9

Listing 10.9. Rozwiązań zad. 1.2.9 w języku C

```
1 #include <stdio.h>
2 #include <math.h>

4 int main(){
    double x;
6    scanf("%lf", &x);
    printf("%f", sqrt(x));
8    return 0;
}
```

Aby powyższy program skompilować przy użyciu gcc należy użyć parametru „-lm”. Jest tak, gdyż w programie użyta została funkcja `sqrt` z biblioteki `math`. Podobnie należy postępować także przy komplikacji za pomocą gcc programów używających innych operacji z biblioteki `math`. Problem taki nie występuje w przypadku g++, a więc poniższe rozwiązanie dla C++ kompliuje się poprawnie bez dodatkowego parametru:

Listing 10.10. Rozwiązań zad. 1.2.9 w języku C

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;

5 int main(){
    double x;
7    cin>>x;
    cout<<sqrt(x);
9    return 0;
}
```

Zadanie 1.2.11 Rozwiązań dla języka C:

Listing 10.11. Rozwiązań zad. 1.2.8 w języku C

```
1 #include <stdio.h>
2
3 int main(){
4     float f;
5     scanf("%f", &f);
6     printf("%.2f", f);
8 }
```

Rozwiązanie dla języka C++:

Listing 10.12. Rozwiązanie zad. 1.2.8 w języku C

```

#include<iostream>
using namespace std;

int main(){
    float f;
    cin>>f;
    cout.precision(2);
    cout.setf( ios::fixed, ios::floatfield );
    cout<<fixed<<f;
}

```

10.2. Rozwiązania do zadań z rozdziału 1.3

Zadanie 1.3.1

Listing 10.13. Rozwiązanie zadania 1.3.1 w języku C

```

#include <stdio.h>

int main(){
    int liczba;
    printf("Podaj liczbę całkowitą:");
    scanf("%d", &liczba);
    if (liczba < 0)
        liczba *= -1;
    printf(" | liczba | = %d", liczba);
    return 0;
}

```

To samo zadanie można rozwiązać używając operatora warunkowego:

Listing 10.14. Rozwiązanie zadania 1.3.1 w języku C

```

#include <stdio.h>

int main(){
    int liczba;
    printf("Podaj liczbę całkowitą:");
    scanf("%d", &liczba);
    printf(" | liczba | = %d", (liczba>=0)?liczba:(-1)*liczba);
    return 0;
}

```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 1.3.5

Listing 10.15. Rozwiązanie zadania 1.3.5 w języku C

```
#include <stdio.h>
#include <math.h>

int main(){
    int w;
    double bok1, bok2, bok3, h, p, s;
    printf("Witam. Obliczam pole trojkata. Wpisz:\n");
    printf("1-jesli chcesz podac dl. podstawy i wys.\n");
    printf("2-jesli chcesz podac dl. trzech bokow\n");
    scanf("%d", &w);
    if (w == 1){
        printf("Podaj dlugosc podstawy trojkata.");
        scanf("%lf", &bok1);
        printf("Podaj wysokosc trojkata.");
        scanf("%lf", &h);
        p=bok1*h/2;
    }
    else {
        printf("Podaj dlugosc pierwszego boku trojkata:");
        scanf("%lf", &bok1);
        printf("Podaj dlugosc drugiego boku trojkata:");
        scanf("%lf", &bok2);
        printf("Podaj dlugosc trzeciego boku trojkata:");
        scanf("%lf", &bok3);
        s=(bok1 + bok2 + bok3)/2;
        p=sqrt(s * (s - bok1) * (s - bok2) * (s - bok3));
    }
    printf("Pole trojkata o podanych wymiarach wynosi %f", p);
    return 0;
}
```

Rozwiązanie w języku C++:

Listing 10.16. Rozwiązanie zadania 1.3.5 w języku C++

```
#include <iostream>
#include <cmath>
using namespace std;

int main(){
    int w;
    double bok1, bok2, bok3, h, p, s;
    cout<<"Witam. Obliczam pole trojkata. Wpisz:"<<endl;
    cout<<"1-jesli chcesz podac dl. podstawy i wys."<<endl;
    cout<<"2-jesli chcesz podac dl. i trzech bokow"<<endl;
```

```

    cin>>w;
    if (w == 1){
        cout<<"Podaj_dlugosc_podstawy_trojkata:_";
        cin>>bok1;
        cout<<"Podaj_wysokosc_trojkata:_";
        cin>>h;
        p=bok1*h/2;
    }
    else {
        cout<<"Podaj_dlugosc_pierwszego_boku_trojkata:_";
        cin>>bok1;
        cout<<"Podaj_dlugosc_drugiego_boku_trojkata:_";
        cin>>bok2;
        cout<<"Podaj_dlugosc_trzeciego_boku_trojkata:_";
        cin>>bok3;
        s=(bok1 + bok2 + bok3)/2;
        p=sqrt(s * (s - bok1) * (s - bok2) * (s - bok3));
    }
    cout<<"Pole_trojkata_o_podanych_wymiarach_wynosi_";
    return 0;
}

```

Zadanie 1.3.8

Listing 10.17. Rozwiązanie zadania 1.3.8 w języku C

```

#include <stdio.h>

int main(){
    int i;
    double a,b,h,p;
    printf("Pole_jakiej_figury_chcesz_policzyc?\n");
    printf("1_kwadrat\n");
    printf("2_prostokat\n");
    printf("3_trojkat\n");
    scanf("%d",&i);
    switch (i){
        case 1: printf("Podaj_dl_boku_kwadratu");
                  scanf("%lf",&a);
                  p=a*a;
                  break;
        case 2: printf("Podaj_dl_bokow_prostokata");
                  scanf("%lf %lf",&a,&b);
                  p=a*b;
                  break;
        case 3: printf("Podaj_dl_podstawy_i_wys_trojkata");
                  scanf("%lf %lf",&a,&h);
                  p=0.5*a*h;
    }
    printf("Pole_figury_o_podanych_wymiarach_wynosi_%f\n",p);
    return 0;
}

```

```
}
```

W powyższym programie zamiast instrukcji `switch` można użyć kilku instrukcji `if`, jednak dzięki użyciu instrukcji `switch` program jest bardziej czytelny.

Wersja dla języka C++ jest analogiczna.

10.3. Rozwiązania do zadań z rozdziału 1.4

Zadanie 1.4.1

Listing 10.18. Rozwiązanie zadania 1.4.1 w języku C

```
#include <stdio.h>

int main(){
    int n,m;
    printf("Podaj liczbę całkowitą n: ");
    scanf("%d", &n);
    printf("Podaj liczbę całkowitą m: ");
    scanf("%d", &m);
    for(int i = n ; i < m ; i += n)
        printf("%d\n", i);
    return 0;
}
```

W powyższym programie zmienna `i` została zadeklarowana jednocześnie ze swoją inicjacją w pętli `for`. Taka możliwość w języku C pojawiła się w standardzie C99. W powyższym programie zmiennej `i` nie można używać poza pętlą `for`. W trakcie komplikacji program 10.18 może wymagać dodatkowego parametru mówiącego kompilatorowi, że program jest zgodny ze standardem C99 języka C (w gcc jest to parametr „`-std=c99`”).

Rozwiązanie w języku C++:

Listing 10.19. Rozwiązanie zadania 1.4.1 w języku C++

```
#include <iostream>
using namespace std;

int main(){
    int n,m;
    cout<<"Podaj liczbę całkowitą n: ";
    cin>>n;
    cout<<"Podaj liczbę całkowitą m: ";
    cin>>m;
    for(int i = n ; i < m ; i += n)
        cout<<i<<endl;
```

```
    return 0;
}
```

Zadanie 1.4.4

Listing 10.20. Rozwiązanie zadania 1.4.4 w języku C

```
#include <stdio.h>

int main(){
    int n, i, silnia=1;
    printf("Podaj liczbę całkowitą n:");
    scanf("%d", &n);
    for(i=2; i<=n; i++)
        silnia *= i;
    printf("Silnia z %d wynosi %d\n", n, silnia);
    return 0;
}
```

W programie 10.20 należy zwrócić uwagę na kilka rzeczy:

- Zmienna **silnia** jest inicjowana od razu przy swojej deklaracji.
- Dla $n = 0$ oraz $n = 1$ pętla **for** nie wykona się ani razu i program zwróci początkową wartość zmiennej **silnia**, a więc 1. Jest to oczywiście poprawna odpowiedź dla tych przypadków.
- Warto przyjrzeć się sposobowi wyliczania silni dla $n > 1$. Kluczowa jest tu linia **silnia *= i;**, którą inaczej można zapisać jako **silnia = silnia * i;**. Jest to typowa programistyczna sztuczka. W i -tym obrocie pętli obliczana jest wartość $i!$ (program zapisuje ją do zmiennej **silnia**), korzystając ze wzoru $i! = (i - 1)! * i$, gdzie $(i - 1)!$ to stara wartość zmiennej **silnia**. W ten sposób po n obrotach pętli **for** w zmiennej **silnia** zapisana jest wartość $n!$.

Czytelnik, któremu zaprezentowane powyżej rozwiązanie wydaje się niejasne powinien spróbować rozwiązać kilka kolejnych zadań. Można je rozwiązać w bardzo podobny sposób.

Rozwiązań zadania w języku C++ jest podobne.

Zadanie 1.4.8

Listing 10.21. Rozwiązanie zadania 1.4.8 w języku C

```
#include <stdio.h>

int main(){
    int n, i, fib1=1, fib2=1, pom;
    printf("Podaj liczbę całkowitą n:");
    scanf("%d", &n);
```

```

for( i=2; i<=n; i++){
    pom= fib1;
    fib1 = fib2 + fib1;
    fib2 = pom;
}
printf("Elem. ciagu Fib. o indeksie %d to %d\n", n, fib1);
return 0;
}

```

W powyższym rozwiązaniu założono, że indeks pierwszego elementu ciągu Fibonacciego to 0.

Rozwiązanie w języku C++ jest podobne.

Zadanie 1.4.9 Tym razem przedstawiono rozwiązanie zadania w języku C++. Rozwiązanie w języku C różni się od zaprezentowanego jedynie operacjami wejścia/wyjścia.

Zadanie można rozwiązać poprzez przeszukanie wszystkich liczb dodatnich mniejszych równych zarówno od n , jak i od m :

Listing 10.22. Rozwiązanie zadania 1.4.9 w języku C++

```

#include <iostream>
using namespace std;

int main(){
    int n, m, nwd=1, max;
    cout<<"Podaj liczbę całkowitą n:" ;
    cin>>n;
    cout<<"Podaj liczbę całkowitą m:" ;
    cin>>m;
    max=(n>m)?n:m;
    for (int i=2; i<=max; i++)
        if ((n % i == 0) && (m % i == 0))
            nwd=i;
    cout<<"NWD liczb "<<n<<" i "<<m<<" wynosi "<<nwd<<endl;
    return 0;
}

```

Ponieważ powyższy program przeszukuje zbiór potencjalnych rozwiązań od dołu, to ostatni znaleziony wspólny dzielnik n i m będzie tym największym. W programie użyto operatora „%” zwracającego resztę z dzielenia pierwszego argumentu przez drugi.

Innym sposobem rozwiązania zadania jest zaimplementowanie algorytmu Euklidesa szukania NWD:

Listing 10.23. Rozwiązanie zadania 1.4.9 w języku C++

```
#include <iostream>
using namespace std;

int main(){
    int n, m, pom1, pom2;
    cout<<"Podaj liczbę całkowitą n:" ;
    cin>>n;
    cout<<"Podaj liczbę całkowitą m:" ;
    cin>>m;
    pom1=n;
    pom2=m;
    while(pom1*pom2!=0)
        if (pom1>pom2)
            pom1=pom1%pom2;
        else
            pom2=pom2%pom1;
    cout<<"NWD liczby "<<n<<" i "<<m<<" wynosi " ;
    if (pom1!=0)
        cout<<pom1<<endl;
    else
        cout<<pom2<<endl;
    return 0;
}
```

Drugi z zaprezentowanych programów dla dużych liczb n i m działa znacznie szybciej niż pierwszy.

Zadanie 1.4.10 Także to zadanie można rozwiązać poprzez przeszukiwanie wszystkich potencjalnych rozwiązań.

Listing 10.24. Rozwiązanie zadania 1.4.10 w języku C

```
#include <stdio.h>

int main(){
    int x, i, pierw=0;
    printf("Podaj liczbę całkowitą x:");
    scanf("%d", &x);
    for(i=1; i<=x; i++)
        if (i * i <= x)
            pierw=i;
    printf("Pierwszy z %d to w przybliżeniu %d\n", x, pierw);
    return 0;
}
```

Zadanie 1.4.10 można rozwiązać efektywniej. Jedną z możliwości jest zastosowanie algorytmu bisekcji:

Listing 10.25. Rozwiązanie zadania 1.4.10 w języku C

```
#include <stdio.h>

int main(){
    int x, pocz, kon, sr;
    printf("Podaj liczbę całkowitą x:");
    scanf("%d", &x);
    pocz=0;
    kon=x;
    while (kon - pocz > 1){
        sr=(pocz + kon) / 2;
        if (sr * sr <= x)
            pocz=sr;
        else
            kon=sr;
    }
    printf("Pierwiastek z %d to %w przybliżeniu ", x);
    if (x<=1)
        printf("%d\n", kon);
    else
        printf("%d\n", pocz);
    return 0;
}
```

W powyższym programie w każdym obrocie pętli `while` zbiór potencjalnych rozwiązań zmniejsza się o połowę. Przydaje się tu fakt, iż dzielenie liczb całkowitych daje w wyniku liczbę całkowitą.

W języku C++ to zadanie rozwiązuje się w analogiczny sposób.

Zadanie 1.4.12 To zadanie można rozwiązać za pomocą zagnieżdzonych pętli:

Listing 10.26. Rozwiązanie zadania 1.4.12 w języku C

```
#include <stdio.h>

int main(){
    int n, m, pom1, pom2, suma=0, i;
    printf("Podaj liczbę całkowitą n:");
    scanf("%d", &n);
    for( i = 2 ; i < n ; i++ ){
        pom1=n;
        pom2=i;
        while(pom1*pom2!=0)
            if (pom1<pom2)
                pom2=pom2%pom1;
            else
                pom1=pom1%pom2;
        if (( pom1 == 1 )||( pom2 == 1 ))
```

```

        suma += i;
    }
    printf("Wynik obliczenia to: %d\n", suma);
    return 0;
}

```

W języku C++ to zadanie rozwiązuje się w analogiczny sposób.

Zadanie 1.4.13 Pozornie zadanie wymaga zastosowania pętli w pętli. W rzeczywistości jednak tak nie jest:

Listing 10.27. Rozwiązań zadania 1.4.13 w języku C++

```

#include <iostream>
using namespace std;

int main(){
    int n, silnia=1,suma=1;
    cout<<"Podaj liczbę całkowitą n:";
    cin>>n;
    for(int i=1; i<=n; i++){
        silnia *= i;
        suma+=silnia;
    }
    cout<<"0! + 1! + ... + "<<n<<"! = "<<suma<<endl;
    return 0;
}

```

W języku C to zadanie rozwiązuje się w analogiczny sposób.

10.4. Rozwiązań do zadań z rozdziału 2.2

Zadanie 2.2.1

Listing 10.28. Rozwiązań zadania 2.2.1 w języku C

```

#include <stdio.h>

int bezwzgledna(int liczba){
    if (liczba < 0)
        return liczba*(-1);
    else
        return liczba;
}

int main(){

```

```

int n;
printf("Podaj liczbę całkowitą:");
scanf("%d", &n);
printf("|%d|= %d\n", n, bezwzględna(n));
return 0;
}

```

Rozwiązanie w języku C++ jest podobne.

Zadanie 2.2.2

Listing 10.29. Rozwiązanie zadania 2.2.2 w języku C

```

#include <stdio.h>

int silnia(unsigned int liczba){
    int i, sil=1;
    for(i=2;i <= liczba; i++)
        sil*=i;
    return sil;
}

int main(){
    int n;
    printf("Podaj liczbę całkowitą:");
    scanf("%d", &n);
    printf("silnia z %d = %d\n", n, silnia(n));
    return 0;
}

```

W języku C++ to zadanie rozwiązuje się w analogiczny sposób.

Zadanie 2.2.10

Listing 10.30. Rozwiązanie zadania 2.2.10 w języku C

```

#include <stdio.h>

unsigned int NWD(unsigned int p, unsigned int d){
    while(p != d)
        if (p > d)
            p=p-d;
        else
            d=d-p;
    return p;
}

unsigned int suma(unsigned int n){
    int i, sum=0;

```

```

for( i = 1 ; i < n ; i++ ){
    if ( NWD(i,n) == 1 )
        sum += i;
}
return sum;
}

int main(){
    int n;
    printf("Podaj liczbę całkowitą: ");
    scanf("%d",&n);
    printf("Wynik obliczenia: %d\n" ,suma(n));
    return 0;
}

```

Dzięki używaniu funkcji programy zyskują na czytelności. Ponadto, raz zapelmatowane i przetestowane funkcje mogą być traktowane jak „czarne skrzynki”, do których kodu się nie wraca. Dzięki takiemu podejściu programista może skupiać się naraz na niewielkich fragmentach kodu zawartych w pojedynczych funkcjach. Znacznie upraszcza to pisanie programów. Przykładowo, dzięki wydzieleniu funkcji NWD w programie 10.30 napisanie funkcji suma stało się proste.

W językach C i C++ argumenty funkcji przekazywane są przez wartość. Oznacza to, że wewnątrz funkcji pracuje się na „kopiąch” wartości przekazanych w argumentach. Przykładowo, w programie 10.30, zmiana wartości zmiennych p i d wewnątrz funkcji NWD nie pociąga za sobą zmiany wartości zmiennych i i n z funkcji suma.

Rozwiążanie w języku C++ jest analogiczne.

Zadanie 2.2.13 a)

Listing 10.31. Rozwiązanie zadania 2.2.13 a) w języku C

```

int pierw(unsigned int x){
    int pocz=0, kon=x, sr;
    while (kon - pocz > 1){
        sr=(pocz + kon) / 2;
        if (sr * sr <= x)
            pocz=sr ;
        else
            kon=sr ;
    }
    if (x<=1)
        return kon;
    else
        return pocz;
}

```

```

void wypisz(unsigned int n){
    int i ,p;
    for( i = 1 ; i <=pierw(n) ; i++ ){
        p=pierw(n-i*i);
        if ((p!=0)&&(i*i+p*p==n))
            printf("%d%d%d%d\n",i,i,p,p,n);
    }
}

```

W języku C++ to zadanie rozwiązuje się w analogiczny sposób.

Zadanie 2.2.13 b)

Listing 10.32. Rozwiązanie zadania 2.2.13 b) w języku C

```

int pierw(unsigned int x){
    int pocz=0, kon=x, sr;
    while (kon - pocz > 1){
        sr=(pocz + kon) / 2;
        if (sr * sr <= x)
            pocz=sr ;
        else
            kon=sr ;
    }
    if (x<=1)
        return kon;
    else
        return pocz;
}

void wypisz(unsigned int n){
    int i ,p;
    for( i = 1 ; i <= pierw(n); i++ ){
        p=pierw(n-i*i);
        if ((i*i+p*p==n)&&(i<p))
            printf("%d%d%d%d\n",i,i,p,p,n);
    }
}

```

W tym wariantie zadania, aby nie powtarzać tych samych rozkładów, program sprawdza, czy i jest mniejsze od p .

W języku C++ to zadanie rozwiązuje się w analogiczny sposób.

Zadanie 2.2.17

Listing 10.33. Rozwiązanie zadania 2.2.17 w języku C

```
void zlicz(){
    static unsigned int liczba=0;
    liczba++;
    printf("Funkcja zostala wywolana %d razy\n", liczba );
}
```

W powyższym rozwiązaniu istotne jest słowo kluczowe **static**.

Rozwiązanie dla języka C++ jest analogiczne.

Zadanie 2.2.20 Rozwiązanie tego zadania dla języków C i C++ niczym się nie różni.

Listing 10.34. Rozwiązanie zadania 2.2.20 w języku C/C++

```
unsigned int silnia(unsigned int n){
    if (n <=1 )
        return 1;
    else
        return silnia(n-1)*n;
}
```

Stosując funkcje rekurencyjne w wielu przypadkach można uzyskać elegancki i prosty zapis algorytmu. Niestety szukanie błędów w funkcjach rekurencyjnych nie należy do rzeczy prostych. Dlatego przy pisaniu takich funkcji trzeba szczególnie uważać. W szczególności trzeba pamiętać o tym, że funkcja musi posiadać **warunek stopu**, czyli warunek przy którym oblicza swoją wartość wprost (już się dalej rekurencyjnie nie wywołuje) oraz o tym, żeby kolejne rekurencyjne wywołania funkcji prowadziły do spełnienia warunku stopu. Kolejnych kilka zadań ma sprawdzić umiejętność implementowania prostych funkcji rekurencyjnych.

Zadanie 2.2.23 Ciąg Fibonacciego wydaje się być stworzony do tego, by jego elementy generować przy pomocy funkcji rekurencyjnej:

Listing 10.35. Rozwiązanie zadania 2.2.23 w języku C/C++

```
unsigned int fib(unsigned int n){
    if (n <=1 )
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

Po analizie złożoności obliczeniowej powyższej funkcji okazuje się jednak, że dużo bardziej efektywna jest iteracyjne generowanie elementów ciągu Fibonacciego. Aby się o tym przekonać, wystarczy uruchomić powyższą funkcję oraz program 10.21 dla odpowiednio dużych danych. Słaba efektywność rekurencyjnej implementacji wynika z tego, że wielokrotnie liczy ona wartości tych samych elementów ciągu Fibonacciego.

Zadanie 2.2.26

Listing 10.36. Rozwiążanie zadania 2.2.26 w języku C/C++

```
unsigned int ciag(unsigned int n){  
    if (n <= 2 )  
        return 1;  
    else  
        switch (n%3){  
            case 0: return ciag(n-1) + ciag(n-2);  
            case 1: return 5 * ciag(n-1) + 4;  
            case 2: return ciag(n-1);  
        }  
}
```

W funkcji `ciag` przedstawionej w Listingu 10.36 nie użyto instrukcji `break` w bloku instrukcji `switch`. Wynika to z tego, że instrukcja `return` kończy wykonywanie funkcji (a więc także instrukcji wyboru `switch`) i instrukcja `break` nie jest już potrzebna.

Zadanie 2.2.27

Listing 10.37. Rozwiążanie zadania 2.2.27 w języku C/C++

```
unsigned int f(unsigned int n, unsigned int m){  
    if (n == 0)  
        return m;  
    if (m == 0)  
        return n;  
    return f(n-1,m) + f(n,m-1) + f(n-1, m-1);  
}
```

W funkcji `f` przedstawionej w Listingu 10.37 nie użyto instrukcji `else`. Wynika to z tego, że instrukcja `return` kończy wykonywanie funkcji. Jeżeli więc `return` znajduje się w bloku instrukcji polecenia `if`, to wszystko, co jest po tym bloku instrukcji, zostanie wykonane tylko w przypadku nie spełnienia warunku z `if-a` (a więc tak, jakby po `if` była instrukcja `else`). Pominięcie instrukcji `else` skraca kod funkcji, ale może także zmniejszyć jej czytelność.

Zadanie 2.2.29Listing 10.38. Rozwiązanie zadania 2.2.29 w języku C/C++

```
unsigned int NWD(unsigned int n, unsigned int m){
    if (n == m)
        return m;
    if (n > m)
        return NWD(n%m, m);
    else
        return NWD(m%n, n);
}
```

W powyższej funkcji usunięcie instrukcji `else` nie zmieni działania funkcji.

Zadanie 2.2.30Listing 10.39. Rozwiązanie zadania 2.2.30 w języku C++

```
unsigned int pot(unsigned int n, unsigned int m = 2){
    int p=1;
    if (n == 0)
        return 0;
    for (int i=1; i <= m; i++)
        p*=n;
    return p;
}
```

Zadanie 2.2.36Listing 10.40. Rozwiązanie zadania 2.2.36 w języku C++

```
double pot(double n, unsigned int m){
    double p=1;
    if (n == 0)
        return 0;
    for (int i=1; i <= m; i++)
        p*=n;
    return p;
}

int pot(int n, unsigned int m){
    int p=1;
    if (n == 0)
        return 0;
    for (int i=1; i <= m; i++)
        p*=n;
    return p;
}
```

```
unsigned int pot(unsigned int n, unsigned int m){  
    unsigned int p=1;  
    if (n == 0)  
        return 0;  
    for (int i=1; i <= m; i++)  
        p*=n;  
    return p;  
}
```

10.5. Rozwiązania do zadań z rozdziału 3.2

Zadanie 3.2.1

Listing 10.41. Rozwiązanie zadania 3.2.1 w języku C/C++

```
int mniejsza(int * a, int * b){  
    if (*a<*b)  
        return *a;  
    else  
        return *b;  
}
```

Zadanie to posiada także krótsze rozwiązanie:

Listing 10.42. Rozwiązanie zadania 3.2.1 w języku C/C++

```
int mniejsza(int * a, int * b){  
    return (*a<*b)?*a:*b;  
}
```

Zadanie 3.2.2

Listing 10.43. Rozwiązanie zadania 3.2.2 w języku C/C++

```
int* mniejsza(int * a, int * b){  
    if (*a<*b)  
        return a;  
    else  
        return b;  
}
```

Zadanie 3.2.3

Listing 10.44. Rozwiązanie zadania 3.2.3 w języku C/C++

```
void zamien(int * a, int * b){
```

```

int pom;
pom=*a;
*a=*b;
*b=pom;
}

```

Zadanie 3.2.7Listing 10.45. Rozwiązanie zadania 3.2.7 w języku C++

```

void zamien(int & a, int & b){
    int pom;
    pom=a;
    a=b;
    b=pom;
}

```

Zadanie 3.2.9 Rozwiązanie w języku C:Listing 10.46. Rozwiązanie zadania 3.2.9 w języku C

```

int * alokuj(){
    return malloc(sizeof(int));
}

```

Używając funkcji `malloc` należy pamiętać o dołączenia pliku nagłówkowego `<stdlib.h>`. Jeżeli się tego nie zrobi, kompilator zgłosi ostrzeżenie o próbie przypisania wartości całkowitoliczbowej do zmiennej wskaźnikowej bez rzutowania.

Rozwiązanie w języku C++:

Listing 10.47. Rozwiązanie zadania 3.2.9 w języku C

```

int * alokuj(){
    return new int;
}

```

Zadanie 3.2.11 Rozwiązanie w języku C:Listing 10.48. Rozwiązanie zadania 3.2.11 w języku C

```

int * alokuj(unsigned int n){
    return malloc(n*sizeof(int));
}

```

Rozwiązanie w języku C++:

Listing 10.49. Rozwiązanie zadania 3.2.11 w języku C

```
int * alokuj(unsigned int n){  
    return new int[n];  
}
```

W obu językach alokowanie w pamięci jednowymiarowych tablic dynamicznych odbywa się w identyczny sposób jak w powyższych rozwiązaniach.

Zadanie 3.2.13

Listing 10.50. Rozwiązanie zadania 3.2.13 w języku C/C++

```
double wywolaj(double (* fun)(int arg), int a){  
    return fun(a);  
}
```

Zadanie 3.2.15

Listing 10.51. Rozwiązanie zadania 3.2.15 w języku C/C++

```
void przepisz(int const * a, int * b){  
    *b=*a;  
}
```

Zadanie 3.2.16

Listing 10.52. Rozwiązanie zadania 3.2.15 w języku C/C++

```
void przepisz(int const * a, int * const b){  
    *b=*a;  
}
```

10.6. Rozwiązania do zadań z rozdziału 4.2

Zadanie 4.2.1a

Listing 10.53. Rozwiązanie zadania 4.2.1a w języku C/C++

```
void zeruj (unsigned int n, int * tab){  
    int i;  
    for(i=0;i<n;i++)  
        tab[i]=0;  
}
```

Warto zapamiętać, że w przypadku tablic przekazanych do funkcji w jej argumentach, wewnątrz funkcji pracujemy na „oryginałach” tablic, a nie ich kopiąch. Oznacza to, że zmiana wewnątrz funkcji wartości elementów takich tablic ma konsekwencje także na zewnątrz funkcji. Jest to zachowanie odmienne od argumentów funkcji o typach prostych, w przypadku których pracujemy na kopiąch argumentów. Takie a nie inne zachowanie tablic wynika z faktu, że w C i C++ są one wskaźnikami na bloki pamięci. Skopiowany wskaźnik cały czas wskazuje na ten sam blok pamięci.

Zadanie 4.2.1b

Listing 10.54. Rozwiązań zadania 4.2.1b w języku C/C++

```
void inicjuj (unsigned int n, int * tab){
    int i;
    for(i=0;i<n;i++)
        tab[ i ]=i;
}
```

Zadanie 4.2.2a

Listing 10.55. Rozwiązań zadania 4.2.2a w języku C/C++

```
double srednia (unsigned int n, int * tab){
    int i;
    double sred=0;
    for(i=0;i<n;i++)
        sred+=tab[ i ];
    sred/=n;
    return sred;
}
```

Zastosowane powyżej rozwiązanie jest rozwinięciem metody zastosowanej w rozwiązań zadania 1.4.4.

Zadanie 4.2.3

Listing 10.56. Rozwiązań zadania 4.2.3 w języku C/C++

```
double srednia (unsigned int n,const int * tab){
    int i;
    double sred=0;
    for(i=0;i<n;i++)
        sred+=tab[ i ];
    sred/=n;
    return sred;
}
```

Zadanie 4.2.5

Listing 10.57. Rozwiązanie zadania 4.2.5 w języku C

```
int pierwsza (unsigned int n){
    int i, j, pom;
    bool sito[n];
    for(i=0;i<n;i++)
        sito[i]=true;
    for(i=2;i<n;i++)
        if (sito[i]){
            pom=i;
            for(j=2*i;j<n;j+=i)
                sito[j]=false;
        }
    return pom;
}
```

W pierwszych standardach języka C nie było typu `bool`. Został on wprowadzony dopiero w standardzie C99. Aby móc go użyć, należy dołączyć plik nagłówkowy `stdbool.h`.

Rozwiążanie w języku C++ nie różni się bardzo od powyższego. Istnieje jedna istotna różnica, która uzasadnia prezentację rozwiązań dla obu języków.

Listing 10.58. Rozwiązanie zadania 4.2.5 w języku C++

```
int pierwsza (unsigned int n){
    int pom;
    bool * sito = new bool[n];
    for(int i=0;i<n;i++)
        sito[i]=true;
    for(int i=2;i<n;i++)
        if (sito[i]){
            pom=i;
            for(int j=2*i;j<n;j+=i)
                sito[j]=false;
        }
    delete [] sito;
    return pom;
}
```

Język C++, inaczej niż język C, nie pozwala na tworzenie tablic automatycznych o rozmiarze zadanym przez zmienną. Stąd w tym przypadku pamięć dla tablicy `sito` ręcznie rezerwowana operatorem `new` i ręcznie zwalniana przy użyciu operatora `delete`. Usuwając tablicę należy dodać „`[]`” po operatorze `delete`, a przed wskaźnikiem na tablicę. W języku C++ zamiast

tablicy można użyć szablonu klasy `vector`, jednak w niniejszym zbiorze zadań przedstawiona jest tylko strukturalna części C++.

Także w rozwiązyaniu w języku C można ręcznie zarezerwować pamięć dla tablicy `sito`:

Listing 10.59. Rozwiązań zadania 4.2.5 w języku C

```
int pierwsza (unsigned int n){
    int i, j, pom;
    bool * sito=malloc(n*sizeof(bool));
    for(i=0;i<n; i++)
        sito[ i]=true;
    for(i=2;i<n; i++)
        if (sito[ i]){
            pom=i ;
            for(j=2*i ;j<n; j+=i )
                sito[ j]=false ;
        }
    free( sito );
    return pom;
}
```

Jak można zauważyć w Listingu 10.59 w języku C, inaczej niż ma to miejsce w C++, pamięć zajmowaną przez tablicę zwalnia się dokładnie w taki sam sposób, jak w przypadku typów prostych.

Zadanie 4.2.6a

Listing 10.60. Rozwiązań zadania 4.2.6a w języku C/C++

```
void przepisz (unsigned int n, int * tab1 , int * tab2){
    int i;
    for(i=0;i<n; i++)
        tab2[ i]=tab1[ i ];
}
```

Zadanie 4.2.10a

Listing 10.61. Rozwiązań zadania 4.2.10a w języku C/C++

```
int maksimum(unsigned int n, int * tab){
    int i, max=tab[ 0 ];
    for(i=1;i<n; i++)
        if (tab[ i]>max)
            max=tab[ i ];
    return max;
}
```

Zadanie 4.2.10cListing 10.62. Rozwiązanie zadania 4.2.10c w języku C/C++

```
int maksimum(unsigned int n, int * tab){  
    int i, max=0;  
    for(i=1;i<n;i++)  
        if (tab[i]>tab[max])  
            max=i ;  
    return max;  
}
```

Zadanie 4.2.12aListing 10.63. Rozwiązanie zadania 4.2.12a w języku C/C++

```
void odwroc (unsigned int n, int * tab){  
    int i ,pom;  
    for(i=0;i<n/2; i++){  
        pom=tab[ i];  
        tab[ i]=tab[ n-1-i ];  
        tab[ n-1-i]=pom;  
    }  
}
```

W powyższym rozwiązaniu ważne jest, że zmienna i przebiega wartości mniejsze od $n/2$. Gdyby i przebiegało wartości od 0 do $n-1$, wtedy kolejność elementów w tablicy zostałaby odwrócona dwukrotnie, a co za tym idzie, po zakończeniu działania funkcji `odwroc` kolejność elementów tablicy `tab` pozostałaby niezmieniona.

Zadanie 4.2.12bListing 10.64. Rozwiązanie zadania 4.2.12b w języku C/C++

```
void przesun(unsigned int n, int * tab){  
    int i ,pom=tab[ 0];  
    for(i=0;i<n-1; i++)  
        tab[ i]=tab[ i+1];  
    tab[ n-1]=pom;  
}
```

Kolejność, w jakiej zmienna i przebiega wartości ze zbioru $\{0, \dots, n\}$, nie jest obojętna, o czym można się przekonać porównując powyższe rozwiązanie z rozwiązaniem zadania 4.2.12c

Zadanie 4.2.12c

Listing 10.65. Rozwiązanie zadania 4.2.12c w języku C/C++

```
void przesun(unsigned int n, int * tab){
    int i ,pom=tab[n-1];
    for(i=n-2;i>=0;i--)
        tab[i+1]=tab[i];
    tab[0]=pom;
}
```

Gdyby w powyższym programie zmienna *i* przebiegała wartości od 0 do $n - 2$, tak jak ma to miejsce w programie 10.64, to w efekcie do wszystkich komórek tablicy *tab*, za wyjątkiem komórki *tab[0]*, zostałaby wstawiona pierwotna wartość komórki *tab[0]*.

Zadanie 4.2.12d Istnieje wiele algorytmów sortujących tablice. Poniżej zaimplementowany został jeden z prostszych algorytmów. Jego idea jest następująca: wyszukać największy element tablicy i zamienić go miejscami z ostatnim, po czym znaleźć największy element spośród pierwszych $n - 1$ elementów i zamienić go miejscami z elementem o indeksie $n - 1$ etc.

Listing 10.66. Rozwiązanie zadania 4.2.12d w języku C/C++

```
int maksimum(unsigned int n, int * tab){
    int i , max=0;
    for(i=1;i<n;i++)
        if (tab[i]>tab[max])
            max=i ;
    return max;
}

void sortuj (unsigned int n, int * tab){
    int i ,j ,pom;
    for(i=0;i<n-1;i++){
        j=maksimum(n-i ,tab);
        pom=tab[n-i -1];
        tab[n-i -1]=tab[j];
        tab[j]=pom;
    }
}
```

Co prawda niniejszy zbiór zadań nie obejmuje zagadnień z zakresu algorytmiki, ale mimo to, jako ciekawostkę, przedstawiono poniżej implementację algorytmu sortowania przez scalanie. Dla dużych tablic algorytm ten działa znacznie szybciej od tego przedstawionego w programie 10.66

Listing 10.67. Rozwiązanie zadania 4.2.12d w języku C/C++

```
void merge(int* tablica ,int start ,int srodek ,int koniec){
    int tab_pom[koniec-start];
    int i,j,k;
    i = start;
    k = 0;
    j = srodek + 1;

    while ((i <= srodek)&&(j <= koniec)){
        if (tablica[j] < tablica[i]){
            tab_pom[k] = tablica[j];
            j = j + 1;
        }
        else{
            tab_pom[k] = tablica[i];
            i = i + 1;
        }
        k = k + 1;
    }

    if (i <= srodek)
        while (i <= srodek){
            tab_pom[k] = tablica[i];
            i = i + 1;
            k = k + 1;
        }
    else
        while (j <= koniec){
            tab_pom[k] = tablica[j];
            j = j + 1;
            k = k + 1;
        }
    for(i= 0;i<=koniec-start;i++)
        tablica[start + i]= tab_pom[i];
}

void merge_sort(int * tablica , int start , int koniec){
    int srodek;
    if (start != koniec){
        srodek = (start + koniec) / 2;
        merge_sort(tablica , start , srodek);
        merge_sort(tablica , srodek + 1 , koniec);
        merge(tablica , start , srodek , koniec);
    }
}

void sortuj(unsigned int n, int * tab){
    merge_sort(tab ,0 ,n-1);
}
```

Zadanie 4.2.13

Listing 10.68. Rozwiązanie zadania 4.2.13 w języku C

```
int * alokuj (unsigned int n){
    return malloc(n*sizeof(int));
}
```

Należy pamiętać, że w takiej sytuacji nie należy zwracać wskaźnika do tablicy utworzonej automatycznie, gdyż przestaje ona istnieć w momencie zakończenia działania funkcji. Poniżej rozwiązanie dla języka C++

Listing 10.69. Rozwiązanie zadania 4.2.13 w języku C++

```
int * alokuj (unsigned int n){
    return new int[n];
}
```

Zadanie 4.2.15

Listing 10.70. Rozwiązanie zadania 4.2.15 w języku C

```
void zwolnij (int *tab){
    free(tab);
}
```

To samo w języku C++:

Listing 10.71. Rozwiązanie zadania 4.2.15 w języku C++

```
void zwolnij (int * tab){
    delete [] tab;
}
```

10.7. Rozwiązań do zadań z rozdziału 5.2

Zadanie 5.2.1 Najpierw wersja dla typu `char`:

Listing 10.72. Rozwiązanie zadania 5.2.1 w języku C/C++

```
int wyczysc(char *nap){
    nap[0]=0;
}
```

Wersja dla typu `wchar_t` różni się niewiele:

Listing 10.73. Rozwiążanie zadania 5.2.1 w języku C/C++

```
int wyczysc(wchar_t *nap){
    nap[0]=0;
}
```

Zadanie 5.2.2 Najpierw wersja dla typu `char`:

Listing 10.74. Rozwiążanie zadania 5.2.2 w języku C/C++

```
int dlugosc(char *nap){
    int i=0;
    while(nap[i]!=0)
        i++;
    return i;
}
```

Wersja dla typu `wchar_t` różni się niewiele:

Listing 10.75. Rozwiążanie zadania 5.2.2 w języku C/C++

```
int dlugosc(wchar_t *nap){
    int i=0;
    while(nap[i]!=0)
        i++;
    return i;
}
```

Zadanie 5.2.4

Listing 10.76. Rozwiążanie zadania 5.2.4 w języku C/C++

```
int porownaj(char *nap1, char * nap2){
    int i;
    for(i=0;(nap1[i]!=0)&&(nap2[i]!=0);i++)
        if (nap1[i]!=nap2[i])
            return (nap1[i]<nap2[i])?1:0;
    if (nap1[i]==0)
        return 0;
    else
        return 1;
}
```

W powyższym rozwiążaniu wykorzystany został fakt, że w tablicy kodów ASCII małe łacińskie litery są ułożone zgodnie z porządkiem alfabetycznym

Zadanie 5.2.7

Listing 10.77. Rozwiązanie zadania 5.2.7 w języku C/C++

```
void sklej(char *nap1, char * nap2, char * nap3){
    int i, j;
    for(i=0;nap1[i]!=0; i++)
        nap3[i]=nap1[i];
    for(j=0;nap2[j]!=0; i++,j++)
        nap3[i]=nap2[j];
    nap3[i]=0;
}
```

Wersja dla napisów o znakach typu **wchar_t** jest podobna.

Zadanie 5.2.8

Listing 10.78. Rozwiązanie zadania 5.2.8 w języku C/C++

```
void maleduze(char *nap){
    int i;
    for(i=0;nap[i]!=0; i++)
        if ((nap[i]>='a')&&(nap[i]<='z'))
            nap[i]-=(‘a’-‘A’);
}
```

Zadanie 5.2.9

Listing 10.79. Rozwiązanie zadania 5.2.9 w języku C/C++

```
void wytnij(char *nap, int n, int m){
    int i, dl;
    for(dl=0;nap[dl]!=0; dl++);
    if (dl+1>m){
        for(i=0;i+m<dl; i++)
            nap[n+i]=nap[i+m+1];
    }
    else
        if ((n<dl)&&(dl+1<=m) )
            nap[n]=0;
}
```

Wersja dla napisów o znakach typu **wchar_t** jest podobna.

Zadanie 5.2.10

Listing 10.80. Rozwiązanie zadania 5.2.10 w języku C/C++

```

bool porownaj(char *nap1, char* nap2, int n){
    int i;
    for(i=0;(nap1[ i ]!=0)&&(nap2[ i ]!=0) ;i++)
        if (nap1[ n+i ]!=nap2[ i ])
            return false;
    if (nap2[ i ]==0)
        return true;
    else
        return false;
}

void wytnij2(char *nap1, char* nap2){
    int i ,dl;
    for(dl=0;nap2[ dl ]!=0;dl++);
    for(i=0;nap1[ i ]!=0; i++)
        if (porownaj(nap1,nap2,i)){
            wytnij(nap1,i,i+dl-1);
            return;
        }
}

```

Użyta w powyższym rozwiązaniu funkcja `wytnij`, to funkcja z rozwiązania zadnia 5.2.9.

Wersja dla napisów o znakach typu `wchar_t` jest podobna.

Zadanie 5.2.11

Listing 10.81. Rozwiązanie zadania 5.2.11 w języku C/C++

```

void wytnijzw(char *nap1, char* nap2){
    int i ,dl;
    int wyst[256]={};
    for(i=0;nap2[ i ]!=0; i++)
        wyst[nap2[ i ]]=1;
    for(i=0,j=0;nap1[ i ]!=0; i++)
        if (wyst[nap1[ i ]]==0){
            if (j<i)
                nap1[ j ]=nap1[ i ];
            j++;
        }
        nap1[ j ]=0;
}

```

Wersja dla typu `wchar_t` jest trochę inna ze względu na fakt, że tablica `wyst` musiałaby w tym przypadku być bardzo duża:

Listing 10.82. Rozwiązanie zadania 5.2.11 w języku C/C++

```

bool czywyst(wchar_t z, wchar_t * nap){
    int i;
    for(i=0;nap[i]!=0; i++)
        if (nap[i]==z)
            return true;
    return false;
}

void wytnijzw(wchar_t *nap1, wchar_t * nap2){
    int i,j;
    for(i=0,j=0;nap1[i]!=0; i++){
        if (!czywyst(nap1[i],nap2)){
            if (j<i)
                nap1[j]=nap1[i];
            j++;
        }
        nap1[j]=0;
    }
}

```

Zadanie 5.2.13

Listing 10.83. Rozwiązanie zadania 5.2.13 w języku C/C++

```

void wytnijtm(wchar_t *nap1, wchar_t * nap2){
    int i,j;
    for(i=0,j=0;nap1[i]!=0; i++)
        if (nap1[i]!=nap2[i]){
            if (j<i)
                nap1[j]=nap1[i];
            j++;
        }
        nap1[j]=0;
}

```

Wersja dla napisów o znakach typu `char` jest podobna.

Zadanie 5.2.14 Wersja dla napisów o znakach typu `char`:

Listing 10.84. Rozwiązanie zadania 5.2.14 w języku C/C++

```

void wypisz(char* nap){
    printf("%"s",nap);
}

```

Wersja dla napisów o znakach typu `wchar_t`:

Listing 10.85. Rozwiązanie zadania 5.2.14 w języku C/C++

```
void wypisz(wchar_t* nap){
    wprintf(L"%ls", nap);
}
```

Do wyświetlania napisów o znakach typu `wchar_t` można użyć także funkcji `printf`:

Listing 10.86. Rozwiązanie zadania 5.2.14 w języku C/C++

```
void wypisz(wchar_t* nap){
    printf("%ls", nap);
}
```

Rozwiązańia w języku C++:

Listing 10.87. Rozwiązanie zadania 5.2.14 w języku C++

```
void wypisz(char* nap){
    cout<<nap;
}
```

Wersja dla typu `wchar_t` w języku C++:

Listing 10.88. Rozwiązanie zadania 5.2.14 w języku C++

```
void wypisz(wchar_t* nap){
    wcout<<nap;
}
```

W przeciwieństwie do języka C, gdzie funkcji `printf` można używać do wypisywania napisów o znakach typu `wchar_t`, w języku C++, aby operować na takich napisach, trzeba używać strumienia `wcout` zamiast `cout`.

Zadanie 5.2.15 Wersja dla napisów typu `string`:

Listing 10.89. Rozwiązanie zadania 5.2.15 w języku C++

```
void wypisz(string s){
    cout<<s;
}
```

Wersja dla napisów typu `wstring`:

Listing 10.90. Rozwiązanie zadania 5.2.15 w języku C++

```
void wypisz(wstring s){
    wcout<<s;
}
```

Zadanie 5.2.16 Wersja dla napisów o znakach typu `char`:

Listing 10.91. Rozwiązanie zadania 5.2.16 w języku C/C++

```
void wczytaj(char* nap){
    scanf("%s", nap);
}
```

Wersja dla napisów o znakach typu `wchar_t`:

Listing 10.92. Rozwiązanie zadania 5.2.16 w języku C/C++

```
void wczytaj(wchar_t* nap){
    wsscanf(L"%ls", nap);
}
```

Do wczytywania napisów o znakach typu `wchar_t` można użyć także funkcji `scanf`:

Listing 10.93. Rozwiązanie zadania 5.2.16 w języku C/C++

```
void wczytaj(wchar_t* nap){
    scanf("%ls", nap);
}
```

Przy wczytywaniu napisów przy użyciu funkcji `scanf` nie pisze się znaku „`&`” przed zmienną, do której wczytujemy wartość. Jest tak dlatego, że zmienne tablicowe są wskaźnikami na bloki pamięci zawierające właściwą tablicę.

Inną rzeczą, o której należy pamiętać, jest fakt, że wczytując napis do tablicy należy zadbać o to, żeby tablica była wystarczającego rozmiaru. W praktyce jedynym sposobem na zapewnienie tego jest dodanie parametru przed specyfikatorami formatu `s` i `ls` w funkcjach `scanf` i `wsscanf`, który ograniczy rozmiar wczytywanych napisów.

Rozwiązań w języku C++:

Listing 10.94. Rozwiązanie zadania 5.2.16 w języku C++

```
void wczytaj(char* nap){
    cin>>nap;
}
```

Wersja dla typu `wchar_t` w języku C++:

Listing 10.95. Rozwiązanie zadania 5.2.16 w języku C++

```
void wczytaj(wchar_t* nap){
    wcin>>nap;
}
```

W przeciwieństwie do języka C, gdzie funkcji `scanf` można używać do wczytywania napisów o znakach typu `wchar_t`, w języku C++, aby operować na takich napisach, trzeba używać strumienia `wcin` zamiast `cin`.

Zadanie 5.2.17 Wersja dla napisów typu `string`:

Listing 10.96. Rozwiązanie zadania 5.2.17 w języku C++

```
void wczytaj(string& s){
    cin>>s;
}
```

Wersja dla napisów typu `wstring`:

Listing 10.97. Rozwiązanie zadania 5.2.16 w języku C++

```
void wczytaj(wstring& s){
    wcin>>s;
}
```

Zadanie 5.2.18 Wersja dla napisów o znakach typu `char`:

Listing 10.98. Rozwiązanie zadania 5.2.18 w języku C

```
char * pierwszy(char** tnap, int n){
    int i, min=0;
    char * wyn;
    for(i=1;i<n; i++)
        if (stromo(tnap[min],tnap[i])>0)
            min=i ;
    wyn=malloc ((strlen(tnap[min])+1)*sizeof(char));
    strcpy(wyn,tnap[min]);
    return wyn;
}
```

Wersja dla napisów o znakach typu `wchar_t`:

Listing 10.99. Rozwiązanie zadania 5.2.18 w języku C

```
wchar_t * pierwszy(wchar_t** tnap, int n){
    int i, min=0;
```

```
wchar_t * wyn;
for(i=1;i<n;i++)
    if (wcscmp(tnap[min],tnap[i])>0)
        min=i;
wyn=malloc((wcslen(tnap[min])+1)*sizeof(wchar_t));
wcscpy(wyn,tnap[min]);
return wyn;
}
```

Rozwiązanie w języku C++ jest analogiczne. Wystarczy skorzystać z faktu, że funkcje biblioteczne języka C są dostępne również w C++.

Zadanie 5.2.19 Wersja dla napisów typu `string`:

Listing 10.100. Rozwiązanie zadania 5.2.19 w języku C

```
string pierwszy(string* nap,int n){
    int i, min=0;
    for(i=1;i<n;i++)
        if (nap[min]>nap[i])
            min=i;
    return nap[min];
}
```

Wersja dla napisów typu `wstring` jest niemal identyczna.

Zadanie 5.2.20 Wersja dla napisów o znakach typu `char`:

Listing 10.101. Rozwiązanie zadania 5.2.20 w języku C

```
char * godzina(int godz, int min, int sek){
    char * wyn=malloc(9*sizeof(char));
    sprintf(wyn,"%02d:%02d:%02d",godz,min,sek);
    return wyn;
}
```

Wersja dla napisów o znakach typu `wchar_t`:

Listing 10.102. Rozwiązanie zadania 5.2.20 w języku C

```
wchar_t * godzina(int godz, int min, int sek){
    wchar_t * wyn=malloc(9*sizeof(wchar_t));
    swprintf(wyn,9,L"%02d:%02d:%02d",godz,min,sek);
    return wyn;
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 5.2.21 Wersja zwracająca napis o typie `string`:

Listing 10.103. Rozwiązanie zadania 5.2.21 w języku C++

```
string godzina(int godz, int min, int sek){
    stringstream wyn;
    wyn<<((godz<10)?"0":"")<<godz;
    wyn<<": "<<((min<10)?"0":"")<<min;
    wyn<<": "<<((sek<10)?"0":"")<<sek;
    return wyn.str();
}
```

Wersja zwracająca napis o typie `wstring`:

Listing 10.104. Rozwiązanie zadania 5.2.21 w języku C++

```
wstring godzina(int godz, int min, int sek){
    wstringstream wyn;
    wyn<<((godz<10)?L"0":L"")<<godz;
    wyn<<L": "<<((min<10)?L"0":L"")<<min;
    wyn<<L": "<<((sek<10)?L"0":L"")<<sek;
    return wyn.str();
}
```

Zadanie 5.2.22 Wersja dla napisów o znakach typu `char`:

Listing 10.105. Rozwiązanie zadania 5.2.22 w języku C

```
char * sklej(char * nap1, char * nap2, char * nap3){
    char * wyn=malloc((strlen(nap1)+strlen(nap2)
                       +strlen(nap3)+1)*sizeof(char));
    strcpy(wyn,nap1);
    strcat(wyn,nap2);
    strcat(wyn,nap3);
    return wyn;
}
```

Wersja dla napisów o znakach typu `wchar_t`:

Listing 10.106. Rozwiązanie zadania 5.2.22 w języku C

```
wchar_t * sklej(wchar_t * nap1, wchar_t * nap2, wchar_t * nap3){
    wchar_t * wyn=malloc((wcslen(nap1)+wcslen(nap2)
                          +wcslen(nap3)+1)*sizeof(wchar_t));
    wcscpy(wyn,nap1);
    wcscat(wyn,nap2);
    wcscat(wyn,nap3);
    return wyn;
```

 }

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 5.2.23 Wersja zwracająca napis o typie `string`:

Listing 10.107. Rozwiązanie zadania 5.2.23 w języku C++

```
string sklej(string nap1, string nap2, string nap3){
    return nap1+nap2+nap3;
}
```

Rozwiązanie dla napisów typu `wstring` jest podobne.

Zadanie 5.2.26 Wersja dla napisów o znakach typu `char`:

Listing 10.108. Rozwiązanie zadania 5.2.26 w języku C/C++

```
void maleduze(char * nap){
    int i;
    for(i=0;nap[i]!=0;i++)
        nap[i]=toupper(nap[i]);
}
```

Wersja dla napisów o znakach typu `wchar_t`: jest bardzo podobna:

Listing 10.109. Rozwiązanie zadania 5.2.26 w języku C/C++

```
void maleduze(wchar_t * nap){
    int i;
    for(i=0;nap[i]!=0;i++)
        nap[i]=towupper(nap[i]);
}
```

Zadanie 5.2.27 Wersja dla napisów typu `string`:

Listing 10.110. Rozwiązanie zadania 5.2.27 w języku C++

```
void maleduze(string& nap){
    for(int i=0;i<nap.length();i++)
        nap[i]=toupper(nap[i]);
}
```

Wersja dla napisów typu `wstring` jest bardzo podobna.

Listing 10.111. Rozwiązanie zadania 5.2.27 w języku C++

```
void maleduze(wstring & nap){
```

```
for (int i=0;i<nap.length();i++)
    nap[i]=towupper(nap[i]);
}
```

10.8. Rozwiązania do zadań z rozdziału 6.2

Zadanie 6.2.1

Listing 10.112. Rozwiązanie zadania 6.2.1 w języku C

```
int** alokuj(unsigned int n, unsigned int m){
    int ** t=malloc(n*sizeof(int *));
    int i;
    for(i=0;i<n;i++)
        t[i]=malloc(m*sizeof(int));
    return t;
}
```

W języku C++ rozwiązanie powyższego zadania wygląda następująco:

Listing 10.113. Rozwiązanie zadania 6.2.1 w języku C++

```
int** alokuj(unsigned int n, unsigned int m){
    int ** t= new int*[n];
    for(int i=0;i<n;i++)
        t[i]= new int[m];
    return t;
}
```

Zadanie 6.2.2

Listing 10.114. Rozwiązanie zadania 6.2.2 w języku C

```
int (* alokuj(unsigned int n, unsigned int m))[]{
    return malloc(n*sizeof(int [m]));
}
```

W języku C++ nie da się wprost rozwiązać powyższego zadania ze względu na brak możliwości utworzenia w nim wielowymiarowych, w pełni dynamicznych tablic, takich jak te w programie 10.114. Ten problem można rozwiązać tworząc klasę zawierającą jednowymiarową tablicę dynamiczną i przeciążając odpowiednie operatory. Nie mieści się to jednak w zakresie materiału obejmowanego przez niniejszy skrypt.

Zadanie 6.2.3

Listing 10.115. Rozwiązanie zadania 6.2.3 w języku C

```
void zwolnij(unsigned int n, unsigned int m, int ** t){
    int i;
    for(i=0;i<n; i++)
        free(t[i]);
    free(t);
}
```

W języku C++ rozwiązanie powyższego zadania wygląda następująco:

Listing 10.116. Rozwiązanie zadania 6.2.3 w języku C++

```
void zwolnij(unsigned int n, unsigned int m, int ** t){
    for(int i=0;i<n; i++)
        delete [] t[i];
    delete [] t;
}
```

Zadanie 6.2.4

Listing 10.117. Rozwiązanie zadania 6.2.4 w języku C

```
void zwolnij(unsigned int n, int t[ ][n]){
    free(t);
}
```

Zadanie 6.2.7

Listing 10.118. Rozwiązanie zadania 6.2.7 w języku C

```
int** alokuj(unsigned int n){
    int ** t=malloc(n*sizeof(int*));
    int i ;
    for(i=0;i<n; i++)
        t[i]=malloc((i+1)*sizeof(int));
    return t ;
}
```

W języku C++ rozwiązanie powyższego zadania wygląda następująco:

Listing 10.119. Rozwiązanie zadania 6.2.7 w języku C++

```
int** alokuj(unsigned int n){
    int ** t= new int *[n];
    for(int i=0;i<n; i++)
        t[i]= new int [i+1];
    return t ;
}
```

 }

Zadanie 6.2.8

Listing 10.120. Rozwiązanie zadania 6.2.8 w języku C/C++

```
void zeruj(int t [][100], unsigned int n){
    int i ,j ;
    for(i=0;i<n ;i++)
        for(j=0;j<100;j++)
            t [ i ][ j ]=0;
}
```

Zadanie 6.2.9

Listing 10.121. Rozwiązanie zadania 6.2.9 w języku C/C++

```
void zeruj(int** t , unsigned int n, unsigned int m){
    int i ,j ;
    for(i=0;i<n ;i++)
        for(j=0;j<m; j++)
            t [ i ][ j ]=0;
}
```

Zadanie 6.2.10

Listing 10.122. Rozwiązanie zadania 6.2.10 w języku C

```
void zeruj(unsigned int n, unsigned int m,int t [][m]){
    int i ,j ;
    for(i=0;i<n ;i++)
        for(j=0;j<m; j++)
            t [ i ][ j ]=0;
}
```

W powyższym rozwiążaniu ważne jest, żeby deklaracja argumentu **m** wystąpiła przed deklaracją argumentu **t** (innym przypadku kompilator zgłosi błąd użycia niezadeklarowanej zmiennej **m** w deklaracji **t**).

Zadanie 6.2.14

Listing 10.123. Rozwiązanie zadania 6.2.14 w języku C/C++

```
int suma(int t [][100][100]){
    int i ,j ,k,sum=0;
    for(i=0;i<100;i++)
        for(j=0;j<100;j++)
            for(k=0;k<100;k++)
```

```
    sum+=t[i][j][k];
  return sum;
}
```

Zadanie 6.2.16

Listing 10.124. Rozwiązań zadania 6.2.16 w języku C/C++

```
int max_sred(int **t, unsigned int n, unsigned int m){
  int i,j,sum,max;
  double wart;
  for(i=0;i<n;i++){
    sum=0;
    for(j=0;j<m;j++)
      sum+=t[i][j];
    if (((double)sum/m>wart) || (i==0)){
      max=i;
      wart=(double)sum/m;
    }
  }
  return max;
}
```

Zadanie 6.2.17

Listing 10.125. Rozwiązań zadania 6.2.17 w języku C/C++

```
double max_sred(int **t, unsigned int n, unsigned int m){
  int i,j,sum;
  double wart;
  for(i=0;i<n;i++){
    sum=0;
    for(j=0;j<m;j++)
      sum+=t[i][j];
    if (((double)sum/m>wart) || (i==0))
      wart=(double)sum/m;
  }
  return wart;
}
```

Zadanie 6.2.19

Listing 10.126. Rozwiązań zadania 6.2.19 w języku C/C++

```
void przepisz(int **t1, int **t2, unsigned int n,
                unsigned int m){
  int i,j;
  for(i=0;i<n;i++)
    for(j=0;j<m;j++)
      t2[i][j]=t1[i][j];
```

 }

Zadanie 6.2.21

Listing 10.127. Rozwiązanie zadania 6.2.21 w języku C/C++

```
void pom (unsigned int n, int * tab){
    int i ,p;
    for(i=0;i<n/2; i++){
        p=tab[ i ];
        tab[ i ]=tab[ n-1-i ];
        tab[ n-1-i ]=p;
    }
}

void odwroc(int **t, unsigned int n, unsigned int m){
    int i ,j;
    for(i=0;i<n; i++)
        pom(n,t[ i ]);
}
```

W powyższym rozwiązaniu wykorzystano strukturę wielowymiarowych tablic tablic oraz rozwiązanie zadania 4.2.12a.

Zadanie 6.2.22

Listing 10.128. Rozwiązanie zadania 6.2.22 w języku C

```
void pom (unsigned int n, unsigned int j, int tab [][]n) {
    int i ,p;
    for(i=0;i<n/2; i++){
        p=tab[ j ][ i ];
        tab[ j ][ i ]=tab[ j ][ n-1-i ];
        tab[ j ][ n-1-i ]=p;
    }
}

void odwroc(unsigned int n, unsigned int m, int t [][]m) {
    unsigned int i ,j;
    for(i=0;i<n; i++)
        pom(n,i,t);
}
```

Zadanie 6.2.23 Zadanie to można rozwiązać na kilka sposobów. Korzystając z faktu, że funkcja dostaje jako argument tablicę tablic można po prostu przepiąć wskaźniki do wierszy:

Listing 10.129. Rozwiązanie zadania 6.2.23 w języku C/C++

```
void przepnij( unsigned int n, unsigned int m, int ** t){
    int i;
    int * pom=t[n-1];
    for(i=n-1;i>0;i--)
        t[i]=t[i-1];
    t[0]=pom;
}
```

Powyzszego rozwiązania nie można zastosować w przypadku, gdy istnieje możliwość, że gdzieś w programie przechowywany jest wskaźnik do pojedynczego wiersza przekazanej w argumencie tablicy. W takim przypadku trzeba przepisywać wartości poszczególnych elementów tablicy:

Listing 10.130. Rozwiązań zadania 6.2.23 w języku C/C++

```
void przepisz( unsigned int n, unsigned int m, int ** t){
    int i,j,pom;
    for(i=0;i<m;i++){
        pom=t[n-1][i];
        for(j=n-1;j>0;j--)
            t[j][i]=t[j-1][i];
        t[0][i]=pom;
    }
}
```

Drugie z zaprezentowanych rozwiązań ma tę zaletę, że jest możliwe do zastosowania także w przypadku tablic dwuwymiarowych oraz łatwo je zaadoptować do zamiany miejscami kolumn.

Zadanie 6.2.27

Listing 10.131. Rozwiązań zadania 6.2.27 w języku C/C++

```
void zamien( unsigned int n, int *** tab){
    int i,j,k,pom;
    for(i=0;i<n;i++){
        for(j=i;j<n;j++){
            if (i==j)
                k=i;
            else
                k=i+1;
            for (;k<n;k++){
                pom=tab[i][j][k];
                tab[i][j][k]=tab[j][k][i];
                tab[j][k][i]=tab[k][i][j];
                tab[k][i][j]=pom;
            }
        }
    }
}
```

Zadanie 6.2.32

Listing 10.132. Rozwiązanie zadania 6.2.32 w języku C/C++

```

int ** pomnoz(unsigned int n, int **tab1, int **tab2){
    int i,j,k;
    int ** tab3=malloc(n*sizeof(int *));
    for(i=0;i<n;i++)
        tab3[i]=malloc(n*sizeof(int));
    for(i=0;i<n;i++)
        for(j=0;j<n;j++){
            tab3[i][j]=0;
            for(k=0;k<n;k++)
                tab3[i][j]+=tab1[i][k]*tab2[k][j];
        }
    return tab3;
}

```

Zadanie 6.2.36

Listing 10.133. Rozwiązanie zadania 6.2.36 w języku C

```

int ** utworz(int n){
    int i, ** t=malloc(n*sizeof(int *));
    for(i=0;i<n;i++)
        t[i]=malloc(n*sizeof(int));
    return t;
}

void usun(int n, int ** t){
    int i;
    for(i=0;i<n;i++)
        free(t[i]);
    free(t);
}

void przepisz(int n, int m, int ** t1, int ** t2){
    int i1,i2,j1,j2;
    for(i1=0,i2=0;i1<n;i1++)
        if (i1!=m){
            for(j1=1,j2=0;j1<n;j1++,j2++)
                t2[i2][j2]=t1[i1][j1];
            i2++;
        }
}

int wyznacznik (unsigned int n, int **tab){
    int i,j, wyz;

```

```

int ** t;
if (n==1)
    return tab[0][0];
wyz=0;
t=utworz(n-1);
for(i=0;i<n;i++){
    przepisz(n,i,tab,t);
    if (i%2==0)
        wyz+=tab[i][0]*wyznacznik(n-1,t);
    else
        wyz-=tab[i][0]*wyznacznik(n-1,t);
}
usun(n,t);
return wyz;
}

```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 6.2.37

Listing 10.134. Rozwiązanie zadania 6.2.37 w języku C/C++

```

void przepisz(int n, int m, int t1[][n], int t2[][n-1]){
    int i1,i2,j1,j2;
    for(i1=0,i2=0;i1<n;i1++)
        if (i1!=m){
            for(j1=1,j2=0;j1<n;j1++,j2++)
                t2[i2][j2]=t1[i1][j1];
            i2++;
        }
}

int wyznacznik (unsigned int n, int tab[][n]){
    int i,j, wyz;
    int t[n-1][n-1];
    if (n==1)
        return tab[0][0];
    wyz=0;
    for(i=0;i<n;i++){
        przepisz(n,i,tab,t);
        if (i%2==0)
            wyz+=tab[i][0]*wyznacznik(n-1,t);
        else
            wyz-=tab[i][0]*wyznacznik(n-1,t);
    }
    return wyz;
}

```

10.9. Rozwiązania do zadań z rozdziału 7.2

Zadanie 7.2.1

Listing 10.135. Rozwiązanie zadania 7.2.1 w języku C/C++

```
struct trojkat {
    double a,b,c;
};

double obwod(struct trojkat t){
    return t.a+t.b+t.c;
}
```

Ważną rzeczą jest, żeby nie zapominać o średniku po definicji typów złożonych. Komunikat kompilatora o błędzie w przypadku pominięcia średnika może mówić o błędzie w zupełnie innym miejscu programu.

W języku C++, nie trzeba powtarzać słowa kluczowego **struct** przy deklaracji zmiennej mającej przechowywać wcześniej zdefiniowaną strukturę, a więc rozwiązanie w nim zadania mogłoby wyglądać następująco:

Listing 10.136. Rozwiązanie zadania 7.2.1 w języku C++

```
struct trojkat {
    double a,b,c;
};

double obwod(trojkat t){
    return t.a+t.b+t.c;
}
```

W języku C, aby uniknąć powtarzania przed nazwą typu słowa kluczowego **struct** (a także słów kluczowych **union** i **enum**), można użyć polecenia **typedef**:

Listing 10.137. Rozwiązanie zadania 7.2.1 w języku C/C++

```
struct trojkat {
    double a,b,c;
};

typedef struct trojkat troj;

double obwod(troj t){
    return t.a+t.b+t.c;
}
```

Polecenia `typedef` można używać także w języku C++.

Zadanie 7.2.2

Listing 10.138. Rozwiązanie zadania 7.2.2 w języku C/C++

```
void przepisz(struct trojkat troj1, struct trojkat *troj2){
    *troj2=troj1;
}
```

Warto zauważyć, że wszystkie pola struktury są przepisywane za jednym zamachem.

Zadanie 7.2.3

Listing 10.139. Rozwiązanie zadania 7.2.3 w języku C/C++

```
struct punkt{
    double x,y,z;
};

double minimum(struct punkt tab[], int n){
    int i,j;
    double pom,min=sqrt(pow(tab[1].x-tab[0].x,2)
        +pow(tab[1].y-tab[0].y,2)
        +pow(tab[1].z-tab[0].z,2));
    for(i=0;i<n-1;i++)
        for(j=i+1;j<n; j++){
            pom=sqrt(pow(tab[j].x-tab[i].x,2)
                +pow(tab[j].y-tab[i].y,2)
                +pow(tab[j].z-tab[i].z,2));
            if (pom<min) min=pom;
        }
    return min;
}
```

Zadanie 7.2.4

Listing 10.140. Rozwiązanie zadania 7.2.4 w języku C/C++

```
void przepisz(struct punkt tab1[], struct punkt tab2[],
                unsigned int n){
    int i;
    for(i=0;i<n; i++)
        tab2[i]=tab1[i];
}
```

Zadanie 7.2.5

Listing 10.141. Rozwiązanie zadania 7.2.5 w języku C/C++

```
struct punkt10{
    double t[10];
};

void przepisz(struct punkt10 tab1[], struct punkt10 tab2[],  
              unsigned int n){
    int i;
    for(i=0;i<n;i++)
        tab2[i]=tab1[i];
}
```

W zdefiniowanej powyżej funkcji **przepisz** cała struktura, a więc także cała zawartość pola **t** jest przepisywana przy użyciu pojedynczego operatora przypisania.

Zadanie 7.2.6

Listing 10.142. Rozwiązanie zadania 7.2.6 w języku C

```
struct punktn{
    double *t;
    int w;
};

void przepisz(struct punktn tab1[], struct punktn tab2[],  
              unsigned int n){
    int i,j;
    for(i=0;i<n;i++){
        tab2[i].t=malloc(tab1[i].w*sizeof(double));
        for(j=0;j<tab1[i].w;j++)
            tab2[i].t[j]=tab1[i].t[j];
    }
}
```

W tym przypadku należy tworzyć i przepisywać tablice ręcznie (operator przypisania dla **struct punktn** przepisałby wskaźnik, ale nie stworzyłby nowej tablicy).

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.2.9

Listing 10.143. Rozwiązanie zadania 7.2.9 w języku C/C++

```
struct lista{
    int i;
```

```
struct lista * wsk;
};
```

Zadanie 7.2.10

Listing 10.144. Rozwiązanie zadania 7.2.10 w języku C/C++

```
union super_int{
    int i;
    unsigned int u;
};
```

Zadanie 7.2.11

Listing 10.145. Rozwiązanie zadania 7.2.11 w języku C

```
union Liczba{
    int i;
    double d;
};

struct Dane{
    int tp;
    union Liczba zaw;
};

struct Dane wczytaj(){
    struct Dane dane;
    printf("Jezeli chcesz podac liczbe calkowita wpisz 0\n");
    printf("jeżeli chcesz podac liczbe wymierna wpisz 1\n");
    scanf("%d", &dane.tp);
    if (dane.tp==0){
        printf("Wpisz liczbe calkowita");
        scanf("%d", &dane.zaw.i);
    }
    else{
        printf("Wpisz liczbe wymierna");
        scanf("%lf", &dane.zaw.d);
    }
    return dane;
}
```

W językach C i C++ rozróżniane są duże i małe litery. Dlatego w funkcji `wczytaj` możliwe było zadeklarowanie zmiennej `dane` typu `struct Dane`. W praktyce nie należy używać w programie nazw, które różnią się tylko wielkością liter. Używanie w programie podobnych nazw zmiennych, funkcji etc. sprzyja robienniu przypadkowych i trudnych do wykrycia błędów.

Rozwiązań w języku C++ wygląda analogicznie.

Zadanie 7.2.13

Listing 10.146. Rozwiązań zadania 7.2.13 w języku C/C++

```

struct trojkat{
    double a,h;
};

struct prostokat{
    double a,b;
};

struct trapez{
    double a,b,h;
};

union wymiary{
    struct trojkat troj, rown;
    struct prostokat prost;
    struct trapez trap;
};

struct figura{
    union wymiary wym;
    int fig;
};

double pole(struct figura f){
    switch (f.fig){
        case 0: return f.wym.troj.a*f.wym.troj.h/2;
        case 1: return f.wym.prost.a*f.wym.prost.b;
        case 2: return f.wym.rown.a*f.wym.rown.h;
        case 3: return f.wym.trap.h
                  *(f.wym.trap.a+f.wym.trap.b)/2;
    }
}

```

W powyższym rozwiązaniu pola `rown` i `troj` unii `wymiary` są tego samego typu. Jest to możliwe gdyż do liczenia pól trójkątów i równoległoboków potrzebne są te same wymiary (podstawa i wysokość).

Zadanie 7.2.14

Listing 10.147. Rozwiązań zadania 7.2.14 w języku C/C++

```

enum czworokat{
    kwadrat, prostokat, romb, rownoleglobok, trapez
};

```

Zadanie 7.2.17

Listing 10.148. Rozwiązanie zadania 7.2.17 w języku C

```
enum Plec{
    mezczyzna, kobieta
};

enum mezczyzna{
    wolny, zonaty
};

enum kobieta{
    wolna, mezatka
};

union czlowiek{
    enum mezczyzna m;
    enum kobieta k;
};

struct dane_osobowe{
    char imie[30];
    char nazwisko[30];
    enum Plec plec;
    union czlowiek stan_cywilny;
};

void wczytaj(struct dane_osobowe tab[], int n){
    int i, d;

    for(i=0;i<n;i++){
        printf("Czy teraz wczytujemy dane kobiety (1) ");
        printf("czy mezczyzny (2)?");
        scanf("%d",&d);
        if (d==1)
            tab[i].plec=kobieta;
        else
            tab[i].plec=mezczyzna;

        printf("podaj imie");
        scanf("%s",tab[i].imie);

        printf("podaj nazwisko");
        scanf("%s",tab[i].nazwisko);

        printf("podaj stan cywilny");
        if (tab[i].plec==kobieta)
            printf("(wolna - 0, mezatka - 1)");
    }
}
```

```

    else
        printf("(wolny=0,zonaty=1)");
        scanf("%d",&tab[ i ].stan_cywilny.k);
    }
}

```

Jak widać w powyższym rozwiążaniu, typ wyliczeniowy można wczytywać jak zmienne typu `int`. Ponadto niezależnie, czy wczytywane są dane kobiety czy mężczyzny, wczytywane są dane do pola `k` w unii `stan_cywilny` typu `union człowiek`. Można tak zrobić, gdyż pola `k` i `m` znajdują się w tym samym miejscu w pamięci, a więc wczytując dane do jednego pola, zmienia się jednocześnie wartość drugiego. Ponadto oba wspomniane pola są typów wyliczeniowych o tej samej liczbie zdefiniowanych wartości.

Funkcja `wczytaj` w języku C++ może wyglądać następująco:

Listing 10.149. Rozwiążanie zadania 7.2.17 w języku C++

```

void wczytaj(struct dane_osobowe tab[], int n){
    int i,d;

    for(i=0;i<n;i++){
        cout<<"Czy teraz wczytujemy dane kobiety (1) ";
        cout<<"czy mezczyzny (2) ?" <<endl;
        cin>>d;
        if (d==1)
            tab[ i ].plec=kobieta;
        else
            tab[ i ].plec=mezczyzna;

        cout<<"Podaj imię " <<endl;
        cin>>tab[ i ].imie;

        cout<<"Podaj nazwisko " <<endl;
        cin>>tab[ i ].nazwisko;

        cout<<"Podaj stan cywilny " <<endl;;
        if (tab[ i ].plec==kobieta){
            cout<<"wolna=0,mezatka=1" <<endl;
            cin>>d;
            if (d==0)
                tab[ i ].stan_cywilny.k=wolna;
            else
                tab[ i ].stan_cywilny.k=mezatka;
        }
        else{
            cout<<"wolny=0,zonaty=1" <<endl;
            cin>>d;
            if (d==0)
                tab[ i ].stan_cywilny.m=wolny;
            else
                tab[ i ].stan_cywilny.m=zonaty;
        }
    }
}

```

```

        else
            tab[ i ].stan_cywilny.m=zonaty ;
        }
    }
}

```

Podstawową różnicą w stosunku do rozwiązania w języku C jest niemożność wczytania wprost wartości do zmiennej typu wyliczeniowego. Aby to zrobić, trzeba by przeciążyć operator „`>>`”, ale to nie mieści się w zakresie materiału niniejszego zbioru zadań. Innym rozwiązaniem mogłoby być wczytanie wartości typu `int` i zrzutowanie jej do typu wyliczeniowego.

Podobnie jak w C, w języku C++ typy wyliczeniowe są domyślnie wyświetlane jako wartości typu `int`.

Zadanie 7.2.18

Listing 10.150. Rozwiązanie zadania 7.2.18 w języku C/C++

```

union sup_int{
    unsigned int l;
    unsigned char t[4];
};

```

Poprawność powyższego rozwiązania jest zależna od używanego kompilatora, gdyż standard języka C nie określa, z ilu bajtów ma się składać zmienna typu `int`. Typowy rozmiar typu `int` w systemach 32-bitowych to 4 bajty, stąd taki rozmiar tablicy `t`.

Zadanie 7.2.19

Listing 10.151. Rozwiązanie zadania 7.2.19 w języku C/C++

```

unsigned int funkcja(unsigned int a, unsigned int b){
    union sup_int poma, pomb, pomwyn;
    poma.l=a;
    pomb.l=b;
    pomwyn.t[0]=poma.t[0]*pomb.t[0];
    pomwyn.t[1]=poma.t[1]*pomb.t[1];
    pomwyn.t[2]=poma.t[2]*pomb.t[2];
    pomwyn.t[3]=poma.t[3]*pomb.t[3];
    return pomwyn.l;
}

```

10.10. Rozwiązania do zadań z rozdziału 7.3

Zadanie 7.3.1

Listing 10.152. Rozwiązanie zadania 7.3.1 w języku C/C++

```
struct element * utworz(){
    return NULL;
}
```

Zadanie 7.3.2

Listing 10.153. Rozwiązanie zadania 7.3.2 w języku C

```
void wyczysc(struct element* Lista){
    struct element * wsk=Lista;
    while(Lista!=NULL){
        Lista=Lista->next;
        free(wsk);
        wsk=Lista;
    }
}
```

Rozwiązanie w języku C++:

Listing 10.154. Rozwiązanie zadania 7.3.2 w języku C++

```
void wyczysc(element* Lista){
    element * wsk=Lista;
    while(Lista!=NULL){
        Lista=Lista->next;
        delete wsk;
        wsk=Lista;
    }
}
```

Zadanie 7.3.3

Listing 10.155. Rozwiązanie zadania 7.3.3 w języku C

```
struct element * dodaj(struct element* Lista, int a){
    struct element * wsk=malloc(sizeof(struct element));
    wsk->i=a;
    wsk->next=Lista;
    return wsk;
}
```

Rozwiązanie w języku C++:

Listing 10.156. Rozwiązanie zadania 7.3.3 w języku C++

```
element * dodaj(element* Lista , int a){
    element * wsk = new element;
    wsk->i=a;
    wsk->next=Lista ;
    return wsk;
}
```

Zadanie 7.3.4Listing 10.157. Rozwiązanie zadania 7.3.4 w języku C

```
struct element * dodajk(struct element* Lista , int a){
    struct element * wsk;
    if (Lista==NULL){
        Lista=wsk=malloc(sizeof(struct element));
    }
    else{
        wsk=Lista ;
        while(wsk->next!=NULL)
            wsk=wsk->next;
        wsk->next=malloc(sizeof(struct element));
        wsk=wsk->next;
    }
    wsk->i=a;
    wsk->next=NULL;
    return Lista;
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.5Listing 10.158. Rozwiązanie zadania 7.3.5 w języku C

```
struct element* dodajw(struct element* Lista ,
                        struct element* elem,int a){
    struct element * wsk=malloc(sizeof(struct element));
    wsk->i=a;
    if (elem==NULL){
        wsk->next=Lista ;
        Lista=wsk;
    }
    else{
        wsk->next=elem->next ;
        elem->next=wsk;
    }
    return Lista;
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.6

Listing 10.159. Rozwiązanie zadania 7.3.6 w języku C/C++

```
struct element * znajdz(struct element* Lista , int a){
    while((Lista!=NULL)&&(Lista->i!=a))
        Lista=Lista->next;
    return Lista;
}
```

W rozwiązaniu wykorzystany został fakt, że w językach C i C++, jeżeli pierwszy argument koniunkcji jest fałszywy, to drugi nie jest już sprawdzany (dzięki temu w drugim argumencie koniunkcji można założyć, że `Lista->i` istnieje).

Zadanie 7.3.7

Listing 10.160. Rozwiązanie zadania 7.3.7 w języku C

```
struct element * usun(struct element* Lista , int a){
    struct element * wsk,*wsk2;
    if (Lista==NULL)
        return Lista;
    wsk=Lista;
    if (Lista->i==a){
        Lista=Lista->next;
        free(wsk);
    }
    else {
        while((wsk->next!=NULL)&&(wsk->next->i!=a))
            wsk=wsk->next;
        if (wsk->next!=NULL){
            wsk2=wsk->next;
            wsk->next=wsk2->next;
            free(wsk2);
        }
    }
    return Lista;
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.8

Listing 10.161. Rozwiązanie zadania 7.3.8 w języku C

```

struct element * usunw(struct element* Lista ,
                      struct element * elem){
    struct element * wsk,*wsk2;
    if (Lista==NULL)
        return Lista;

    wsk=Lista;

    if (Lista==elem){
        Lista=Lista->next;
        free(wsk);
        return Lista;
    }

    while(( wsk->next!=NULL)&&(wsk->next!=elem ))
        wsk=wsk->next;
    if (wsk->next!=NULL){
        wsk2=wsk->next;
        wsk->next=wsk2->next;
        free(wsk2);
    }
    return Lista;
}

```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.9

Listing 10.162. Rozwiązanie zadania 7.3.9 w języku C

```

struct element * usunw2(struct element* Lista ,
                        struct element * elem){
    struct element * wsk;

    if(Lista==NULL)
        return Lista;

    if (elem==NULL){
        wsk=Lista;
        Lista=Lista->next;
    }
    else if (elem->next==NULL)
        return Lista;
    else{
        wsk=elem->next;
        elem->next=wsk->next;
    }
}

```

```
    free(wsk);
    return Lista;
}
```

Złożoność obliczeniowa powyższej funkcji jest mniejsza niż funkcji z Listingu 10.161. Z tego powodu operacje na listach w argumentach często wymagają podania, zamiast wskaźnika na jakiś element, wskaźnika do elementu poprzedniego.

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.10

Listing 10.163. Rozwiązanie zadania 7.3.10 w języku C

```
struct element * utworz(){
    struct element *wsk=malloc(sizeof(struct element));
    wsk->next=NULL;
    return wsk;
}
```

Inaczej niż ma to miejsce w przypadku listy bez głowy, tworzenie pustej listy z głową wymaga wykonania pewnych prostych operacji.

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.11

Listing 10.164. Rozwiązanie zadania 7.3.11 w języku C

```
void wyczysc(struct element* Lista){
    struct element * wsk=Lista->next;
    Lista=wsk;
    while(Lista!=NULL){
        Lista=Lista->next;
        free(wsk);
        wsk=Lista;
    }
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.12

Listing 10.165. Rozwiązanie zadania 7.3.12 w języku C

```
void dodaj(struct element *Lista , int a){
    struct element *wsk=malloc(sizeof(struct element));
    wsk->i=a;
```

```
wsk->next=Lista->next;
Lista->next=wsk;
}
```

W przeciwnieństwie do podobnego zadania dla list bez głowy, w przypadku list z głową funkcja nie musi zwracać wskaźnika do początku listy. Wynika to z tego, że w przypadku list z głową przed pierwszym elementem listy znajduje się głowa, do której wskaźnik się nie zmienia.

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.13

Listing 10.166. Rozwiązanie zadania 7.3.13 w języku C

```
void dodajk(struct element* Lista , int a){
    struct element * wsk=Lista ;
    while(wsk->next!=NULL)
        wsk=wsk->next ;
    wsk->next=malloc(sizeof(struct element));
    wsk=wsk->next ;
    wsk->i=a ;
    wsk->next=NULL;
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.14

Listing 10.167. Rozwiązanie zadania 7.3.14 w języku C

```
void dodajw(struct element *Lista , struct element * elem ,
            int a){
    struct element *wsk=malloc(sizeof(struct element));
    wsk->i=a ;
    wsk->next=elem->next ;
    elem->next=wsk ;
}
```

Należy zauważyć, że w powyższym rozwiązaniu pierwszy argument (wskaźnik na głowę listy) nie jest wykorzystywany..

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.15

Listing 10.168. Rozwiązanie zadania 7.3.15 w języku C/C++

```
struct element * znajdz(struct element* Lista , int a){
```

```

Lista=Lista->next;
while(( Lista!=NULL)&&(Lista->i!=a))
    Lista=Lista->next;
return Lista;
}

```

Zadanie 7.3.16

Listing 10.169. Rozwiązanie zadania 7.3.16 w języku C/C++

```

struct element * znajdzp(struct element* Lista , int a){
    while(( Lista->next!=NULL)&&(Lista->next->i!=a))
        Lista=Lista->next;
    return Lista;
}

```

Zadanie 7.3.17

Listing 10.170. Rozwiązanie zadania 7.3.17 w języku C

```

void usun(struct element* Lista , int a){
    struct element * wsk;
    while(( Lista->next!=NULL)&&(Lista->next->i!=a))
        Lista=Lista->next;
    if (Lista->next!=NULL){
        wsk=Lista->next;
        Lista->next=wsk->next;
        free(wsk);
    }
}

```

Porównując powyższe rozwiązanie z rozwiązaniem zadania 7.3.7, widać, o ile prostsze jest operowanie na listach z głową w stosunku do list bez głowy.

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.18

Listing 10.171. Rozwiązanie zadania 7.3.18 w języku C

```

void usunw(struct element* Lista , struct element * elem){
    struct element * wsk;
    while(( Lista->next!=NULL)&&(Lista->next!=elem))
        Lista=Lista->next;
    wsk=Lista->next;
    Lista->next=wsk->next;
    free(wsk);
}

```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.19

Listing 10.172. Rozwiązanie zadania 7.3.19 w języku C

```
void usunw2(struct element* Lista, struct element * elem){
    struct element * wsk=elem->next;
    elem->next=wsk->next;
    free(wsk);
}
```

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.20 Wersja dla listy bez głowy:

Listing 10.173. Rozwiązanie zadania 7.3.20 w języku C/C++

```
void zeruj(struct element* Lista){
    while(Lista!=NULL){
        Lista->i=0;
        Lista=Lista->next;
    }
}
```

Wersja dla listy z głową:

Listing 10.174. Rozwiązanie zadania 7.3.20 w języku C/C++

```
void zeruj(struct element* Lista){
    while(Lista->next!=NULL){
        Lista=Lista->next;
        Lista->i=0;
    }
}
```

Prostym sposobem uzyskania rozwiązania zadania dla listy wskaźnikowej z głową jest dodanie jednego wiersza do rozwiązania dedykowanego dla listy wskaźnikowej bez głowy:

Listing 10.175. Rozwiązanie zadania 7.3.20 w języku C/C++

```
void zeruj(struct element* Lista){
    Lista=Lista->next;
    while(Lista!=NULL){
        Lista->i=0;
        Lista=Lista->next;
    }
}
```

Zadanie 7.3.23 Wersja dla listy bez głowy:

Listing 10.176. Rozwiązań zadania 7.3.23 w języku C/C++

```

struct trojka{
    unsigned int a,b,c;
    struct trojka * next;
};

bool tr(struct trojka * e){
    if (e->a*e->a + e->b*e->b == e->c*e->c)
        return true;
    else
        return false;
}

struct trojka * pitagoras(struct trojka* Lista){
    struct trojka * pom, *pom2;
    while ((Lista!=NULL)&&(!tr(Lista))){
        pom=Lista;
        Lista=Lista->next;
        free(pom);
    }
    if (Lista==NULL)
        return NULL;
    pom=Lista;
    while(pom->next!=NULL){
        if (tr(pom->next))
            pom=pom->next;
        else{
            pom2=pom->next;
            pom->next=pom2->next;
            free(pom2);
        }
    }
    return Lista;
}

```

Rozwiązań dla listy z głową różni się tylko funkcją pitagoras:

Listing 10.177. Rozwiązań zadania 7.3.23 w języku C/C++

```

void pitagoras(struct trojka* Lista){
    struct trojka * pom;
    while(Lista->next!=NULL){
        if (tr(Lista->next))
            Lista=Lista->next;
        else{
            pom=Lista->next;
            Lista->next=pom->next;
            free(pom);
        }
    }
}

```

```
    }
}
```

Zadanie 7.3.24 Wersja dla listy bez głowy:

Listing 10.178. Rozwiązanie zadania 7.3.24 w języku C/C++

```
int suma (struct element* Lista){
    int sum=0;
    while(Lista!=NULL){
        sum+=Lista->i ;
        Lista=Lista->next ;
    }
    return sum;
}
```

Wersja dla listy z głową jest podobna:

Listing 10.179. Rozwiązanie zadania 7.3.24 w języku C/C++

```
int suma (struct element* Lista){
    int sum=0;
    Lista=Lista->next ;
    while(Lista!=NULL){
        sum+=Lista->i ;
        Lista=Lista->next ;
    }
    return sum;
}
```

Zadanie 7.3.25 Wersja dla listy bez głowy:

Listing 10.180. Rozwiązanie zadania 7.3.25 w języku C/C++

```
int minimum (struct element* Lista){
    int min=Lista->i ;
    while(Lista!=NULL){
        if (Lista->i<min)
            min=Lista->i ;
        Lista=Lista->next ;
    }
    return min;
}
```

Wersja dla listy z głową jest podobna:

Listing 10.181. Rozwiązanie zadania 7.3.25 w języku C/C++

```
int minimum (struct element* Lista){
    int min=Lista->next->i ;
    while(Lista->next!=NULL){
        Lista=Lista->next ;
        if (Lista->i<min)
            min=Lista->i ;
    }
    return min;
}
```

Zadanie 7.3.26 Wersja dla listy bez głowy:

Listing 10.182. Rozwiązanie zadania 7.3.26 w języku C/C++

```
struct element * minimum (struct element* Lista){
    struct element * min=Lista ;
    while(Lista!=NULL){
        if (Lista->i<min->i)
            min=Lista ;
        Lista=Lista->next ;
    }
    return min;
}
```

Wersja dla listy z głową jest podobna.

Zadanie 7.3.27 Wersja dla listy bez głowy:

Listing 10.183. Rozwiązanie zadania 7.3.27 w języku C/C++

```
struct element * minimum (struct element* Lista){
    struct element * min=NULL, *pom;
    if ((Lista==NULL) || (Lista->next==NULL))
        return NULL;
    pom=Lista ;
    while(pom->next!=NULL){
        if (((min==NULL)&&(pom->next->i<Lista->i)) ||
            ((min!=NULL)&&(pom->next->i<min->next->i)))
            min=pom;
        pom=pom->next ;
    }
    return min;
}
```

Wersja dla listy z głową jest prostsza:

Listing 10.184. Rozwiązanie zadania 7.3.27 w języku C/C++

```
struct element * minimum (struct element* Lista){
```

```

struct element * min=Lista;
while(Lista->next!=NULL){
    if (Lista->next->i<min->next->i )
        min=Lista ;
    Lista=Lista->next ;
}
return min;
}

```

Zadanie 7.3.28 Wersja dla listy bez głowy:

Listing 10.185. Rozwiązanie zadania 7.3.28 w języku C/C++

```

int maks_rozn (struct element* Lista){
    struct element * pom;
    int maks=abs(Lista->i-Lista->next->i);
    for (;Lista->next!=NULL;Lista=Lista->next)
        for(pom=Lista->next ;pom!=NULL;pom=pom->next)
            if ( abs(pom->i-Lista->i)<maks)
                maks=abs(pom->i-Lista->i );
    return maks;
}

```

Powyzsze rozwiązanie polega na przeszukaniu wszystkich par elementów listy. Efektywniejsze obliczeniowo byłoby znalezienie elementów o najmniejszej i największej wartości pola **i**, i zwrócenie ich różnicy.

Wersja dla listy z głową jest podobna.

Zadanie 7.3.29 Wersja dla listy bez głowy:

Listing 10.186. Rozwiązanie zadania 7.3.29 w języku C

```

struct element * kopijuj(struct element* Lista){
    struct element * kopia ,*pom;
    if (Lista==NULL)
        return NULL;
    kopia=malloc(sizeof(struct element));
    pom=kopia ;
    pom->i=Lista->i ;
    Lista=Lista->next ;
    while(Lista!=NULL){
        pom->next=malloc(sizeof(struct element));
        pom=pom->next ;
        pom->i=Lista->i ;
        Lista=Lista->next ;
    }
    pom->next=NULL;
    return kopia ;
}

```

Uważny czytelnik łatwo przerobi powyższe rozwiązanie na rozwiązanie w języku C++ oraz na rozwiązanie dla listy wskaźnikowej z głową.

Zadanie 7.3.30

Listing 10.187. Rozwiązanie zadania 7.3.30 w języku C/C++

```
struct element * sklej(struct element* Lista1 ,
                      struct element * Lista2){
    struct element * pom;
    if (Lista1==NULL)
        return Lista2;
    pom=Lista1;
    while(pom->next!=NULL)
        pom=pom->next;
    pom->next=Lista2;
    return Lista1;
}
```

Zadanie 7.3.31 Wersja dla listy bez głowy:

Listing 10.188. Rozwiązanie zadania 7.3.31 w języku C/C++

```
struct element * odwroc(struct element* Lista){
    struct element * pom1, *pom2;
    if ((Lista==NULL) || (Lista->next==NULL))
        return Lista;
    pom1=Lista->next;
    pom2=pom1->next;
    Lista->next=NULL;
    pom1->next=Lista;
    while(pom2!=NULL){
        Lista=pom1;
        pom1=pom2;
        pom2=pom2->next;
        pom1->next=Lista;
    }
    return pom1;
}
```

Wersja dla listy z głową jest podobna.

Zadanie 7.3.32

Listing 10.189. Rozwiązanie zadania 7.3.32 w języku C/C++

```
struct element * polacz(struct element* Lista1 ,
                        struct element* Lista2){
    struct element * pom, *pom2;
    if (Lista1==NULL)
```

```

return NULL;
pom=pom2=Lista1 ;
Lista1=Lista1->next ;
pom2->next=Lista2 ;
pom2=Lista2 ;
Lista2=Lista2->next ;
while(Lista1!=NULL){
    pom2->next=Lista1 ;
    pom2=Lista1 ;
    Lista1=Lista1->next ;
    pom2->next=Lista2 ;
    pom2=Lista2 ;
    Lista2=Lista2->next ;
}
return pom;
}

```

Zadanie 7.3.33 Wersja dla listy bez głowy:

Listing 10.190. Rozwiązanie zadania 7.3.33 w języku C/C++

```

struct element * przesun(struct element* Lista){
    struct element * pom;
    if ((Lista==NULL) || (Lista->next==NULL))
        return Lista;
    pom=Lista;
    while(Lista->next->next!=NULL)
        Lista=Lista->next;
        Lista->next->next=pom;
        pom=Lista->next;
        Lista->next=NULL;
    return pom;
}

```

Wersja dla listy z głową jest podobna.

Zadanie 7.3.34 Wersja dla listy bez głowy:

Listing 10.191. Rozwiązanie zadania 7.3.34 w języku C

```

bool wystepuje(struct element * Lista , int i){
    while(Lista!=NULL){
        if (Lista->i==i)
            return true;
        Lista=Lista->next;
    }
    return false;
}

struct element * powtorzone(struct element* Lista1 ,

```

```

struct element* Lista2){
struct element * Lista3=NULL, *pom,*pom2;
if ((Lista1==NULL)||(Lista2==NULL))
    return NULL;
pom=Lista1;
while(pom!=NULL){
    if ((wystepuje(Lista2 , pom->i))
        &&(!wystepuje(Lista3 , pom->i))){
        if (Lista3==NULL)
            pom2=Lista3=malloc (sizeof (struct element));
        else{
            pom2->next=malloc (sizeof (struct element));
            pom2=pom2->next ;
        }
        pom2->i=pom->i ;
    }
    pom=pom->next ;
}
pom2->next=NULL;
return Lista3;
}

```

Uważny czytelnik szybko stworzy wersję dla listy z głową. Rozwiązanie w języku C++ jest analogiczne.

Zadanie 7.3.35 Tym razem zaprezentowany zostanie prosty algorytm sortujący dla listy z głową:

Listing 10.192. Rozwiązań zadania 7.3.35 w języku C/C++

```

struct element * minimum (struct element* Lista){
    struct element * min=Lista;
    while(Lista->next!=NULL){
        if (Lista->next->i<min->next->i )
            min=Lista ;
        Lista=Lista->next ;
    }
    return min;
}

void sort (struct element* Lista){
    struct element * pom,*pom2;
    while(Lista->next!=NULL){
        pom=minimum(Lista );
        if (pom!=Lista ){
            pom2=pom->next;
            pom->next=pom2->next ;
            pom2->next=Lista->next ;
            Lista->next=pom2 ;
        }
    }
}

```

```

        Lista=Lista->next;
    }
}

```

Podobnie jak miało to miejsce w przypadku rozwiązania zadania 4.212d dotyczącego sortowania elementów tablic jednowymiarowych, także tym razem zostanie przedstawione nieco bardziej skomplikowany w implementacji ale też bardziej efektywny algorytm sortujący (w wersji dla list bez głowy):

Listing 10.193. Rozwiązanie zadania 7.3.35 w języku C/C++

```

struct element * merge(struct element* Lista1 ,
                      struct element *Lista2){
    struct element * pom, *pom2;
    if (Lista1->i<Lista2->i){
        pom=pom2=Lista1;
        Lista1=Lista1->next;
    }
    else{
        pom=pom2=Lista2;
        Lista2=Lista2->next;
    }
    while((Lista1!=NULL)&&(Lista2!=NULL))
        if (Lista1->i<Lista2->i){
            pom2->next=Lista1;
            Lista1=Lista1->next;
            pom2=pom2->next;
        }
        else{
            pom2->next=Lista2;
            Lista2=Lista2->next;
            pom2=pom2->next;
        }
    if (Lista1!=NULL)
        pom2->next=Lista1;
    else
        pom2->next=Lista2;
    return pom;
}

struct element * mergesort(struct element* Lista){
    struct element * pom1, *pom2, *l1, *l2;
    unsigned int i=0;
    if ((Lista==NULL) || (Lista->next==NULL))
        return Lista;
    l1=pom1=Lista;
    l2=pom2=Lista->next;
    Lista=Lista->next->next;
    while(Lista!=NULL){

```

```

if ( i%2){
    pom1->next=Lista ;
    pom1=pom1->next ;
}
else{
    pom2->next=Lista ;
    pom2=pom2->next ;
}
i++;
Lista=Lista->next ;
}
pom1->next=NULL;
pom2->next=NULL;
l1=mergesort( l1 );
l2=mergesort( l2 );
l1=merge( l1 , l2 );
return l1 ;
}

```

10.11. Rozwiązania do zadań z rozdziału 8.2

Zadanie 8.2.1 Program w wersji dla języka C:

Listing 10.194. Rozwiązanie zadania 8.2.1 w języku C

```

FILE * otworz(char * plik){
    return fopen( plik , "r" );
}

```

Program w wersji dla języka C++:

Listing 10.195. Rozwiązanie zadania 8.2.1 w języku C++

```

fstream* otworz(char * plik){
    return new fstream( plik , ios :: in );
}

```

Warto zapamiętać, że w powyższym rozwiążaniu funkcja **otworz** nie mogłaby zwrócić wartości strumienia **fstream** ze względu na to, że nie posiada on konstruktora kopiującego, ani nie można zdefiniować dla niego operatora przypisania. Funkcja **otworz** mogłaby zwrócić referencję do strumienia, ale w ten sposób program straciłby wskaźnik do niego i nie mógłby go w przyszłości usunąć z pamięci.

Zadanie 8.2.2 Rozwiązanie w wersji dla języka C:

Listing 10.196. Rozwiązań zadania 8.2.2 w języku C

```
void wypisz(FILE * plik){
    char c;
    while(fscanf(plik , "%c",&c)==1)
        printf("%c",c);
    fclose(plik);
}
```

Rozwiązań w wersji dla języka C++:

Listing 10.197. Rozwiązań zadania 8.2.2 w języku C++

```
void wypisz(fstream& plik){
    char c;
    while(! plik.eof()){
        plik.get(c);
        if (! plik.eof())
            cout<<c;
    }
    plik.close();
}
```

Zadanie 8.2.3 Rozwiązań w wersji dla języka C i znaków typu **char**:

Listing 10.198. Rozwiązań zadania 8.2.3 w języku C

```
void wypisz(char * sciezka){
    FILE * plik=fopen(sciezka , "r");
    char c;
    while(fscanf(plik , "%c",&c)==1)
        if (!isspace(c))
            printf("%c",c);
    fclose(plik);
}
```

Rozwiązań w wersji dla języka C i znaków typu **wchar_t**

Listing 10.199. Rozwiązań zadania 8.2.3 w języku C

```
void wypisz(char * sciezka){
    FILE * plik=fopen(sciezka , "r");
    wchar_t c;
    while(fscanf(plik , "%lc",&c)==1)
        if (!iswspace(c))
            printf("%lc",c);
    fclose(plik);
}
```

Rozwiązań w wersji dla języka C++ i znaków typu **char**:

Listing 10.200. Rozwiązanie zadania 8.2.3 w języku C++

```
void wypisz(char * sciezka){
    fstream plik(sciezka, ios::in);
    char c;
    while(! plik.eof()){
        plik>>c;
        if (! plik.eof())
            cout<<c;
    }
    plik.close();
}
```

W programie 10.200 nie ma konieczności zamykania pliku ręcznie, tak jak jest to zrobione. Zrobiłby to destruktor obiektu `plik` przy wyjściu z funkcji `wypisz`. Ręczne zamykanie pliku w takiej sytuacji ma sens wtedy, kiedy do zakończenia działania funkcji zostało jeszcze dużo czasu lub gdy program ma śledzić, czy przy zamykaniu pliku nie wystąpił błąd.

Rozwiązanie w wersji dla języka C++ i znaków typu `wchar_t`:

Listing 10.201. Rozwiązanie zadania 8.2.3 w języku C++

```
void wypisz(char * sciezka){
    wfstream plik(sciezka, ios::in);
    wchar_t c;
    while(! plik.eof()){
        plik>>c;
        if (! plik.eof())
            wcout<<c;
    }
}
```

Warto zapamiętać różnicę pomiędzy metodą `get` a operatorem `>>`. Metoda `get` wyłuskuje kolejne znaki, natomiast operator `>>` pomija białe znaki.

Zadanie 8.2.4 Rozwiązanie w wersji dla języka C i znaków typu `char`:

Listing 10.202. Rozwiązanie zadania 8.2.4 w języku C

```
int wystapien(char * sciezka, char c){
    char pom;
    int licz=0;
    FILE * plik=fopen(sciezka, "r");
    while(fscanf(plik, "%c",&pom)==1)
        if (c==pom)
            licz++;
    fclose(plik);
    return licz;
}
```

Rozwiązanie w wersji dla języka C++ i znaków typu `char`:

Listing 10.203. Rozwiązanie zadania 8.2.4 w języku C++

```
int wystapien(const char * sciezka , char c ){
    char pom;
    int licz=0;
    fstream plik(sciezka , ios::in);
    while(!plik.eof()){
        plik.get(pom);
        if ((!plik.eof())&&(c==pom))
            licz++;
    }
    return licz ;
}
```

Rozwiązanie dla znaków typu `wchar_t` jest analogiczne.

Zadanie 8.2.9 Rozwiązanie w wersji dla języka C i znaków typu `char`:

Listing 10.204. Rozwiązanie zadania 8.2.9 w języku C

```
int porownaj(char * sciezka1 , char * sciezka2){
    char c1,c2;
    FILE * plik1=fopen(sciezka1 , "r");
    FILE * plik2=fopen(sciezka2 , "r");
    fscanf(plik1 , "%c",&c1);
    fscanf(plik2 , "%c",&c2);
    while ((!feof(plik1))&&(!feof(plik2))){
        if (c1!=c2)
            return 0;
        fscanf(plik1 , "%c",&c1);
        fscanf(plik2 , "%c",&c2);
    }
    if ((!feof(plik1))||(!feof(plik2))){
        fclose( plik1 );
        fclose( plik2 );
        return 0;
    } else{
        fclose( plik1 );
        fclose( plik2 );
        return 1;
    }
}
```

Rozwiązań w wersji dla języka C++ i znaków typu char:

Listing 10.205. Rozwiązań zadania 8.2.9 w języku C++

```

int porownaj(char * sciezka1, char * sciezka2){
    char c1='0',c2='0';
    fstream plik1(sciezka1, ios::in);
    fstream plik2(sciezka2, ios::in);
    while((!plik1.eof())&&(!plik2.eof())){
        plik1.get(c1);
        plik2.get(c2);
        if (c1!=c2)
            return 0;
    }
    if ((!plik1.eof())||(!plik2.eof()))
        return 0;
    else
        return 1;
}

```

Rozwiązań dla znaków typu wchar_t jest analogiczne.

Zadanie 8.2.10 Rozwiązań w wersji dla języka C i znaków typu char:

Listing 10.206. Rozwiązań zadania 8.2.10 w języku C

```

int porownaj(char * sciezka1, char * sciezka2){
    char c1,c2;
    FILE * plik1=fopen(sciezka1, "r");
    FILE * plik2=fopen(sciezka2, "r");
    while ((fscanf(plik1, "%c",&c1)==1)&&(isspace(c1)));
    while ((fscanf(plik2, "%c",&c2)==1)&&(isspace(c2)));
    while ((!feof(plik1))&&(!feof(plik2))){
        if (c1!=c2)
            return 0;
        while ((fscanf(plik1, "%c",&c1)==1)&&(isspace(c1)));
        while ((fscanf(plik2, "%c",&c2)==1)&&(isspace(c2)));
    }
    if ((!feof(plik1))||(!feof(plik2))){
        fclose(plik1);
        fclose(plik2);
        return 0;
    } else{
        fclose(plik1);
        fclose(plik2);
        return 1;
    }
}

```

Rozwiązań w wersji dla języka C++ i znaków typu char:

Listing 10.207. Rozwiązanie zadania 8.2.10 w języku C++

```

int porownaj(char * sciezka1, char * sciezka2){
    char c1='0',c2='0';
    fstream plik1(sciezka1, ios::in);
    fstream plik2(sciezka2, ios::in);
    while((!plik1.eof())&&(!plik2.eof())){
        plik1>>c1;
        plik2>>c2;
        if (c1!=c2)
            return 0;
    }
    if ((!plik1.eof())||(!plik2.eof()))
        return 0;
    else
        return 1;
}

```

Rozwiązanie dla typu znaków **wchar_t** jest analogiczne.

Zadanie 8.2.11 Rozwiązanie dla języka C:

Listing 10.208. Rozwiązanie zadania 8.2.11 w języku C

```

FILE * otworz(char * plik){
    return fopen(plik, "a");
}

```

Rozwiązanie dla języka C++:

Listing 10.209. Rozwiązanie zadania 8.2.11 w języku C++

```

fstream* otworz(char * plik){
    return new fstream(plik, ios::out|ios::app);
}

```

Zadanie 8.2.12 Rozwiązania w wersji dla języka C i znaków typu **char**:

Listing 10.210. Rozwiązanie zadania 8.2.12 w języku C

```

void wczytaj(FILE * plik, int n){
    int i;
    char tab[100];
    for(i=0;i<n;i++){
        fgets(tab,100, stdin);
        fprintf(plik, "%s", tab);
    }
    fclose(plik);
}

```

Wadą powyższego rozwiązania jest fakt, że linie mogą mieć co najwyżej 99 znaków. Poniższe rozwiązanie jest pozbawione tej wady:

Listing 10.211. Rozwiązanie zadania 8.2.12 w języku C

```
void wczytaj(FILE * plik, int n){
    int i=0;
    char c;
    while(i<n){
        scanf("%c",&c);
        if (c=='\n')
            i++;
        if (i<n)
            fprintf(plik,"%c",c);
    }
    fclose(plik);
}
```

Rozwiązanie w wersji dla języka C++ i znaków typu **char**:

Listing 10.212. Rozwiązanie zadania 8.2.12 w języku C++

```
void wczytaj(fstream& plik, int n){
    int i;
    string s;
    for(i=0;i<n;i++){
        getline(cin,s);
        plik<<s<<endl;
    }
    plik.close();
}
```

W programie 10.212 trzeba zamknąć plik ręcznie, gdyż zmienna **plik** nie jest lokalnym obiektem funkcji **wczytaj**, ale referencją do obiektu utworzonego poza tą funkcją. W takiej sytuacji zakończenie działania funkcji **wczytaj** nie spowoduje zamknięcia pliku skojarzonego ze zmienną **plik**.

Rozwiązanie dla typu **wchar_t** jest analogiczne.

Zadanie 8.2.13 Program w wersji dla języka C:

Listing 10.213. Rozwiązanie zadania 8.2.13 w języku C

```
void przepisz(FILE * plik1, FILE * plik2){
    char c;
    while(fscanf(plik1,"%c",&c)==1)
        fprintf(plik2,"%c",c);
}
```

Program w wersji dla języka C++:

Listing 10.214. Rozwiązanie zadania 8.2.13 w języku C++

```
void przepisz(fstream& plik1 , fstream& plik2){
    char c;
    while(! plik1.eof()){
        plik1.get(c);
        if (! plik1.eof())
            plik2<<c;
    }
}
```

Zadanie 8.2.14 Ponieważ w teści nie jest zaznaczone, że przepisywany plik jest tekstowy, to należy go przepisać w trybie binarnym:

Najpierw wersja dla języka C:

Listing 10.215. Rozwiązanie zadania 8.2.14 w języku C

```
void przepisz(char * sciezka1 , char * sciezka2){
    FILE * plik1=fopen(sciezka1 , "rb");
    FILE * plik2=fopen(sciezka2 , "wb");
    char c[100];
    int n=100;
    while(n==100){
        n=fread(c,1,100 ,plik1 );
        fwrite(c,1 ,n ,plik2 )
    }
    fclose(plik1 );
    fclose(plik2 );
}
```

Wersja dla języka C++:

Listing 10.216. Rozwiązanie zadania 8.2.14 w języku C++

```
void przepisz(char * sciezka1 , char * sciezka2){
    fstream plik1(sciezka1 , ios::in|ios::bin);
    fstream plik2(sciezka2 , ios::out|ios::bin);
    char c[100];
    int n=100;
    while(n==100){
        plik1.read(c,100 );
        n=plik1.gcount();
        plik2.write(c,n);
    }
}
```

Zadanie 8.2.17 Wersja w języku C

Listing 10.217. Rozwiązanie zadania 8.2.17 w języku C

```
void zapisz(char * sciezka , int ** tab , int n , int m){
    FILE * plik=fopen(sciezka , "wb");
    int i;
    for(i=0;i<n;i++)
        fwrite(tab[i],sizeof(int) ,m, plik);
    fclose(plik);
}
```

Wersja w języku C++:

Listing 10.218. Rozwiązanie zadania 8.2.17 w języku C++

```
void zapisz(const char * sciezka , int ** tab , int n , int m){
    fstream plik(sciezka , ios::out | ios::binary);
    int i;
    for(i=0;i<n;i++)
        plik.write(reinterpret_cast<char *>(tab[i]) ,
                    sizeof(int)*m);
}
```

W powyższym rozwiązaniu konieczne było rzutowanie typu **int*** na typ **char*** gdyż pierwszy argument metody **write** klasy **fstream** musi być wskaźnikiem na typ **char**.

Zadanie 8.2.18 Wersja w języku C

Listing 10.219. Rozwiązanie zadania 8.2.18 w języku C

```
void wczytaj(char * sciezka , int ** tab , int n , int m){
    FILE * plik=fopen(sciezka , "rb");
    int i;
    for(i=0;i<n;i++)
        fread(tab[i],sizeof(int) ,m, plik);
    fclose(plik);
}
```

Rozwiązanie w języku C++

Listing 10.220. Rozwiązanie zadania 8.2.18 w języku C

```
void wczytaj(const char* sciezka ,int** tab ,int n ,int m){
    fstream plik(sciezka , ios::in | ios::binary);
    int i;
    for(i=0;i<n;i++)
        plik.read(reinterpret_cast<char *>(tab[i]) ,
```

```
    }  
    sizeof(int)*m);
```

10.12. Rozwiązania do zadań z rozdziału 9.2

Zadanie 9.2.1

Listing 10.221. Rozwiązanie zadania 9.2.1 w języku C/C++

```
#define suma(a,b,c) (a+b+c)
```

Dobrym nawykiem jest umieszczanie w nawiasach definicji makr (czyli tak, jak w powyższym rozwiązaniu $(ab+c)+$, a nie $ab+c+$).

Zadanie 9.2.3

Listing 10.222. Rozwiązanie zadania 9.2.3 w języku C/C++

```
#define wiekszy(a,b) \  
    if(a>b) printf("%d",a); else printf("%d",b)
```

Dyrektywy preprocesora powinny mieścić się w pojedynczej linii. Jeżeli z jakiegoś powodu chce się podzielić dyrektywę na kilka linii, należy w miejscach przejścia do nowej linii, wstawić lewy ukośnik, tak jak to ma miejsce w programie 10.222.

Zadanie 9.2.4

Listing 10.223. Rozwiązanie zadania 9.2.4 w języku C/C++

```
#define parzysta(a) (a%2==0)?1:0
```

Zadanie 9.2.6

Listing 10.224. Rozwiązanie zadania 9.2.6 w języku C/C++

```
#define najwiersza(a,b,c) if ((a>=b)&&(a>=c))\  
    printf("%d",a); else if (b>=c) printf("%d",b);\  
    else printf("%d",c)
```

Zadanie 9.2.7

Listing 10.225. Rozwiązanie zadania 9.2.7 w języku C/C++

```
#define najwiersza(a,b,c) ((a>=b)&&(a>=c))?a:((b>=c)?b:c)
```

Zadanie 9.2.8

Listing 10.226. Rozwiązań do zadania 9.2.8 w języku C/C++

```
#define petla(x,y) for(x=0;x<y;x++)
```

Zadanie 9.2.10

Listing 10.227. Rozwiązań do zadania 9.2.10 w języku C/C++

```
#define zeruj(x) x=0
```

10.13. Rozwiązań do zadań z rozdziału 9.3

Zadanie 9.3.1 Plik głównego programu:

Listing 10.228. Plik glowny.c

```
#include <stdio.h>
#include "pierwiastek.c"

int main(){
    double a,b,c,delta;
    printf("Podaj współczynniki równania kwadratowego");
    scanf("%lf %lf %lf",&a,&b,&c);
    delta=b*b-4*a*c;
    if (delta>0)
        printf("x1=%f x2=%f",(-b-pierw(delta))/(2*a),
               (-b+pierw(delta))/(2*a));
    else if (delta==0)
        printf("x=%f",(-b)/(2*a));
    else
        printf("Brak rozwiązań");
    return 0;
}
```

Plik z funkcją liczącą pierwiastek:

Listing 10.229. Plik pierwiastek.c

```
double pierw(double n){
    float p,k,sr;
    p=0;
    k=n;
    sr=(p+k)/2;
    while((sr!=p)&&(sr!=k)){
        if ((sr*sr)>n)
            k=sr;
```

```

    else
        p=sr ;
        sr=(p+k)/2;
    }
    return sr;
}

```

Aby skompilować powyższy program, należy oba pliki umieścić w jednym katalogu i skompilować plik głowny.c. Plik pierwiastek.c zostanie w całości wklejony do pliku głownego.c za sprawą dyrektywy `include`.

Rozwiążanie dla języka C++ jest analogiczne

Zadanie 9.3.3 Plik głównego programu:

Listing 10.230. Plik głowny.c

```

#include <stdio.h>
#include "pierwiastek.h"

int main(){
    double a,b,c,delta;
    printf("Podaj współczynniki równania kwadratowego");
    scanf("%lf %lf %lf",&a,&b,&c);
    delta=b*b-4*a*c;
    if (delta>0)
        printf("x1=%f x2=%f",(-b-pierw(delta))/(2*a),
               (-b+pierw(delta))/(2*a));
    else if (delta==0)
        printf("x=%f",(-b)/(2*a));
    else
        printf("Brak rozwiązań");
    return 0;
}

```

Plik nagłówkowy:

Listing 10.231. Plik pierwiastek.h

```

#ifndef _pierw_
#define _pierw_
double pierw(double);
#endif

```

I plik z funkcją liczącą pierwiastek:

Listing 10.232. Plik pierwiastek.c

```
#include "pierwiastek.h"
```

```
double pierw(double n){  
    float p,k,sr;  
    p=0;  
    k=n;  
    sr=(p+k) / 2;  
    while(( sr!=p)&&(sr!=k)){  
        if (( sr*sr )>n)  
            k=sr ;  
        else  
            p=sr ;  
        sr=(p+k) / 2;  
    }  
    return sr;  
}
```

W pliku **pieriwastek.h** umieszczone zostały dyrektywy preprocesora zabezpieczające przed wielokrotnym dołączaniem tego samego nagłówka. Nie jest to konieczne w powyższym przypadku, ale jest to standardowa praktyka przy tworzeniu bibliotek. Podobnie standardową praktyką jest dołączanie do plików ze źródłami bibliotek ich plików nagłówkowych. Dzięki temu ewentualne błędy w plikach nagłówkowych są wykrywane już na etapie komplikacji bibliotek.

Aby skompilować powyższy program, należy najpierw oddziennie skompliować piliki **glowny.c** i **pieriwastek.c**, a następnie je połączyć. Przy użyciu kompilatora gcc wygląda to następująco:

```
gcc pierwiastek.c -c -o pierwiatek.o  
gcc glowny.c -c -o glowny.o  
gcc glowny.o pierwiastek.o -o glowny
```

Rozwiązanie dla języka C++ jest analogiczne.

Zadanie 9.3.4 W pliku **Makefile** zapisuje się potrzebne do wykonania operacje w odwrotnej kolejności (czyli jako pierwszą wymienia się operację, która ma być wykonana jako pierwsza):

Listing 10.233. plik Makefile

```
glowny: glowny.o pierwiastek.o  
        gcc glowny.o pierwiastek.o -o glowny  
  
glowny.o: glowny.c pierwiastek.h  
        gcc glowny.c -c -o glowny.o  
  
pieriastek.o: pierwiastek.h pierwiastek.c  
        gcc pierwiastek.c -c -o pierwiastek.o
```

W powyższym rozwiązaniu wśród plików potrzebnych do stworzenia plików `glowny.o` i `pierwiastek.o` wymieniony został także plik `pierwiastek.h`, który nie jest parametrem w komplikacji. Dzięki temu program make, śledzi zmiany także w pliku nagłówkowym.

W pliku `Makefile` nie trzeba wymieniać wszystkich komend potrzebnych do komplikacji programu. Wystarczy wypisać zależności. Poniżej skrócona wersja pliku `Makefile`:

Listing 10.234. plik Makefile, krótsza wersja

```
glowny: glowny.o pierwiastek.o
        gcc glowny.o pierwiastek.o -o glowny

glowny.o: glowny.c pierwiastek.h
pierwiastek.o: pierwiastek.h pierwiastek.c
```

Rozwiązanie dla języka C++ jest analogiczne. Wystarczy podmienić `gcc` na `g++` i zmienić rozszerzenia nazw plików.

Zadanie 9.3.7 Plik głównej części programu:

Listing 10.235. Plik `glowny.c`

```
#include <stdio.h>
#include "zespolone.h"
#include "arytmetyka.h"

int main(){
    zespolone suma, wczyt;
    int i;
    printf("Ile liczb zespolonych chcesz zsumowac?");
    scanf("%d",&i);
    while(i>0){
        wczyt=wczytaj();
        suma=dodaj(suma, wczyt);
        i--;
    }
    wypisz(suma);
    return 0;
}
```

Plik nagłówkowy biblioteki `zespolone`:

Listing 10.236. Plik `zespolone.h`

```
#ifndef _zespolone_
#define _zespolone_

typedef struct zesp{ double r,u;} zespolone;
```

```
zespolone wczytaj();  
void wypisz(zespolone);
```

#endif

Plik główny biblioteki zespolone:

Listing 10.237. Plik zespolone.c

```
#include <stdio.h>
#include "zespolone.h"

zespolone wczytaj(){
    zespolone z;
    printf("Podaj \u0142\u0142\u0144czesc \u0142rzeczywista \u0142wczytywanej \u0142liczby");
    scanf("%lf",&(z.r));
    printf("Podaj \u0142\u0142\u0144czesc \u0142urojona \u0142wczytywanej \u0142liczby");
    scanf("%lf",&(z.u));
    return z;
}

void wypisz(zespolone z){
    printf("Czesc \u0142rzeczywista \u0142ma \u0142wartosc %f \u0142a \u0142czesc \u0142urojona %f",
           z.r,z.u);
}
```

Plik nagłówkowy biblioteki z funkcjami arytmetycznymi:

Listing 10.238. Plik arytmetyka.h

```
#ifndef _pierw_
#define _arytmetyka_

#include "zespolone.h"

zespolone dodaj(zespolone, zespolone);
zespolone pomnoz(zespolone, zespolone);

#endif
```

Plik biblioteki z funkcjami arytmetycznymi:

Listing 10.239 Plik arytmetyka.cs

```
#include "zespolone.h"
#include "arytmetyka.h"

zespolone dodaj(zespolone a, zespolone b){
```

```

zespolone z;
z.r=a.r+b.r;
z.u=a.u+b.u;
return z;
}

zespolone pomnoz(zespolone a, zespolone b){
    zespolone z;
    z.r=a.r*b.r-a.u*b.u;
    z.u=a.r*b.u+a.u*b.r;
    return z;
}

```

W plikach biblioteki z operacjami arytmetycznymi na liczbach zespolonych dyrektywą `include` został dołączony plik `zespolone.h` ze względu na zdefiniowany w nim typ `zespolone`.

Rozwiążanie w języku C++ jest analogiczne.

Zadanie 9.3.8

Listing 10.240. plik Makefile, krótsza wersja

```

glowny: glowny.o zespolone.o arytmetyka.o
        gcc glowny.o pierwiastek.o arytmetyka.o -o glowny

glowny.o: glowny.c zespolone.h arytmetyka.h
arytmetyka.o: arytmetyka.h zespolone.h arytmetyka.c
zespolone.o: zespolone.h zespolone.c

```

Zadanie 9.3.11 Plik głównej części programu:

Listing 10.241. Plik glowny.c

```

#include <stdio.h>
#include "dane.h"
#include "statystyka.h"

int main(){
    int i;
    printf("Dane ilu osob chcesz wczytac?");
    scanf("%d",&i);
    while(i>0){
        wczytaj();
        i--;
    }
    printf("%f",srednia());
    return 0;
}

```

Plik nagłówkowy biblioteki `dane`:

Listing 10.242. Plik dane.h

```
#ifndef _dane_
#define _dane_

typedef struct os {
    char imie[15];
    nazwisko[30];
    int wiek;
} osoba;

void wczytaj();
void wypisz(osoba);

#endif
```

Plik główny biblioteki `dane`:

Listing 10.243. Plik dane.c

```
#include <stdio.h>
#include <stdlib.h>
#include "dane.h"

int liczba=0;
int pojemnosc=0;
osoba * tab=NULL;

void wczytaj(){
    if (liczba==pojemnosc){
        osoba * pom=tab;
        tab=malloc(2*(pojemnosc+1)*sizeof(osoba));
        int i;
        for(i=0;i<pojemnosc; i++)
            tab[i]=pom[i];
        if (pom!=NULL)
            free(pom);
        pojemnosc=2*(pojemnosc+1);
    }
    printf("Podaj imię");
    scanf("%s",&(tab[liczba].imie));
    printf("Podaj nazwisko");
    scanf("%s",&(tab[liczba].nazwisko));
    printf("Podaj wiek");
    scanf("%d",&(tab[liczba].wiek));
    liczba++;
}

void wypisz(osoba o){
```

```

    printf("imie : %s\n", nazwisko : %s\n", wiek %d\n",
           o.imie ,o.nazwisko ,o.wiek);

}

```

Plik nagłówkowy biblioteki z funkcjami statystycznymi:

Listing 10.244. Plik statystyka.h

```

#ifndef _pierw_
#define _statystyka_

double srednia();
int minimalny();
int maksymalny();

#endif

```

Plik biblioteki z funkcjami statystycznymi:

Listing 10.245. Plik statystyka.c

```

#include <stdlib.h>
#include "dane.h"
#include "statystyka.h"

extern osoba *tab;
extern int liczba;

double srednia(){
    if (tab==NULL)
        return 0;
    int suma=0,i;
    for(i=0;i<liczba ; i++)
        suma+=tab[ i ].wiek ;
    return (double)suma/liczba ;
}

int minimalny(){
    if (tab==NULL)
        return 0;
    int min=tab[ 0 ].wiek ,i ;
    for(i=1;i<liczba ; i++)
        if (tab[ i ].wiek<min)
            min=tab[ i ].wiek ;
    return min;
}

int maksymalny(){
    if (tab==NULL)

```

```

return 0;
int maks=tab[0].wiek , i ;
for(i=1;i<liczba ; i++)
    if ( tab[i ].wiek>maks)
        maks=tab[i ].wiek ;
return maks;
}

```

Dostęp do zmiennych zadeklarowanych w bibliotece `dane` biblioteka `statystyka` uzyskuje poprzez zadeklarowanie potrzebnych zmiennych ze specyfikatorem `extern`.

Innym sposobem udostępnienia na zewnątrz zmiennych `tab` i `liczba` bez konieczności ich każdorazowego importu przy pomocy modyfikatora `extern` jest umieszczenie kodu odpowiadającego za import zmiennych w pliku nagłówkowym biblioteki `dane`:

Listing 10.246. Plik dane.h

```

#ifndef _dane_
#define _dane_

typedef struct os{
    char imie[15];
    nazwisko[30];
    int wiek;
} osoba;

extern osoba *tab;
extern int liczba;

void wczytaj();
void wypisz(osoba);

#endif

```

W powyższym przypadku zmienne `tab` i `liczba` byłyby dostępne we wszystkich modułach programu używających biblioteki `dane`, niezależnie od tego czy byłyby potrzebne czy nie.

Rozwiązańe w języku C++ jest analogiczne.

Zadanie 9.3.12

Listing 10.247. plik Makefile, krótsza wersja

```

glowny: glowny.o dane.o statystyka.o
    gcc glowny.o dane.o statystyka.o -o glowny

glowny.o: glowny.c dane.h statystyka.h

```

```
statystyka.o: statystyka.h dane.h statystyka.c
dane.o: dane.h dane.c
```

Zadanie 9.3.13 Plik nagłówkowy biblioteki `dane`:

Listing 10.248. Plik dane.h

```
#ifndef _dane_
#define _dane_

typedef struct os{
    char imie[15];
    nazwisko[30];
    int wiek;
} osoba;

void wczytaj();
void wypisz();
int ile();
osoba wczytana(int);

#endif
```

Plik główny biblioteki `dane`:

Listing 10.249. Plik dane.c

```
#include <stdio.h>
#include <stdlib.h>
#include "dane.h"

static liczba=0;
static pojemnosc=0;
static osoba * tab=NULL;

void wczytaj(){
    if (liczba==pojemnosc){
        osoba * pom=tab;
        tab=malloc(2*(pojemnosc+1)*sizeof(osoba));
        int i;
        for(i=0;i<pojemnosc; i++)
            tab[i]=pom[i];
        if (pom!=NULL)
            free(pom);
        pojemnosc=2*(pojemnosc+1);
    }
    printf("Podaj imie");
    scanf("%s",&(tab[liczba].imie));
    printf("Podaj nazwisko");
    scanf("%s",&(tab[liczba].nazwisko));
```

```

    printf("Podaj wiek");
    scanf("%d",&(tab[liczba].wiek));
    liczba++;
}

void wypisz(){
    int i;
    for(i=0;i<liczba;i++)
        printf("imie: %s\nnazwisko : %s\nwiek %d\n",
               tab[i].imie, tab[i].nazwisko, tab[i].wiek);
}

int ile(){
    return liczba;
}

osoba wczytana(int i){
    return tab[i];
}

```

Dzięki użyciu specyfikatora `static` zmienne zadeklarowane w bibliotece dane nie są dostępne poza tą biblioteką (także przy użyciu specyfikatora `extern`).

Rozwiązanie w języka C++ jest podobne.

Zadanie 9.3.15 Rozwiązanie w języku C: Plik nagłówkowy biblioteki `kolo`:

Listing 10.250. Plik kolo.h

```

ifndef _kolo_
#define _kolo_

static double const pi=3.1415;

double pole(double);
double obwod(double);

#endif

```

Gdyby przy deklaracji stałej `pi` pominąć specyfikator `static` to przy próbie komplikacji programu używającego biblioteki `kolo` kompilator zgłosiłby błąd o wielokrotnej definicji stałej `pi`. Plik główny biblioteki `kolo`:

Listing 10.251. Plik kolo.c

```

#include "kolo.h"

double pole(double r){

```

```
    return pi*r*r;
}

double obwod(double r){
    return 2*pi*r;
}
```

Rozwiązanie w języku C++ różni się tylko drobnym szczegółem w pliku nagłówkowym:

Listing 10.252. Plik kolo.h

```
#ifndef _kolo_
#define _kolo_

double const pi=3.1415;

double pole(double);
double obwod(double);

#endif
```

W przeciwieństwie do języka C, stałe w języku C++ są domyślnie statyczne. Użycie w takiej sytuacji specyfikatora `static` jest dozwolone, ale nie wywoła żadnego efektu.

BIBLIOGRAFIA

- [1] Mike Banahan, Declan Brady, Mark Doran, *The C Book, Second Edition*, Addison-Wesley 1991
- [2] Krzysztof Diks, *Wstęp do programowania w języku C*, <http://wazniak.mimuw.edu.pl/>
- [3] Bruce Eckel, *Thinking in C++*. Edycja polska, Helion, Warszawa 2002
- [4] Brian W. Kernighan, Dennis M. Ritchie, *Język ANSI C*, WNT Warszawa 2000, wyd. VI, ISBN 83-204-2620-0
- [5] Bjarne Stroustrup, *Język C++*, WNT, Warszawa 2002
- [6] Kurc C na Wikibooks, <http://pl.wikibooks.org/wiki/C>
- [7] Standard języka C ISO/IEC 9899:1999
- [8] Standard Języka C++ ISO/IEC 14882:2003