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

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UMCS

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: www.esus.pl

ISBN: 978-83-62773-06-0

Spis treści

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 nabyte 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 szczegółowej 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 kompilowane 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 kompilowane przy pomocy polecenia `gcc <nazwa programu>`. Przykładowe programy w C++ były kompilowane przy pomocy polecenia `g++ <nazwa programu>`.

Aby ułatwić posługiwanie się zbiorem zadań wprowadzono następujące oznaczenia:

***** trudne zadanie,
r zadanie rozwiązane 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

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,!,róż)** 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 `2.0e-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 którąkolwiek 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 trój-

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

1.3.9 Napisz program kalkulator, który wykonuje wybraną przez użytkownika operacją 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 $0^2 + 1^2 + 3^2 + ... + n^2$.

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 $2 * 4 * \ldots * n$ dla n parzystych i $2 * 4 * \ldots * (n-1)$ 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 $n * \ldots * m$.

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:

a) wypisuje na standardowe wyjście najmniejszą nieujemną liczbę całkowitą x taką, że $|a| * x^2 + b * x + c > d$.

b) 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.

c) 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 $0! + 1! + \ldots + n!$.

1.4.14 **(*)** Napisz program, który wczytuje ze standardowego wejścia liczbę n i wypisuje na standardowym wyjściu wszystkie trójki pitagorejskie (tj. trójki 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

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 $\lfloor \sqrt{0} \rfloor + \lfloor \sqrt{1} \rfloor + \ldots + \lfloor \sqrt{n} \rfloor$. 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 $\lfloor \sqrt[m]{0} \rfloor + \lfloor \sqrt[m]{1} \rfloor + \ldots + \lfloor \sqrt[m]{n} \rfloor$. 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 równe 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

$$a_0 = a_1 = 1$$

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

.

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

$$a_0 = a_1 = 1$$

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

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

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

$$a_0 = a_1 = a_2 = 1$$

$$\text{oraz dla } k > 2$$

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

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

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

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

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 kompilacji) 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

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 kompilacji 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:
   a) **(r)** nadaje wartość zero wszystkim elementom tablicy `tab`,
   b) **(r)** zapisuje do kolejnych elementów tablicy wartości równe ich indeksom (do komórki o indeksie i funkcja ma zapisywać wartość i),
   c) podwaja wartość wszystkich elementów w tablicy `tab`,
   d) 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:
   a) **(r)** średnią arytmetyczną elementów tablicy `tab`.
   b) sumę elementów tablicy `tab`,
   c) 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 ($n \geqslant 3$) 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:
   a) **(r)** przepisuje zawartość tablicy `tab1` do tablicy `tab2`,
   b) 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:
   a) 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]`),
   b) 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]`),
   c) 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`.
   a) 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`.
   b) 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

Operacje na napisach, nazywanych też łańcuchami lub z języka angielskiego stringami, to jedna z najsłabszych 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 operacji 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 kompilowany i uruchamiany program, używa standardowego kodowania ASCII.

5.2.5 Napisz funkcję `przepisz`, która otrzymuje dwie tablice znaków i przepisuje napis 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 napis w tablicy `nap1` jest krótszy niż n znaków, funkcja powinna po prostu przepisać cały napis. Funkcja powinna zadbać o to, żeby na końcu napisu w tablicy `nap2` znalazł 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 napis "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 napis 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 kompilowany i uruchamiany program, używa standardowego kodowania ASCII.

5.2.9 **(r)** Napisz funkcję `wytnij`, która dostaje jako argumenty napis oraz dwie liczby całkowite `n` i `m`, i wycina z otrzymanego napisu znaki o indeksach od n do m ($n \leqslant m$). Otrzymany w argumencie napis 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 równej 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 znajdującego 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 znajdującego 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 funkcję, 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

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 tablic `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 tablic `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 $i \leqslant j \leqslant k$, 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 $i \leqslant j \leqslant k$, 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 przechowywa-

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

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 zmienną typu `struct trojkat`, i zwraca jako wartość obwód trójkąta przekazanego w argumencie.

7.2.2 **(r,!)** Napisz funkcję, która otrzymuje jako argumenty zmienną `troj1` typu `struct trojkat` zdefiniowanego w zadaniu 7.2.1 oraz zmienną `troj2` wskaźnik na zmienną typu `struct trojkat`, i przepisuje zawartość 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 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_cywilny`. W zależności od płci pole `stan_cywilny` 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_cywilny` 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 wskazywany 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 wskazywanym 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 trójki 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 trójek 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 podpiąć 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 naprzemiennie (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

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 funkcje 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 uproszczania 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

Przed właściwym procesem kompilacji, 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 ułatwiły szukanie błędów w programach, co wcześniej było jedną z dziedzin, w której wykorzystywano dyrektywy preprocesora. Obecnie dyrektywy preprocesora wykorzystuje się niemal wyłącznie do 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 kompilację 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 kompilację 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 wczytanych 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 kompilację 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 kompilację 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 kompilację 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,
— `pomnoz` – 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 `kolo` zawierającą:
— jednoargumentową funkcję `pole` zwracającą pole koła o podanym w argumencie promieniu,
— jednoargumentową funkcję `obwod` zwracającą obwód koła o podanym w argumencie promieniu,
— stałą `pi` przechowującą wartość liczby π. Stała ta powinna być dostępna w programach używających biblioteki `kolo`.

Rozdział 10

Rozwiązania i wskazówki

10.1. Rozwiązania do zadań z rozdziału 1.2

Zadanie 1.2.1

Listing 10.1. Rozwiązanie zad. 1.2.1 w języku C

```c
#include <stdio.h>

int main(){
  int liczba;
  printf("Hello World");
  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++

```cpp
#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++

```cpp
#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

```c
#include <stdio.h>

int main(){
   int liczba;
   scanf("%d", &liczba);
   printf("%d", liczba);
   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++

```cpp
#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

```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>

int main(){
  int l1, l2, l3;
  scanf("%d", &l1);
  scanf("%d", &l2);
  scanf("%d", &l3);
  printf("%f",(double)(l1 + l2 + l3) / 3);
  return 0;
}
```

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);`.

Rozwiązanie w języku C++ jest analogiczne.

Zadanie 1.2.9

Listing 10.9. Rozwiązanie zad. 1.2.9 w języku C

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

int main(){
  double x;
  scanf("%lf", &x);
  printf("%f",sqrt(x));
  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 kompilacji za pomocą gcc programów używających innych operacji z biblioteki `math`. Problem taki nie występuje w przypadku g++, a więc poniższe rozwiązanie dla C++ kompiluje się poprawnie bez dodatkowego parametru:

Listing 10.10. Rozwiązanie zad. 1.2.9 w języku C

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

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

Zadanie 1.2.11 Rozwiązanie dla języka C:

Listing 10.11. Rozwiązanie zad. 1.2.8 w języku C

```
#include <stdio.h>

int main(){
  float f;
  scanf("%f", &f);
  printf("%.2f",f);
  return 0;
}
```

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 liczbe calkowita: ");
  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 liczbe calkowita: ");
  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

```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++

```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 "<<p;
    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 liczbe calkowita n: ");
  scanf("%d", &n);
  printf("Podaj liczbe calkowita 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 kompilacji 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 liczbe calkowita n: ";
  cin>>n;
  cout<<"Podaj liczbe calkowita 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 liczbe calkowita 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.

Rozwiązanie 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 liczbe calkowita 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 liczbe calkowita n :";
  cin>>n;
  cout<<"Podaj liczbe calkowita 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 liczbe calkowita n :";
  cin>>n;
  cout<<"Podaj liczbe calkowita m :";
  cin>>m;
  pom1=n;
  pom2=m;
  while(pom1*pom2!=0)
    if (pom1>pom2)
      pom1=pom1%pom2;
    else
      pom2=pom2%pom1;
  cout<<"NWD liczb "<<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 liczbe calkowita x :");
  scanf("%d", &x);
  for(i=1; i<=x; i++)
    if (i * i <= x)
      pierw=i;
  printf("Pierw. z %d to w przyblizeniu %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

```c
#include <stdio.h>

int main(){
  int x, pocz, kon, sr;
  printf("Podaj liczbe calkowita 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 przyblizeniu ",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 zagnieżdżonych pętli:

Listing 10.26. Rozwiązanie zadania 1.4.12 w języku C

```c
#include <stdio.h>

int main(){
  int n, m, pom1, pom2, suma=0, i;
  printf("Podaj liczbe calkowita 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 obliczen 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. Rozwiązanie zadania 1.4.13 w języku C++

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

int main(){
  int n, silnia=1,suma=1;
  cout<<"Podaj liczbe calkowita 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. Rozwiązania do zadań z rozdziału 2.2

Zadanie 2.2.1

Listing 10.28. Rozwiązanie 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 liczbe calkowita: ");
  scanf("%d", &n);
  printf("|%d| = %d\n", n, bezwzgledna(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 liczbe calkowita: ");
  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 liczbe calkowita n: ");
  scanf("%d",&n);
  printf("Wynik obliczen to: %d\n",suma(n));
  return 0;
}
```

Dzięki używaniu funkcji programy zyskują na czytelności. Ponadto, raz zaimplementowane 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 „kopiach" 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`.

Rozwiązanie 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=%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=%d\n",i,i,p,p,n);
  }
}
```

W tym wariancie 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. Rozwiązanie 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. Rozwiązanie 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 niespełnienia 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 2.2.29

Listing 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 2.2.30

Listing 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 2.2.36

Listing 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 3.2.7

Listing 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 kopiach. 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 kopiach 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. Rozwiązanie 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. Rozwiązanie 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 rozwiązaniu zadania 1.4.4.

Zadanie 4.2.3

Listing 10.56. Rozwiązanie 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`.

Rozwiązanie 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 rozwiązaniu w języku C można ręcznie zarezerwować pamięć dla tablicy `sito`:

Listing 10.59. Rozwiązanie 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. Rozwiązanie 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. Rozwiązanie 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 $\{0,\ldots,n\}$, 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. Rozwiązania 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. Rozwiązanie 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. Rozwiązanie 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. Rozwiązanie 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. Rozwiązanie 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 rozwiązaniu 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);
}
```

Rozwiązania 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){
  wscanf(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 `wscanf`, który ograniczy rozmiar wczytywanych napisów.

Rozwiązania 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 rozwiązaniu 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;
}
```

Powyższego 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. Rozwiązanie 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. Rozwiązanie 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("jezeli 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.

Rozwiązanie w języku C++ wygląda analogicznie.

Zadanie 7.2.13

Listing 10.146. Rozwiązanie 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. Rozwiązanie 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 rozwiązaniu, 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. Rozwiązanie 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 imie "<<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;
      }
    }
  }
```

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 7.3.4

Listing 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 7.3.5

Listing 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 przeciwieństwie 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 element 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. 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;
}
```

Powyższe 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 * kopiuj(struct element* Lista){
  struct element * kopia,*pom;
  if (Lista==NULL)
    return NULL;
  kopia=mall0c(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. Rozwiązanie 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 rozwiązaniu 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. Rozwiązanie zadania 8.2.2 w języku C

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

Rozwiązanie w wersji dla języka C++:

Listing 10.197. Rozwiązanie 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 Rozwiązanie w wersji dla języka C i znaków typu `char`:

Listing 10.198. Rozwiązanie 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);
}
```

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

Listing 10.199. Rozwiązanie 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);
}
```

Rozwiązanie 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;
  }

}
```

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

Listing 10.205. Rozwiązanie 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;
}
```

Rozwiązanie dla znaków typu `wchar_t` jest analogiczne.

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

Listing 10.206. Rozwiązanie 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;
  }
}
```

Rozwiązanie 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 teiści 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 `(a`b+c)+, a nie `a`b+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 najwieksza(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 najwieksza(a,b,c) ((a>=b)&&(a>=c))?a:((b>=c)?b:c)
```

Zadanie 9.2.8

Listing 10.226. Rozwiązanie 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. Rozwiązanie zadania 9.2.10 w języku C/C++

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

10.13. Rozwiązania 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 wspolczynniki rownania 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 rozwiazan");
  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 glowny.c. Plik pierwiastek.c zostanie w całości wklejony do pliku glowny.c za sprawą dyrektywy `include`.

Rozwiązanie dla języka C++ jest analogiczne

Zadanie 9.3.3 Plik głównego programu:

Listing 10.230. Plik glowny.c

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

int main(){
  double a,b,c,delta;
  printf("Podaj wspolczynniki rownania 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 rozwiazan");
  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 kompilacji bibliotek.

Aby skompilować powyższy program, należy najpierw oddzielnie skompilować piliki `glowny.c` i `pierwiastek.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

pierwiastek.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 kompilacji. Dzięki temu program make, śledzi zmiany także w pliku nagłówkowym.

W pliku `Makefile` nie trzeba wymieniać wszystkich komend potrzebnych do kompilacji 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 czesc rzeczywista wczytywanej liczby");
  scanf("%lf",&(z.r));
  printf("Podaj czesc urojona wczytywanej liczby");
  scanf("%lf",&(z.u));
  return z;
  }

void wypisz(zespolone z){
  printf("Czesc rzecz. ma wartosc %f a czesc urojona %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.c

```
#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`.

Rozwiązanie 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 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(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.

Rozwiązanie 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 staystyka.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\n nazwisko: %s\n wiek %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 kompilacji 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