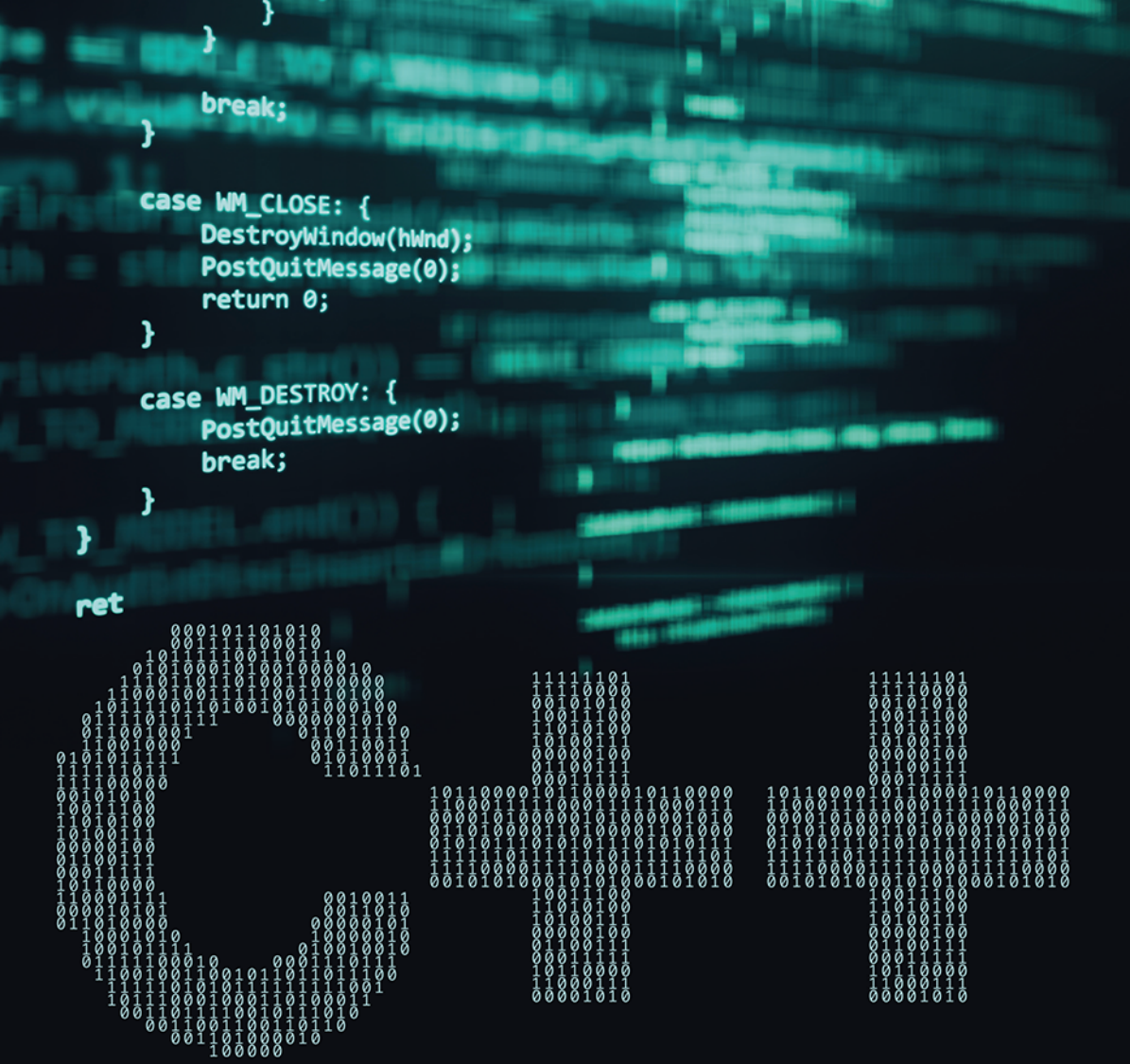

Tomasz Jaśniewski

ZBIÓR ZADAŃ Z ROZWIĄZANIAMI

Helion

Redaktor prowadzący: Szymon Szwajger
Projekt okładki: Studio Gravite / Olsztyn Obarek,
Pokoński, Pazdrijowski, Zaprucki

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

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

Materiały do książki znaleźć można pod adresem:
https://ftp.helion.pl/przyklady/cppzbz.zip

ISBN: 978-83-8322-202-8

Printed in Poland.

Spis treści

O autorze… i programowaniu

Jestem programistą tworzącym oprogramowanie automatyzujące i organizujące działalność firm, jednak nie mniejszą przyjemność sprawia mi nauczanie w szkole. Próbuję łączyć obie pasje, których wspólnym mianownikiem jest język programowania. Postrzegam w nim coś więcej niż narzędzie dające pracę. Programowanie to czysty akt kreacji, przygoda i źródło satysfakcji. To sztuka, choć wyrażona liniami kodu, a nie pędzlem, bardziej logiką niż uczuciami. To strategia działania realizowana w kolejnych iteracjach projektu aż do osiągnięcia celu. To sztuka myślenia.

Tak przy okazji — właśnie dlatego, że jest to sztuka myślenia, programowanie powinno być jednym z podstawowych narzędzi kształtowania umysłu dzieciaków w szkole. Nie trzeba od razu zostawać programistą, by skorzystać na sztuce myślenia. O ileż mniej porażek w szkole przeżywałyby nasze dzieci i młodzież, gdyby sprawniej wyciągały wnioski, szybciej dostrzegały wzorce, łatwiej projektowały kolejne kroki swoich działań dla osiągnięcia celu, aż wreszcie precyzyjniej wyrażały swoje myśli. O korzyściach i fali ukojenia dla nauczycieli płynących z tego faktu nie wspomnę.

Moim zdaniem programowanie nie jest czymś odrębnym od języka naturalnego, potrzebnym tylko wąskiej grupie zawodowców, ale — pomimo formalizmu języków programowania — jest jego fundamentalną cegiełką, wewnętrzną ukrytą składową, bez której każde zdanie i każda treść byłyby tylko nieuporządkowanym zbiorem nieporozumień. Tym, co czynimy, w sporym stopniu są funkcje i algorytmy. A to, co mówimy, w dużym stopniu jest opisem tego, co czynimy. Programowanie pozwala dobrze projektować i lepiej zrozumieć mechanizmy funkcjonowania rzeczy i zjawisk, dlatego nie może nie mieć pozytywnego wpływu na kształtowanie się sposobu naszego myślenia, gdyż język naturalny, język myśli, zawiera w sobie logikę, opis, celowość i ładunek informacji.

Powód powstania zbioru

Pracując w szkole i ucząc języków programowania, często stawałem przed problemem ugruntowania i szlifowania zdobytych umiejętności, sprawdzania w praktyce kolejnych porcji wiedzy, by móc je mierzyć i oceniać. Zanim zaczniemy programować poważniejsze aplikacje, po drodze tworzymy mnóstwo kodu, czego cel jest tylko jeden: sprawdzić siebie, poznać podstawy, zrozumieć elementarne klocki tworzące komputerową rzeczywistość. Uważam, że to bardzo ważny czas, którego nie można uniknąć i który powinien być dobrze wykorzystany. Wielu uczniów, wyczuwając instynktownie, jak wielkie możliwości stoją przed nimi, a równocześnie nie mając jeszcze wyraźnego obrazu tych możliwości,

a także nie posiadając żadnego doświadczenia, stawia pytania: „Co mam teraz robić?", „Jaki program napisać?", „Jak wykorzystać to, co było na lekcjach?". Jednym słowem: „Co zrobić z tą małą porcją wiedzy, którą posiadam? Jak ją zwiększyć?". W odpowiedzi na to pytanie powstawał ten zbiór zadań. Jest to próba ukierunkowania, uporządkowania i wzbogacenia tego czasu, który spędzamy pomiędzy pierwszą lekcją z *Hello World!* a pierwszą stworzoną samodzielnie przez nas aplikacją, którą sprzedaliśmy. Czasu radosnej nauki, kiedy wszystko wydaje się możliwe, a ogranicza nas tylko wyobraźnia.

Innym powodem powstania zbioru jest próba przeciwstawienia się negatywnym trendom obecnym wśród młodzieży szkolnej, które w mojej ocenie wydają się nasilać. Są to: spadające zdolności kreatywnego i abstrakcyjnego myślenia oraz problemy ze skupieniem uwagi. Dodatkowo wydłuża się czas poświęcony na zabawę kosztem nauki, co jeszcze bardziej demotywuje do podjęcia wysiłku związanego z jakąkolwiek pożyteczną pracą. Zdaję sobie sprawę, że jest to nieznacząca próba przeciwstawienia się tym trendom. Wydaje mi się jednak, że lepsze coś niż nic. A „lepsze" nie przyjdzie samo, trzeba je wypracować.

Podziękowania

Praca nad zbiorem to mieszanina frajdy i zmęczenia i kiedy do głosu dochodzi to drugie, przeoczenie błędów jest nieuniknione. Dziękuję **Grzegorzowi Gilowi** za udzielenie mi pomocy w wyłapywaniu błędów i za podrzucenie kilku alternatywnych rozwiązań zadań.

Charakterystyka opracowanego zbioru zadań

- Zbiór zadań jest podzielony na części zależne od posiadanych umiejętności, a te podzielone są tak, aby odpowiadały procesowi nauki języka. Wzrastający poziom wymaganych umiejętności w kolejnych zadaniach jest dostosowany do naturalnego wzrostu poziomu znajomości języka C++. Proponowane rozwiązania do każdego zadania uwzględniają wymagane umiejętności w poszczególnych częściach zbioru. Przykładowo: początkująca osoba nie potrafi tworzyć funkcji i nie zna wszystkich kontenerów, ale posiada już wiedzę o pętlach czy instrukcji warunkowej, zatem w proponowanym rozwiązaniu zadania wykorzystuje się tylko to, co wymienione jest w wymaganiach do poszczególnych części zbioru.
- Zbiór w pierwszej kolejności pomaga w przygotowaniach do matury z informatyki i podczas nauki programowania na studiach, uwzględniając wymagania systemu oświaty. Jednak wraz ze wzrostem wymaganych umiejętności zbiór przeprowadza przez wiele mechanizmów języka C++, które w dużym stopniu przekraczają wymagania szkolne.
- Zbiór uwzględnia standard C++20.

- Zadania zostały przetestowane na żywych organizmach uczniów klas informatycznych, z różnym skutkiem ☺.
- Do każdego zadania istnieje proponowane rozwiązanie. Niekiedy jest ich kilka — czasami eleganckie i krótkie w zapisie rozwiązanie zadania jest obarczone koniecznością dużego wykorzystania zasobów komputera, a niekiedy większa ilość kodu podnosi jego czytelność i przyśpiesza działanie.
- W zbiorze zadań założono, że użytkownik ma wiedzę o języku C++ i dlatego nie postawiono sobie w nim za cel tłumaczenia jego elementów. Jednocześnie proponowane zagadnienia pełne są komentarzy, które odkrywają tajemnice języka poprzez przykłady. Niniejsze opracowanie świetnie uzupełnia i domyka proces nauki C++, jednak same zadania nie wystarczą do poznania tego języka.

To musisz przeczytać!

Ten fragment zbioru zawiera bardzo ważne informacje dotyczące zadań. Przeczytaj je, inaczej możesz nie rozumieć oznaczeń stosowanych w zbiorze, a treść zadań może rodzić niepotrzebne pytania.

- Jeżeli zadanie jasno czegoś nie określa, to oznacza, że masz dowolność interpretacji treści zgodnie z logiką jego rozwiązania. Jeżeli masz lepszy pomysł na zadanie, wykonaj je raz jeszcze! Jeżeli zadanie zrodziło w Twojej głowie pomysł na jakiś wspaniały program, napisz go! Takie podejście pozwala wycisnąć maksimum z zadania i zwiększyć Twoje umiejętności. Nasz mózg zgłębia narzędzie, którego używa, z czasem zwiększając możliwości tegoż narzędzia poprzez kreatywne i twórcze pomysły. Gdy nagle dostrzegasz lepsze rozwiązanie, jest to coś naturalnego i nie należy tego pomijać, ale trzeba iść za ciosem i zrobić to! Tak rodzą się najlepsi.
- Liczba w nawiasie kwadratowym w nagłówku zadania (np. [6]) to sugerowana liczba punktów za zadanie, a zarazem przybliżona informacja o trudności zadania w danej grupie umiejętności. Miarą trudności jest moje widzi-mi-się, zatem jest to miara subiektywna. Wagi punktów w różnych grupach są różne. Punktacja zadania (np. [1]) w grupie początkowej ma mniejszą wagę i mniejszy stopień trudności od takiej samej punktacji w grupie późniejszej, która wymaga większej porcji umiejętności. Niekiedy punktacja jednego zadania jest rozbita i w treści występują takie oznaczenia jak np. [3,], a następnie np. [2], co wskazuje na to, że w zadaniu są etapy punktowane oddzielnie. Punktacja może wesprzeć nauczyciela przy planowaniu np. sprawdzianu, proszę ją jednak traktować orientacyjnie. Zatem [3,] i [2] to razem [5] punktów możliwych za poszczególne fragmenty zadania.
- We wszystkich zadaniach zakładamy poprawność danych (podobnie jest na maturze!). Programy z rozwiązaniami zazwyczaj nie zawierają obsługi błędów. Jeżeli dane początkowe mają być np. liczbami dodatnimi, to takimi dokładnie mają być. Czasami jednak w rozwiązaniu wykonuję

kilka podstawowych testów, np. na nieudany odczyt danych z pliku albo na to, czy liczba przy dzieleniu nie jest zerem, lub czy napis, który nie może być pusty, faktycznie taki nie jest, itd.

- Czasami rozwiązania do zadań będą zapisane na kilka sposobów. Niektóre są wydajniejsze lub bardziej skondensowane w zapisie, ale przez to mniej czytelne. Inne przeciwnie.
- Rozwiązania są niekiedy opatrywane komentarzami, które przybliżają fragmenty kodu. Jednak nie wszędzie komentarz jest obecny. Czytanie kodu ze zrozumieniem jest częścią procesu nauki języka i zakładam wzrost tej umiejętności wraz z kolejnymi zadaniami. Jeżeli mogę zasugerować coś nauczycielom, to jest to odpytywanie z czytania kodu ze zrozumieniem, polegające na dokładnym wyjaśnieniu zapisu i próbie określenia, co będzie wynikiem działania programu — to dobra praktyka. Weryfikuje, czy ktoś z językiem rzeczywiście pracuje, czy zapytany tworzy w przerażeniu po prostu nową historię…
- Niekiedy rozwiązanie zadania nie daje konkretnego wyniku, ale ma zbudować pewien algorytm/funkcjonalność. Dlatego w celach kontrolnych podawane będą jakieś dane przykładowe, mimo że zadanie bazuje np. na danych generowanych losowo. Ma to na celu weryfikację poprawności algorytmu/programu dla wspólnych danych.
- Czasem podaję pewną sugestię, propozycję napisania we własnym zakresie jakiegoś programu, większego projektu. Nie jest to jednak zadanie.
- Niektóre zadania naprawdę nie stanowią wyzwania, są raczej okazją do lepszego poznania gramatyki języka. Rozwiązania takich zadań mają bardziej charakter informacyjny, zapoznawczy, wprowadzający w jakąś przydatną technikę, którą zauważysz, analizując proponowane rozwiązanie.
- Niektóre zadania są podobne dla różnych części uwzględniających stopień znajomości języka. Jest tak dlatego, że pewne rozwiązania warto zaprojektować z uwzględnieniem postępu w rozwoju języka. Pozwala to zobaczyć, jak bogaty język ułatwia programowanie i zwiększa możliwości potencjalnego rozwoju programu. Rozwiązania zadań w poszczególnych częściach zbioru zakładają tylko posiadanie niektórych umiejętności programowania w języku C++. Oznacza to, że pewne zadania w oczywisty sposób da się zrobić szybciej/sprytniej, ale będzie to możliwe w przyszłości, dopiero po poznaniu dalszych własności C++. Takie podejście wiąże się z prostym faktem, że uczymy się od podstaw, a chcemy rozwiązywać zadania, posiadając nawet pewne minimalne umiejętności. Dlatego z czasem warto wrócić do początkowych zadań i rozwiązać je, stosując metody i umiejętności nabyte w późniejszym etapie kształcenia.
- Powinno się porównać własne rozwiązanie z proponowanym. Być może nauczysz się czegoś nowego? Jeżeli Twoje rozwiązanie jest lepsze i bardziej eleganckie, to masz darmową satysfakcję. A jeżeli nie, to masz darmową porcję wiedzy i umiejętności.

- W zadaniach oraz proponowanych ich rozwiązaniach nie zawsze wykorzystywane są wszystkie wymienione wymagane umiejętności w danej części zbioru. Lista umiejętności jest raczej pewnym zakresem, polem, w obrębie którego się poruszamy.
- Druga część zbioru zadań, numerowana #2, zawiera także umiejętności z pierwszej części.
- Zadania rozwiązywałem, korzystając przeważnie z Visual Studio 2022, zdarzało mi się jednak korzystać także z CodeBlocks. W niektórych sytuacjach może istnieć problem z uzyskaniem polskiej czcionki. Wiąże się to zarówno z kodowaniem czcionki w pliku, jak i z ustawieniami kodowymi w terminalu, do tego środowisko IDE i pewne ustawienia lokalne mogą wpływać na to, jak są wyświetlane polskie znaki (i czy w ogóle są). Zwróćcie na to uwagę, gdyż może się okazać, że to, co u mnie wyświetlało się z uwzględnieniem polskiej czcionki, u Was może jej nie uwzględniać, a na ekranie terminala w miejsce naszych „ś" czy „Ż" pojawią się niedrukowalne, dziwne znaki/krzaki. Musicie sobie z tym poradzić samodzielnie…

Dane do zadań

Poniższy link umożliwi Ci pobieranie plików z danymi do zadań:

https://ftp.helion.pl/przyklady/cppzbz.zip

W udostępnionym katalogu znajdziesz podkatalogi oznakowane *#1* albo *#2*, co odpowiada kolejnym częściom zbioru. Pliki zwykle zawierają w nazwie numer zadania, np. plik *2_dane.txt* odwołuje się do zadania nr 2. Jeżeli plików do zadania będzie więcej, nazwy wszystkich plików zostaną wymienione w treści zadania.

CZĘŚĆ 1. (#1) Ślady życia (40 zadań)

Wymagane umiejętności, znajomość zagadnień:

- sprawna obsługa środowiska programistycznego, np. Visual Studio 2019/2022 albo CodeBlocks;
- podstawowy schemat pliku *.cpp* ze źródłem w języku C++;
- podstawowe typy liczbowe całkowite i „z przecinkiem”:
  - `int`, `long long int`, `short int` wraz z modyfikatorem typu `unsigned`;
  - `float`, `double`;
- różne operacje na liczbach (+, –, /, *, `%` itd.);
- tworzenie zmiennych różnego typu, przypisywanie wartości do zmiennych, modyfikowanie wartości zmiennych (operator `=`), rozróżnianie zmiennej globalnej (poza `main()`) oraz lokalnej (np. w pętli `for`);
- typ `bool`, operacje logiczne (`and`, `or`, `||`, `&&`, `!`) , operatory porównania (>, <, `!=`, `<=`, `>=`), pojęcie warunku, znajomość wartości `true`/`false`, wiedza o interpretacji prawdy i fałszu przez komputer (np. liczba 0 to `false`, a nie 0 to `true` itp.);
- instrukcja warunkowa `if-else`, wyrażenie warunkowe `( ) ? _ : _ ;`;
- pętle `for`, `while`, `do..while`, wykorzystanie typu `auto` w pętli `for`, świadomość zasięgu i czasu życia zmiennych inicjowanych w pętli `for`;
- typ `char` (znajomość faktu, że zmienne typu `char` mogą być interpretowane jako liczby);
- kontener `vector<>` i podstawowe operacje z nim związane (sprawdzanie rozmiaru, wstawianie elementów, usuwanie elementów, używanie iteratorów `.begin()`, `.end()`);
  - typ `size_t` i jego związek np. z liczbą elementów w `vector<>`;
- kontener `array<>` (z przyczyn praktycznych będę wyłącznie korzystać z `vector<>` zamiast `array<>`);
- starsze tablice [] (w rozwiązaniach je pomijam, będę korzystać z `vector<>`);
- instrukcje wejścia/wyjścia (na konsolę) `cout`, `cin` i operatory strumienia << oraz >>;
- operatory ++, --, +=, *=, -=, /=, `%=`;
- rozumienie i znajomość kolejności działań i priorytetów wyżej wymienionych operatorów;
- funkcje z biblioteki `<cmath>`, takie jak: `pow()`, `abs()`, `sqrt()`;
- korzystanie z `typedef` np. w celu utworzenia skrótu dla długich typów.

Czasami używam zwrotu „pobierz z klawiatury”. Oznacza on wprowadzenie wartości do zmiennych za pomocą klawiatury, w trakcie działania programu (`cin >>`).

Zwroty „uwzględnij małe litery angielskie”, „uwzględnij tylko cyfry” i inne podobne zwroty oznaczają, że wprowadzane dane mogą być tylko takie i nie zakładamy wykorzystania innych.

Zadania

Zadanie 1.1 [2]

Pobierz z klawiatury trzy nieujemne liczby całkowite. Znajdź największą z nich. Wyświetl sumę pozostałych liczb tyle razy, ile wynosi wartość największej liczby. [2]

Zadanie 1.2 [1]

Pobraną z klawiatury liczbę całkowitą zweryfikuj pod kątem parzystości. Wyświetl `tak` lub `nie`, gdy jest lub nie jest parzysta. [1]

Zadanie 1.3 [1]

Pobierz liczbę całkowitą z klawiatury i sprawdź, czy jest podzielna: przez 3 i przez 5; przez 3, ale nie przez 5; przez 5, ale nie przez 3; ani przez 3, ani przez 5. Właściwą odpowiedź wyświetl na ekranie. [1]

Zadanie 1.4 [1]

Pobierz znak (`char`) z klawiatury. Sprawdź, czy to samogłoska, spółgłoska, czy cyfra. Poinformuj o tym, jaki to znak. Uwzględnij tylko małe litery alfabetu angielskiego i cyfry. [1]

Zadanie 1.5 [1]

Masz takie wyrażenie: `(((a1+a2)*a3)-a4)/a5` (elementy od `a1` do `a5` są typu `float`). Pobierz z klawiatury każdą ze zmiennych `a1` do `a5`, oblicz wartość wyrażenia i wyświetl wynik. [1]

Zaprojektuj program tak, aby w przypadku dzielenia przez zero informował o tym i nie wykonywał działania.

Rozwiązania

Kilka słów wyjaśnienia

Możesz zauważyć, że niektóre zadania można wykonać lepiej, szybciej, stosując pełnię możliwości języka C++. Rozwiązania tu proponowane zakładają jednak posiadanie ograniczonej wiedzy i umiejętności z zakresu języka C++. Wynika to z naturalnego procesu uczenia się języka, w którym kolejne możliwości otwierają się przed nami dopiero po jakimś czasie. Pomimo mniejszego zakresu umiejętności rozwiązywanie zadań na tym etapie jest jak najbardziej możliwe. Tworząc rozwiązania zadań, próbowałem uwzględnić wykorzystanie tylko pewnej porcji umiejętności. Z czasem w rozwiązaniach zadań będą wykorzystywane rosnące umiejętności w zakresie posługiwania się językiem C++.

Poniżej znajdują się proponowane rozwiązania, jednak nie muszą one być wcale najlepsze. Porównaj je ze swoimi rozwiązaniami. Jeżeli moje są lepsze, to jest wydajniejsze, bardziej czytelne, istotnie krótsze itd., to spróbuj je zrozumieć. W ten sposób będziesz się uczyć i rozwijać.

Jeżeli posiadasz większe umiejętności niż zakładane w części pierwszej (#1) i drugiej (#2), śmiało z nich korzystaj. Pamiętaj jednak, że poniższe rozwiązania ograniczają się do technik i umiejętności wymienionych w obydwu częściach tego zbioru zadań.

Część 1.
(#1) Ślady życia

Zadanie #1.1

Przykładowe rozwiązanie 1.

```
#include <iostream>
using namespace std;
int main() {
  setlocale(LC_ALL, "");
  unsigned a, b, c; // czyli unsigned int
  unsigned suma = 0, maks;
  cout << "Podaj kolejne 3 liczby całkowite nieujemne:";
  cin >> a >> b >> c;
  if (a > b) {
      maks = a;
      suma += b;
  }
  else {
```

```
    maks = b;
    suma += a;
  }
  if (maks > c) {
    suma += c;
  }
  else {
    suma += maks;
    maks = c;
  }
  for (unsigned i = 1; i <= maks; i++) {
    cout << suma << " ";
  }
}
```

Przykładowe rozwiązanie 2.

```
#include <iostream>
using namespace std;
int main() {
  setlocale(LC_ALL, "");
  unsigned a, b, c; // to samo co unsigned int
  cout << "Podaj kolejne 3 liczby całkowite nieujemne:";
  cin >> a >> b >> c;
  unsigned suma = a + b + c;
  unsigned maks = (a > b) ? a : b;
  suma -= maks = (maks > c) ? maks : c;
  while (maks--) {
    cout << suma << " ";
  }
}
```

Zadanie #1.2

```
#include <iostream>
using namespace std;
int main() {
  setlocale(LC_ALL, "");
  int a;
  cout << "Podaj liczbę całkowitą:";
  cin >> a;
  if (a % 2 == 0) cout << "tak\n";
  else cout << "nie\n";
}
```

Zadanie #1.3

```
#include <iostream>
using namespace std;
int main() {
  setlocale(LC_ALL, "");
  int a;
  cout << "Podaj liczbę całkowitą:";
  cin >> a;
  if (a % 3 != 0 and a % 5 != 0) {
```

```
    cout << "niepodzielna ani przez 3, ani przez 5\n";
  }
  else if (a % 3 == 0 and a % 5 != 0) {
    cout << "podzielna przez 3, ale nie przez 5\n";
  }
  else if (a % 3 != 0 and a % 5 == 0) {
    cout << "podzielna przez 5, ale nie przez 3\n";
  }
  else {
    cout << "podzielna przez 3 i przez 5 (równocześnie)\n";
    // czy to oznacza, że podzielna przez 15? :)
  }
}
```

Zadanie #1.4

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  setlocale(LC_ALL, "");
  char znak;
  cin >> znak; // wpisz tylko małe litery angielskie lub cyfry
  vector<char> samogloski = { 'a','e','i','o','u','y' };
  // kod '0' to 48, '1' to 49 itd.; przy porównaniu char jest rzutowany na liczbę
  if (znak >= 48 and znak <= 48 + 9) {
    cout << "Znak to cyfra.\n";
    return 0; // zakończ program
  }
  bool samogloska = false;
  for (auto s : samogloski)
    if (znak == s) {
      samogloska = true;
      break;
    }
  if (samogloska) cout << "Znak to samogłoska.\n";
  else cout << "Znak to spółgłoska.\n";
}
```

Zadanie #1.5

```
#include <iostream>
using namespace std;
int main() {
  setlocale(LC_ALL, "");
  float a1, a2, a3, a4, a5;
  cin >> a1 >> a2 >> a3 >> a4 >> a5;
  if (a5 == 0.0) {
    cout << "Dzielenie przez 0 to nie najlepszy pomysł.\n";
    return 0; // opuść program
  }
  cout << "Wartość wyrażenia: " << (((a1 + a2) * a3) - a4) / a5 << endl;
  // uwaga: starsze kompilatory mogą mieć problem z rzeczywistym porównaniem a5 == 0.0,
  // gdyż liczba 0.0 może być przybliżeniem, a faktyczna
  // wartość może wynosić np. 0.00000000000082790 itp.
}
```

Hello world! — i co dalej w C++?

Autor tego zbioru zadań jest programistą i nauczycielem. To prawdziwy pasjonat programowania — w tym w języku C++ — które traktuje jak przedłużenie ludzkiej mowy. Uważa, że praktycznie na wszystko, co robimy w życiu, można spojrzeć jak na wykonywanie funkcji i algorytmów, które opisujemy za pomocą słów. Od razu widać związek między podejściem humanistycznym i informatycznym! Dlatego też nauka programowania w żadnym stopniu nie jest opanowywaniem jakiejś specjalnej wiedzy tylko dla wtajemniczonych. To po prostu utrwalanie tego, co już wiemy i umiemy, tyle że w sposób logiczny i uporządkowany.

Niestety, nauka języków programowania w szkole nader często ogranicza się do przekazania suchych informacji na ich temat. Wielu uczniów, instynktownie wyczuwając potencjał zdobytej właśnie wiedzy na temat C++, stawia pytania: Co mam teraz robić? Jaki program napisać? Jak wykorzystać to, co było na lekcjach? W odpowiedzi na nie powstał ten zbiór zadań. Stanowi on próbę ukierunkowania, uporządkowania i wzbogacenia czasu pomiędzy pierwszą lekcją z „Hello world!" a pierwszą sprzedaną aplikacją — stworzoną samodzielnie lub w zespole. Czasu radosnej nauki, kiedy wszystko wydaje się możliwe, a ogranicza nas tylko wyobraźnia.

Poświęcony C++ zbiór zadań z rozwiązaniami:

- **jest podzielony na części zależne od posiadanych umiejętności, a te zostały uporządkowane tak, by odpowiadać procesowi nauki języka C++**
- **pomaga zwłaszcza w przygotowaniu do matury z informatyki i w nauce programowania na studiach**
- **uwzględnia standard C++20**
- **zawiera propozycje rozwiązań**

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-8322-202-8

9 788383 222028

Cena: 37,00 zł