1.3. Od algorytmu do programu w języku C++

Teraz zajmiemy się zapisem algorytmów w języku programowania, czyli tworzeniem kodów źródłowych programów komputerowych. Aby komputer mógł wykonać napisany przez nas program, kod musi zostać przetłumaczony na język wewnętrzny procesora (kod maszynowy).

Zintegrowane środowisko programistyczne (IDE)

Zintegrowane środowiska programistyczne (IDE, od ang. *integrated development environment*) umożliwiają nie tylko pisanie kodu źródłowego programu, lecz także jego przetłumaczenie oraz wykonanie.

Translator

Programy tłumaczące nazywamy **translatorami**. Możemy je podzielić na dwie kategorie:

Kompilator

- **kompilatory** – tłumaczą cały kod źródłowy, po zakończeniu tłumaczenia powstaje kod wynikowy – żeby go wykonać, nie jest już potrzebny kompilator,

Interpreter

- **interpretery** – interpretują i wykonują polecenia kodu źródłowego instrukcja po instrukcji; dlatego za każdym razem, gdy chcemy sprawdzić, jak działa program, trzeba uruchamiać interpreter.

Przykładem języka kompilowanego jest język C++, natomiast interpretowanego – język Python. W tym podręczniku skupimy się na tworzeniu programów w języku C++.

Struktura programu w języku C++

Zanim napiszemy i uruchomimy pierwszy program, warto poznać kilka reguł obowiązujących podczas pracy w języku C++.

Słowo kluczowe języka programowania oznacza określoną deklarację lub instrukcję i nie można go przedefiniować. Słowa kluczowe są wyróżniane pogrubioną czcionką i/lub określonym kolorem.

- Wielkość liter ma znaczenie – **słowa kluczowe** języka piszemy małymi literami.
- Każdą instrukcję kończymy średnikiem.
- W instrukcjach nie używamy polskich znaków.
- Nawiasy klamrowe pełnią funkcję początku i końca bloku.
- Tekst po znakach // jest komentarzem. Kompilator pomija komentarze, dlatego można używać w nich polskich znaków.
- Na początku kodu umieszczamy dyrektywę `#include <iostream>` – powoduje ona dołączenie biblioteki z obsługą standardowego wejścia (czytanie danych, domyślnie z klawiatury) i wyjścia (wypisywanie wyników, domyślnie na ekranie).

Warto wiedzieć

Nazwa `iostream` pochodzi od angielskiego wyrażenia *input-output stream*, które możemy rozumieć jako strumień danych wejścia i wyjścia.

- Za pomocą instrukcji `using namespace std;` określamy użycie tzw. standardowej przestrzeni nazw, ułatwiającej odwołanie się m.in. do instrukcji czytania i pisania.
- Każdy program składa się z funkcji.

Funkcja `main`

- Główną część programu stanowi **funkcja** `main`:

```
int main()    // nagłówek funkcji main
{             // początek bloku instrukcji
              // tu będą instrukcje funkcji
}             // koniec bloku instrukcji
```


Programy tworzone w językach programowania mają ustaloną strukturę. **Podstawowa struktura kodu źródłowego programu w języku C++** wygląda następująco:

Podstawowa struktura kodu źródłowego programu w języku C++

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  int main()
6.  {
7.
8.  }
```

Dobra rada

Utworzenie pliku zawierającego podstawową strukturę programu przyspieszy pisanie nowych programów. Programy tworzone na jego podstawie zapisuj pod nowymi nazwami.

Obok kodu źródłowego programu znajdują się numery, które są oznaczeniami kolejnych linii kodu. Ułatwią one omawianie znaczenia poszczególnych instrukcji. Programy pisane w języku C++ będziemy zapisywać i uruchamiać w zintegrowanym środowisku programistycznym Code::Blocks.

Aby w środowisku Code::Blocks utworzyć nowy, pusty plik, należy wybrać **File → New → Empty file**. Przy zapisywaniu pliku (**File → Save file as...**) środowisko Code::Blocks domyślnie proponuje podanie nazwy z rozszerzeniem *c* (właściwym dla języka C), dlatego trzeba zmienić rozszerzenie na *cpp*, odpowiadające językowi C++.

Dobra rada

Aktualną wersję programu Code::Blocks pobierzesz ze strony codeblocks.org. Dostępne są na niej wersje programu działające pod różnymi systemami operacyjnymi. Pracującym w systemie Windows zalecamy pobranie wersji oprogramowania zawierającej kompilator języka C++ (GCC/G++).

Ćwiczenie 7

W środowisku Code::Blocks utwórz nowy, pusty plik, a następnie przepisz do niego podstawową strukturę programu podaną powyżej. Zapisz plik pod nazwą *szablon.cpp*.

Dobra rada

Jeśli nie wiesz, jak wymawiać obcojęzyczne nazwy umieszczone w podręczniku, skorzystaj z translatora, który daje możliwość odsłuchania tekstu, np. Tłumacza Google lub Bing Microsoft Translator.

Program sprawdzający, czy liczba jest parzysta

Program, który napiszemy, wczyta z klawiatury liczbę całkowitą, sprawdzi, czy jest parzysta, i wypisze na ekranie odpowiedni napis. Zakres danych ograniczymy ze względu na zakres reprezentowanych liczb całkowitych. Więcej o tym dowiesz się w następnym temacie. Specyfikacja problemu oraz zapis algorytmu w pseudokodzie są następujące:

Specyfikacja

Dane: n – liczba całkowita, $-1\,000\,000\,000 < n < 1\,000\,000\,000$.

Wynik: napis „Parzysta", gdy n jest liczbą parzystą, napis „Nieparzysta", gdy n jest liczbą nieparzystą.

```
jeśli n mod 2 = 0 to wypisz "Parzysta"
w przeciwnym przypadku wypisz "Nieparzysta"
```

Kod źródłowy programu realizującego algorytm zapisany w pseudokodzie wygląda następująco:

Kod źródłowy programu sprawdzającego, czy podana liczba jest parzysta

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  int main()
6.  {
7.      int n;
8.      cin>>n;
9.      if (n%2==0)
10.         cout<<"Parzysta";
11.     else
12.         cout<<"Nieparzysta";
13.     return 0;
14. }
```

Dobra rada

Program C++ ignoruje tzw. białe znaki, czyli np. spacje, tabulacje. Dzięki temu możesz dostosować czytelność kodu do swoich preferencji – ustawić takie wcięcia, jakie chcesz.

Kod bloku instrukcji programu (funkcji `main`) znajduje się pomiędzy nawiasami klamrowymi w liniach 7–13. W linii 7 została zadeklarowana zmienna o nazwie n. **Deklaracja zmiennej** w języku C++ składa się z nazwy zmiennej oraz określenia **typu danych**, jakie może ona przechowywać. Typ **`int`** oznacza liczby całkowite.

Deklaracja zmiennej

Typ danych

W linii 8 za pomocą **instrukcji wejścia `cin`** wczytujemy liczbę ze standardowego wejścia, czyli klawiatury. Operator `>>` wskazuje, dokąd zostanie przesłana dana. W tym przypadku będzie zapamiętana w zmiennej n.

Instrukcja wejścia cin

Linie 9–12 odpowiadają zapisowi algorytmu w pseudokodzie. **Instrukcja warunkowa `if else`** w języku C++ ma postać:

Instrukcja warunkowa if else

```
if (warunek) instrukcja na tak;
else instrukcja na nie;
```

Warunek logiczny musi być ujęty w nawiasy okrągłe. W linii 9 badamy, czy reszta z dzielenia wartości zmiennej n przez 2 jest równa zero, czyli czy n jest liczbą parzystą. Do **porównywania wartości** służy operator `==`. Operacja arytmetyczna reszty z dzielenia (**`mod`** w pseudokodzie) to znak `%`. **Instrukcja wyjścia `cout`** służy do wypisania wyniku na ekranie komputera. Operator `<<` wskazuje dane do przekazania na standardowe wyjście. W tym przypadku danymi są teksty „Parzysta" lub „Nieparzysta". Ciąg znaków (zwany też łańcuchem) w kodzie źródłowym musi znajdować się w tzw. amerykańskim cudzysłowie, czyli między znakami cala: " ". Ostatnia instrukcja programu (linia 13) kończy wykonywanie funkcji `main` (a w konsekwencji całego programu) oraz informuje system operacyjny o poprawnym zakończeniu działania (kod `0`).

Porównywanie wartości w języku C++

Instrukcja wyjścia cout

Warto wiedzieć

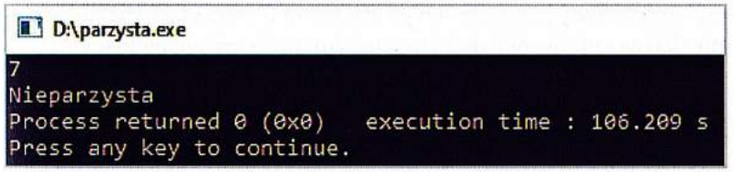
Polecenie **`cin`** pochodzi od angielskiego wyrażenia *console input*, a polecenie **`cout`** – od wyrażenia *console output*.

Po napisaniu programu w środowisku Code::Blocks należy go skompilować i uruchomić – służy do tego opcja **Build → Build and run**.

Jeśli kod programu zapisaliśmy zgodnie z regułami języka, otrzymamy nowy plik: w systemie Windows będzie to plik z rozszerzeniem *exe*. Zostanie on zapisany w tym samym katalogu co plik źródłowy. Jednocześnie Code::Blocks uruchomi program. Można sprawdzić działanie programu, podając liczbę całkowitą (rys. 1.2). Skompilowany program uruchamiamy ponownie, wybierając **Build → Run**.

Warto wiedzieć

W systemach Linux i macOS pliki wykonywalne nie są wyróżniane żadnym rozszerzeniem. Aby można było je uruchomić, muszą mieć ustawiony atrybut wykonywalności.



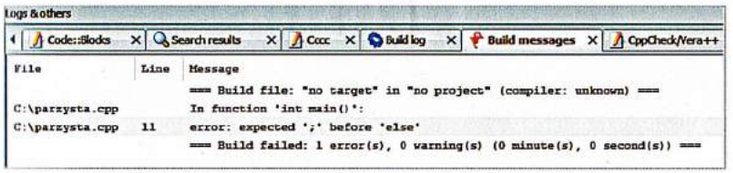
Rys. 1.2. Przykład wywołania programu *parzysta.exe*

Jeśli w kodzie programu znajdzie się **błąd składniowy**, czyli błąd wynikający z zapisania kodu niezgodnie z regułami języka, w oknie programu Code::Blocks wyświetli się odpowiedni komunikat. Na przykład jeśli zapomnimy wstawić średnik na końcu instrukcji w linii 10, otrzymamy informację jak na rysunku 1.3.

Błąd składniowy

Warto wiedzieć

Konsolę zawierającą informację o błędach składniowych włącza się i wyłącza klawiszem F2.



Rys. 1.3. Komunikat o braku średnika na końcu instrukcji w linii 10

Kod źródłowy programu może zawierać także **błędy logiczne**. Są to błędy wynikające np. z błędów w algorytmie. Kompilator nie wykryje błędu logicznego – program się skompiluje, jednak dla poprawnych danych nie zwróci poprawnego wyniku.

Błąd logiczny

Ćwiczenie 8

W środowisku Code::Blocks utwórz plik *parzysta.cpp*. Wpisz kod źródłowy programu sprawdzającego, czy dana liczba jest parzysta (zastosuj wcięcia), i zapisz wprowadzone zmiany. Następnie skompiluj i uruchom program. Jeśli wystąpią błędy kompilacji, popraw kod, a następnie ponownie skompiluj i uruchom program. Sprawdź działanie programu dla różnych danych, zarówno liczb parzystych, jak i nieparzystych.

Program dzielący liczbę na cyfry

Program, który teraz napiszemy, będzie implementował (realizował) algorytm omówiony wcześniej w tym temacie.

Specyfikacja

Dane: n – liczba całkowita, $0 < n < 1\,000\,000\,000$.

Wynik: kolejne cyfry liczby n, zaczynając od cyfry jedności (od prawej do lewej).

Kod źródłowy programu wypisującego cyfry wczytanej liczby w pionie (każda cyfra w oddzielnym wierszu) może być następujący:

Kod źródłowy programu dzielącego liczbę na cyfry

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  int main()
6.  {
7.      int n;
8.      cin>>n;
9.      while (n>0)
10.     {
11.         cout<<n%10<<endl;
12.         n=n/10;
13.     }
14.     return 0;
15. }
```

W liniach kodu 9–13 przeglądane są kolejne cyfry wczytanej liczby. Realizuje to **instrukcja iteracji (pętla) `while`**. Jej składnia w języku C++ jest następująca:

Instrukcja iteracji (pętla) `while`

```
while (warunek) instrukcja;
```

Dopóki `warunek` przyjmuje wartość logiczną **`prawda`**, instrukcja w pętli jest powtarzana. Zwróć uwagę, że jeżeli `warunek` pętli na początku przyjmie wartość logiczną **`fałsz`**, to instrukcja w pętli nie wykona się ani razu. W powyższym programie tak się stanie, jeśli wczytamy liczbę ujemną lub zero. Wykonywana w pętli instrukcja musi mieć też wpływ na warunek logiczny pętli, czyli zmieniać wartości, od których ten warunek zależy. W przeciwnym przypadku warunek będzie miał cały czas wartość logiczną **`prawda`** i pętla będzie się wykonywała w nieskończoność. W takich sytuacjach mówimy, że „program się pętli".

Warto wiedzieć

W języku C++ znak / oznacza zarówno dzielenie całkowite (część całkowitą z dzielenia), jak i dzielenie z częścią ułamkową. Interpretacja znaczenia zależy od typu argumentów.

W linii 12 zmienna `n` zmniejsza swoją wartość (jej dotychczasowa wartość jest dzielona całkowicie przez 10). Znak `=` służy do oznaczania przypisania wartości. **Instrukcja przypisania wartości zmiennej w języku C++** ma postać:

Instrukcja przypisania wartości zmiennej w języku C++

```
zmienna = wartość wyrażenia;
```


W linii 11 wypisywana jest reszta z dzielenia wartości zmiennej `n` przez 10 oraz następuje przejście do następnego wiersza (wstawienie znacznika końca wiersza po drugim operatorze `<<`, słowo specjalne **`endl`**). Zwróć uwagę na linie 10 i 13. Są tam nawiasy klamrowe wyznaczające blok instrukcji powtarzanych w pętli – są one konieczne, jeśli w pętli (lub innej instrukcji, np. warunkowej **`if else`**) chcemy wykonać więcej niż jedną instrukcję. Blok instrukcji jest traktowany jak pojedyncza instrukcja.

Warto wiedzieć

W nazwach zmiennych oprócz małych i wielkich liter oraz cyfr (te nie mogą być na początku nazwy) można używać również znaku podkreślenia _.

Warto wiedzieć

Zapis **`endl`** pochodzi od angielskiego wyrażenia *end line*, które oznacza koniec linii (wiersza) tekstu.

Ćwiczenie 9

Napisz program dzielący liczbę na cyfry i przetestuj jego działanie.

Programy realizujące algorytmy podziału na równoliczne grupy

Przedstawiliśmy wcześniej dwa algorytmy rozwiązania tego problemu. Napiszemy programy realizujące obydwa algorytmy.

Specyfikacja

Dane: n – liczba całkowita, $0 < n < 1\,000\,000\,000$.

Wynik: liczba dzielników właściwych n większych od 1.

W pierwszym z algorytmów badaliśmy podzielność `n` przez kolejne liczby od 2 do połowy `n`. Kod programu może wyglądać następująco:

Kod źródłowy programu liczącego dzielniki właściwe danej liczby większe od 1 – algorytm 1

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  int main()
6.  {
7.      int n, d, ld=0;
8.      cout<<"n="; cin>>n;
9.      for (d=2;d<=n/2;d++)
10.         if (n%d==0) ld++;
11.     cout<<"Liczba dzielnikow wlasciwych wiekszych od 1: ";
12.     cout<<ld<<endl;
13.     return 0;
14. }
```

W linii 7 oprócz `n` zadeklarowane są jeszcze dwie zmienne typu całkowitego: `d` – potencjalny dzielnik i `ld` – licznik dzielników, któremu nadana jest od razu wartość początkowa 0. Wczytanie danej `n` zostało poprzedzone wypisaniem na ekranie napisu informującego, o jakie dane prosimy, tzw. napisu zachęty (linia 8).

Instrukcja iteracji (pętla) for

W liniach 9 i 10 zastosowaliśmy **instrukcję iteracji (pętlę) for**. Stosuje się ją zwykle w sytuacjach, kiedy z góry znamy liczbę powtórzeń. Jej składnia jest następująca:

```
for (instrukcja inicjująca; warunek powtarzania; instrukcja modyfikująca) instrukcja;
```

Wartością początkową zmiennej d jest 2. Wykonywanie instrukcji pętli odbywa się, dopóki wartość zmiennej d jest mniejsza lub równa n/2. Po każdym wykonaniu instrukcji pętli wartość zmiennej d jest powiększana o 1 (d++). W pętli (linia 10) licznik dzielników ld jest powiększany o 1, gdy aktualna wartość d jest dzielnikiem n (reszta z dzielenia jest równa 0).

Wypisanie na ekranie wyniku (linia 12), czyli wartości licznika dzielników, poprzedzone jest wypisaniem komentarza (linia 11).

Poniżej znajduje się kod źródłowy drugiego algorytmu znajdującego liczbę dzielników właściwych danej liczby większych od 1. Zwróć uwagę na modyfikację warunku pętli **for** (linia 9).

Warto wiedzieć

Pętlę **for** można zapisać za pomocą pętli **while**:

```
instrukcja inicjująca;
while (warunek
  powtarzania)
{
  instrukcja;
  instrukcja
  modyfikująca;
}
```

Warto wiedzieć

Instrukcja d++ jest równoznaczna z instrukcją d=d+1.

Kod źródłowy programu liczącego dzielniki właściwe danej liczby większe od 1 – algorytm 2

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  int main()
6.  {
7.      int n, d, ld=0;
8.      cout<<"n="; cin>>n;
9.      for (d=2;d*d<n;d++)
10.         if (n%d==0) ld=ld+2;
11.     if (d*d==n) ld++;
12.     cout<<"Liczba dzielnikow wlasciwych wiekszych od 1: ";
13.     cout<<ld<<endl;
14.     return 0;
15. }
```

Ćwiczenie 10

Zapisz, uruchom i przetestuj obie przedstawione w temacie wersje programów realizujących algorytmy podziału na równoliczne grupy.

Zapamiętaj

Napisanie programu komputerowego stanowi tylko jeden z etapów rozwiązywania problemu algorytmicznego. Najpierw należy precyzyjnie sformułować problem i podać jego specyfikację, następnie poszukać rozwiązania – algorytmu. Dopiero potem można rozpocząć pisanie programu, pamiętając o testowaniu i ewentualnych poprawkach.

Podsumowanie

- Wyróżniamy następujące etapy rozwiązywania problemów z wykorzystaniem komputera: określenie problemu (zadania), podanie specyfikacji, sformułowanie rozwiązania – algorytmu, zaprogramowanie rozwiązania, testowanie rozwiązania.
- Specyfikacja zadania to określenie danych i wyniku oraz związku między nimi.
- Algorytm to skończony ciąg precyzyjnie określonych instrukcji prowadzący do rozwiązania problemu.
- Dany problem można przeważnie rozwiązać na wiele sposobów, czyli podać różne algorytmy rozwiązania.
- Zapis algorytmu w języku programowania to kod źródłowy programu.
- Testowanie programu polega na wielokrotnym uruchamianiu go dla różnych danych (spełniających warunki specyfikacji).
- Różne sposoby zapisu algorytmów to: lista kroków, schemat blokowy, pseudokod, zapis w języku programowania.
- Podstawowe instrukcje sterujące w językach programowania to instrukcja warunkowa (wykonywanie instrukcji pod pewnym warunkiem) oraz instrukcja iteracji, czyli pętli (wielokrotne powtarzanie instrukcji).
- Translatory (które dzielimy na kompilatory i interpretery) to programy tłumaczące kod źródłowy programu na język maszynowy.
- Zintegrowane środowisko programistyczne (IDE) to program posiadający edytor do pisania kodu źródłowego, translator oraz wiele funkcji ułatwiających uruchamianie programów.

Zadania

* **1** Wiele czynności życia codziennego możemy opisać algorytmicznie. Podaj kilka przykładów. Wybrany z nich zapisz w postaci listy kroków.

* **2** Znajdź w sieci film pokazujący wiązanie dowolnego węzła żeglarskiego. Zapisz algorytm wiązania tego węzła w postaci listy kroków.

* **3** Przedstaw w postaci listy kroków algorytm:
- **a.** pisemnego dodawania dwóch liczb naturalnych,
- **b.** pisemnego mnożenia liczby jednocyfrowej przez wielocyfrową.

* **4** Zapisz w wybranej przez siebie notacji (lista kroków, schemat blokowy, pseudokod) algorytm sumujący cyfry danej liczby całkowitej dodatniej.

* **5** Napisz, uruchom i przetestuj program sumujący cyfry wczytanej liczby całkowitej dodatniej. Najpierw zapisz specyfikację.

** **6** Napisz program sprawdzający, czy w zapisie dziesiętnym wczytanej liczby całkowitej dodatniej pierwsza cyfra od lewej jest równa cyfrze jedności. Program powinien wypisać słowo „TAK" lub „NIE".