Temat 5. Iteracyjna realizacja wybranych algorytmów

Warto powtórzyć

1. W jaki sposób określaliśmy w dotychczas wykonywanych zadaniach warunki zakończenia iteracji?
2. W jaki sposób definiowaliśmy warunki?
3. Kiedy warunek jest spełniony w języku Pascal, a kiedy w języku C++?
4. W jaki sposób definiujemy procedury i funkcje?
5. W jaki sposób wywołujemy procedury i funkcje?
6. Przypomnij algorytm wyboru największej z trzech liczb i sposób zapisu tego algorytmu w postaci programu komputerowego (tematy C4 i C5, *Informatyka podstawowa*).

1. Przykłady algorytmów, w których liczba kroków iteracji nie jest z góry określona

W programach tworzonych w językach Pascal i C++ najczęściej wprowadzaliśmy liczbę iteracji z klawiatury po uruchomieniu programu – stosowaliśmy instrukcję iteracyjną (instrukcję pętli) `for`.

W niektórych algorytmach liczba kroków iteracji może zależeć od spełnienia (bądź niespełnienia) warunku logicznego, określonego w zadaniu. W takich sytuacjach możemy zastosować dwie pozostałe instrukcje iteracyjne:

- **`while..do`** (w języku Pascal) lub **`while`** (w języku C++),
- **`repeat..until`** (w języku Pascal) lub **`do..while`** (w języku C++).

1.1. Zastosowanie instrukcji iteracyjnej `while`

Instrukcja iteracyjna **while..do**	**while** *warunek* **do** *instrukcja;*	**Pascal**
Instrukcja iteracyjna **while**	**while** (*warunek*) *instrukcja;*	**C++**

Działanie pętli **while** jest takie samo w obydwu językach: najpierw sprawdzany jest *warunek*; jeśli jest on spełniony, to wykonywana jest *instrukcja*.

Wewnątrz bloku *instrukcji* powinna być zawsze umieszczona instrukcja, która zmienia wartość *warunku* – w przeciwnym wypadku pętla nigdy się nie zakończy. W szczególnej sytuacji, gdy *warunek* od razu jest niespełniony, *instrukcja* może w ogóle nie zostać wykonana. *Instrukcja* może być pojedyncza lub złożona.

Uwaga:
W treści tematu będziemy używali skróconego zapisu instrukcji.
W języku Pascal:
for zamiast **for..to..do**,
while zamiast **while..do**,
repeat zamiast **repeat..until**,
a w języku C++:
do zamiast **do..while**.

Przykład 1. Stosowanie pętli `while` w językach Pascal i C++

Napiszemy program, który będzie wczytywał wprowadzone z klawiatury kolejne pary liczb rzeczywistych (*a* i *b*) i obliczał ich iloraz, dopóki druga liczba nie będzie równa zero. Dla *b* równego zero program powinien wyprowadzić komunikat o treści „dzielenie przez zero jest niewykonalne" i zakończyć działanie.

W zadaniu tym nie określono, dla ilu par liczb program ma być wykonywany, ale wprowadzono warunek (*b* <> 0), zatem nie zastosujemy tu instrukcji **for**. Możemy zastosować instrukcję **while**, której działanie polega na wykonywaniu ciągu instrukcji, dopóki warunek jest spełniany.

Fragment rozwiązania zadania:

Pascal

```
...
Readln(a,b);
while b <> 0 do begin
   iloraz := a/b;
   Writeln('iloraz wynosi ', iloraz:8:2);
   Readln(a, b);
end;
```

C++

```
...
cin >> a >> b;
while(b!=0.0)
{
   iloraz=a/b;
   cout << "Iloraz wynosi " << iloraz << endl;
   cin >> a >> b;
}
```

Ćwiczenie 1.

Otwórz plik *T5_p1.pas* lub *T5_p1.cpp* (CD), w którym zapisane jest rozwiązanie zadania podanego w przykładzie 1, i sprawdź działanie programu dla kilku par danych. Odpowiedz na pytania:

1. W jakim przypadku instrukcje wewnątrz pętli nie zostaną wykonane ani razu?
2. Która instrukcja ma wpływ na zmianę wartości logicznej warunku? Sprawdź, jak będzie działał program, jeśli usuniesz tę instrukcję.

1.2. Zastosowanie instrukcji iteracyjnych repeat i do

Instrukcja iteracyjna **repeat..until**	**repeat** *ciąg instrukcji;* **until** *warunek;*	**Pascal**
Instrukcja iteracyjna **do..while**	**do** instrukcja; **while** *(warunek);*	**C++**

W pętlach **repeat** (Pascal) i **do** (C++), w odróżnieniu od pętli **while**, *warunek* jest sprawdzany po wykonaniu *instrukcji*, więc instrukcja (instrukcje) wewnątrz pętli zostanie wykonana przynajmniej raz, niezależnie od wartości początkowej warunku.

W języku Pascal instrukcje występujące wewnątrz pętli **repeat** są powtarzane, dopóki *warunek* pozostaje niespełniony. Gdy warunek zostanie spełniony, nastąpi wyjście z pętli.

W języku C++ pętla **do** wykonuje się, dopóki warunek jest spełniony. Gdy *warunek* przestanie być spełniany, nastąpi wyjście z pętli. *Instrukcja* może być pojedyncza lub złożona.

Przykład 2. Stosowanie instrukcji repeat w języku Pascal i instrukcji do w języku C++

Napiszemy program, który będzie wczytywał liczby wprowadzone z klawiatury i zliczał wśród nich liczby parzyste. Dla liczby równej zero program powinien zakończyć działanie i podać, ile było liczb parzystych.

Fragment rozwiązania zadania:

Pascal

```
...
repeat
   Readln(x);
   if x mod 2 = 0 then P := P + 1;
until x = 0;
Writeln(P - 1);
...
```

C++

```
...
do
{
          cin >> x;
          if(x%2==0) P++;
} while(x);
cout << P-1 << endl;
```

Uwaga:

W zadaniach podanych w przykładach 1. i 2, aby zakończyć wprowadzanie danych, określaliśmy tzw. **znacznik końca danych**. W tym przypadku jest to liczba zero.

Ćwiczenie 2.

Napisz program realizujący algorytm podany w przykładzie 2. Skompiluj, uruchom i wykonaj program dla wybranych danych.

Wskazówka: Można zmodyfikować rozwiązanie ćwiczenia 18. z tematu 2, w którym wystarczy zmienić zastosowaną pętlę **for** na pętlę **repeat** (Pascal) lub na pętlę **do** (C++).

2. Zastosowanie funkcji w algorytmach iteracyjnych

2.1. Algorytm wyboru minimum z *n* liczb

Algorytm znajdowania minimum (lub maksimum) ma zastosowanie w rozwiązywaniu różnych zadań, nie tylko matematycznych – np. przy wyborze najniższej ceny wśród cen tych samych produktów, przy wyborze zawodnika, który uzyskał najwyższą liczbę punktów, lub ucznia, który uzyskał najwyższą średnią ocen.

Z algorytmu wyboru minimum (lub maksimum) korzysta się również w innych algorytmach, np. w algorytmach sortowania. Jeśli wiemy, jak znaleźć element najmniejszy, to łatwo uporządkować elementy od najmniejszego do największego.

Omówimy przykładowy algorytm znajdowania elementu najmniejszego (szukanie elementu największego przebiega bardzo podobnie).

Opis algorytmu: Wskazujemy dowolny element, np. pierwszy, jako najmniejszy i porównujemy go z drugim elementem. Jeśli drugi element okaże się mniejszy, to on zaczyna być traktowany jako minimum. Następnie porównujemy element aktualnie najmniejszy z trzecim elementem itd. – aż do końca ciągu elementów.

Szukanie elementu najmniejszego to typowy algorytm iteracyjny – powtarzają się w nim operacje porównywania i podstawiania.

Przykład 3. Stosowanie algorytmu wyboru najmniejszego elementu z *n* liczb

Zadanie: Wybierz najmniejszą liczbę wśród *n* liczb.

Dane: liczba naturalna *n*, oznaczająca ilość wprowadzanych liczb, *n* dowolnych liczb rzeczywistych, zapamiętywanych kolejno w zmiennej *x*.

Wynik: wartość elementu najmniejszego: *min*.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź liczbę danych *n*: *n*.
3. Wprowadź pierwszą liczbę *x*: *x*.
4. Zmiennej *min* przypisz wartość liczby *x*: *min* := *x*.
5. Wprowadź kolejną liczbę *x*.
6. Porównaj kolejną liczbę z *min*: *x* < *min*.
7. Jeśli kolejna liczba *x* jest mniejsza od *min*, przypisz jej wartość zmiennej *min*: *min* := *x*.
8. Jeśli nie jest to ostatnia liczba, wróć do kroku 5.
9. Wyprowadź wynik: *min*.
10. Zakończ algorytm.

```
x := 7
min := 7

x := 20
czy x < min?        NIE

x := 6
czy x < min?        TAK
min := x;

x := 15
czy x < min?        NIE
wyprowadzenie liczby 6
```

Rys. 1. *Analiza algorytmu znajdowania minimum dla czterech liczb: 7, 20, 6 i 15*

Ćwiczenie 3.

Narysuj schemat blokowy algorytmu znajdowania minimum z *n* liczb.

Przykład 4. Definicja funkcji znajdowania minimum

Zapisując algorytm znajdowania minimum w postaci programu, zdefiniujemy funkcję *MinN* z jednym parametrem *n* (*n* – liczba elementów). Funkcja będzie zwracać wartość elementu najmniejszego.

Funkcja wywołana jest w programie głównym w instrukcji podstawienia.

Pascal
```
MinWartosc := MinN(LiczbaElementow);
```
C++
```
MinWartosc = MinN(LiczbaElementow);
```

Pascal

```
program Minimum;
var LiczbaElementow, MinWartosc: integer;
function MinN(n: integer): integer;
var i, x, minx: integer;
begin
   for i := 1 to n do begin
                        Write('Podaj liczbę ', i, ': ');
                        Readln(x);
                        if i = 1 then minx := x
                                 else if x < minx then minx := x;
                    end;
   MinN := minx;
end;
begin
   Write('Podaj liczbę elementów: ');
   Readln(LiczbaElementow);
   Writeln('Podaj elementy:');
   MinWartosc := MinN (LiczbaElementow);
   Writeln('Minimum to ', MinWartosc);
end.
```

C++

```
#include <iostream>
using namespace std;
int LiczbaElementow, MinWartosc;
int MinN(int n)
{
              int i, x, minx;
              for(i=0; i<n; i++)
              {
                       cout << "Podaj liczbe " << i+1 << ": ";
                       cin >> x;
                       if(i==0)
                                minx = x;
                       else
                                if(x<minx)        minx = x;
              }
              return minx;
}
int main()
{
   cout << "Podaj liczbe elementow: ";
   cin >> LiczbaElementow;
   cout << "Podaj elementy:" << endl;
   MinWartosc = MinN(LiczbaElementow);
   cout << "Minimum to " << MinWartosc << endl;
   return 0;
}
```

Ćwiczenie 4.

Otwórz plik *T5_p4.pas* lub *T5_p4.cpp* (CD). Wykonaj zapisany w pliku program dla kilku różnych danych.
W programach zostały zdefiniowane dwie funkcje szukające minimum.
Czym się od siebie różnią?

Ćwiczenie 5.

W pliku *T5_c5.pas* i w pliku *T5_c5.cpp* (CD) również są zapisane programy realizujące algorytm znajdowania minimum. Czym różnią się od programów zapisanych w plikach *T5_p4.pas* lub *T5_p4.cpp*? Zwróć uwagę na zastosowane struktury danych.
W programach zostały zdefiniowane dwie funkcje szukające minimum.
Czym się od siebie różnią?

2.2. Algorytm obliczania silni

Obliczanie silni liczby naturalnej *n* to kolejny przykład algorytmu iteracyjnego.

Iteracyjna definicja silni liczby naturalnej *n*:

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n & \text{dla } n > 0 \end{cases}$$

Ćwiczenie 6.

Napisz specyfikację zadania i listę kroków algorytmu obliczania silni.

Przykład 5. Definicja iteracyjna funkcji obliczania silni

Zdefiniujemy iteracyjną wersję funkcji obliczania silni z jednym parametrem formalnym *n*.

Pascal

```
function SilniaIteracyjna(n: word): longint;
var
   i: word;
   wynik: longint;
begin
      wynik := 1;
      for i := 2 to n do wynik := wynik*i;
      SilniaIteracyjna := wynik;
end;
```

C++

```
unsigned long SilniaIteracyjna(unsigned short n)
{
              unsigned short i;
              unsigned long wynik=1;

              for(i=2; i<=n; i++) wynik*=i;
              return wynik;
}
```

Ćwiczenie 7.

Korzystając z funkcji obliczania silni zdefiniowanej w przykładzie 5, napisz program obliczania silni liczby naturalnej *n*. Wywołaj funkcję w programie głównym (w języku Pascal) lub w funkcji `main()` (w języku C++).

3. Algorytm Euklidesa

3.1. Wersja algorytmu Euklidesa z odejmowaniem

Algorytm Euklidesa służy do znajdowania **największego wspólnego dzielnika** (NWD) dwóch liczb naturalnych.

Powtórka z matematyki: **największy wspólny dzielnik** (NWD) dwóch lub więcej liczb naturalnych (różnych od zera) to największa liczba naturalna, przez którą dzieli się bez reszty każda z danych liczb.

Największy wspólny dzielnik liczb (NWD) możemy wykorzystać do skracania ułamków zwykłych (licznik i mianownik dzielimy przez ich NWD), a także do obliczenia ich najmniejszej wspólnej wielokrotności (NWW), np.:

$$NWW(a,b) = \frac{a \cdot b}{NWD(a,b)}.$$

Wyznaczenie NWW umożliwia z kolei sprowadzanie ułamków do wspólnego mianownika, co pozwala wykonać obliczenia przy użyciu możliwie małych liczb.

Przykład 6. Iteracyjna realizacja algorytmu Euklidesa – wersja z odejmowaniem

Zadanie: Przedstaw w postaci listy kroków algorytm znajdowania największego wspólnego dzielnika (NWD) dla dwóch niezerowych liczb naturalnych.

Dane: dwie liczby naturalne: *a, b*.

Wynik: wartość największego wspólnego dzielnika liczb *a* i *b*: *NWD*.

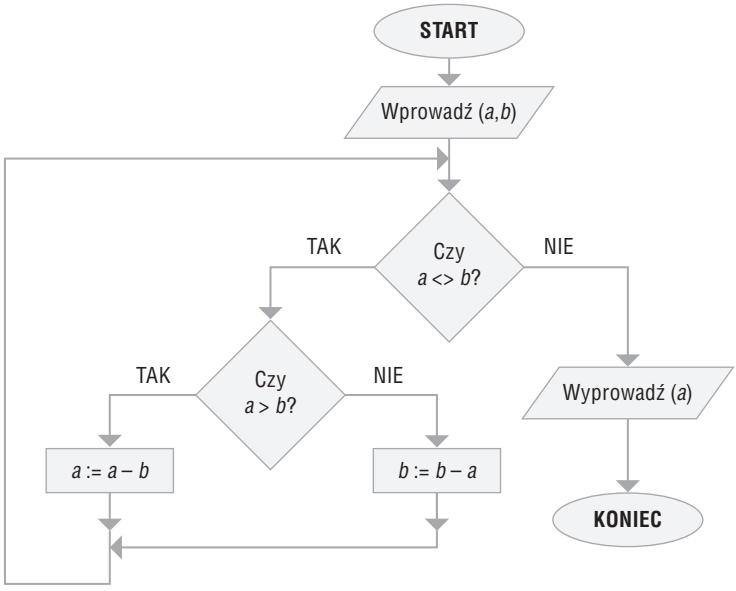
Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartości liczb *a* i *b*.
3. Sprawdź, czy *a* jest różne od *b*.
4. Dopóki *a* nie jest równe *b*, powtarzaj punkt 5;
   w przeciwnym razie przejdź do punktu 6.
5. Od liczby większej odejmij mniejszą i liczbę większą zastąp otrzymaną różnicą.
6. Wyprowadź wynik: *NWD* jest równe pierwszej liczbie.
7. Zakończ algorytm.

Przedstawiona realizacja algorytmu Euklidesa jest tzw. **realizacją z odejmowaniem**.

Ćwiczenie 8.

Napisz program realizujący algorytm Euklidesa w wersji z odejmowaniem.



Rys. 2. *Schemat blokowy algorytmu Euklidesa – wersja z odejmowaniem*

3.2. Wersja algorytmu Euklidesa z dzieleniem

W wersji algorytmu Euklidesa z odejmowaniem odejmowanie posłużyło znalezieniu reszty z dzielenia dwóch liczb. Możemy więc, zamiast odejmowania, od razu zastosować dzielenie z resztą.

Pisząc program w języku Pascal, należy wykorzystać operator **mod** w celu obliczania reszty z dzielenia, np. `reszta := a mod b`.

W języku C++ należy zastosować operator `%`, np. `reszta = a % b`.

Przykład 7. Iteracyjna realizacja algorytmu Euklidesa – wersja z dzieleniem

Wersja algorytmu z dzieleniem polega na wykonywaniu kolejnych dzieleń dwóch liczb, dopóki dzielnik nie osiągnie wartości 0. Wówczas dzielna jest równa NWD i algorytm się kończy. W pętli dzielną zastępujemy dzielnikiem, a dzielnik resztą z dzielenia dzielnej przez dzielnik. Zmienna *k* przechowuje dzielnik.

Zadanie: Przedstaw w postaci listy kroków algorytm znajdowania największego wspólnego dzielnika (NWD) dla dwóch niezerowych liczb naturalnych.

Dane: dwie liczby naturalne: *a, b*.

Wynik: wartość największego wspólnego dzielnika liczb *a* i *b*: *NWD*.

Lista kroków:

1. Zacznij algorytm.
2. Wprowadź wartości liczb *a* i *b*.
3. Zmiennej *k* przypisz wartość zmiennej *b*: *k* := *b*.
4. Zmiennej *b* przypisz wartość wyrażenia: *a* **mod** *b*: *b*:= *a* mod *b*.
5. Zmiennej *a* przypisz wartość zmiennej *k*: *a* := *k*.
6. Jeśli *b* <> 0, przejdź do kroku 3.
7. Wyprowadź wynik: *NWD* jest równe *a*.
8. Zakończ algorytm.

Ćwiczenie 9.

Na podstawie listy kroków podanej w przykładzie 7. narysuj schemat blokowy algorytmu Euklidesa.

Ćwiczenie 10.

Sprawdź liczbę wykonywanych operacji odejmowania w wersji algorytmu Euklidesa z odejmowaniem i liczbę operacji dzielenia w wersji z dzieleniem dla następujących par liczb: (48, 9), (133, 49), (64, 16), (549, 145), (621, 126).

4. Jednoczesne znajdowanie największego i najmniejszego elementu w zbiorze

Znalezienie największego i najmniejszego elementu w zbiorze pozwala na określenie tzw. **rozpiętości** zbioru, czyli różnicy pomiędzy wartością największą a najmniejszą elementów występujących w danym zbiorze.

4.1. Algorytm naiwny

Algorytm naiwny szukania elementu najmniejszego i największego w zbiorze składającym się z n elementów polega na zastosowaniu algorytmu szukania minimum dla n elementów, usunięciu elementu najmniejszego ze zbioru, a następnie zastosowaniu algorytmu szukania maksimum dla zbioru składającego się z n-1 elementów (czyli pomniejszonego o element najmniejszy).

Ćwiczenie 11.

Zmodyfikuj program zapisany w pliku *T5_p4.pas lub T5_p4.cpp* (CD), tak aby realizował algorytm naiwny znajdowania najmniejszego i największego elementu w zbiorze.

4.2. Algorytm optymalny – metoda „dziel i zwyciężaj”

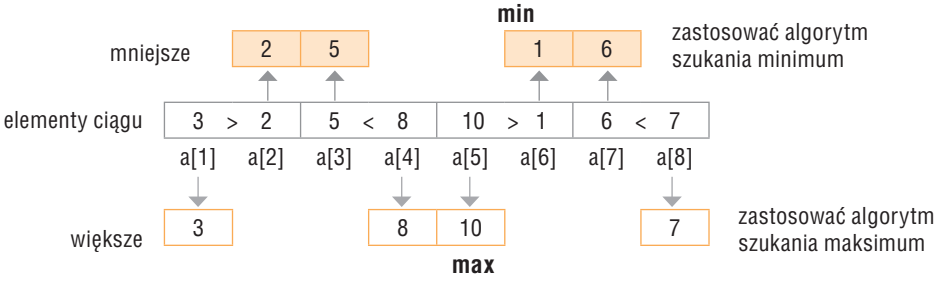
Optymalnym algorytmem jednoczesnego znajdowania elementu najmniejszego i największego jest zastosowanie metody „**dziel i zwyciężaj**”. Nazwa metody pochodzi od angielskiego sformułowania *divide and conquer* (parafraza słynnej łacińskiej formuły politycznej *divide et impera* – dziel i rządź). „Dziel” oznacza podział zadania na mniejsze części, a „zwyciężaj” – znalezienie rozwiązania (osiągnięcie sukcesu) w każdej z nich oddzielnie. Zastosowanie tej metody pokażemy na przykładzie algorytmu jednoczesnego szukania najmniejszego i największego elementu zbioru.

Zbiór dzielimy na dwa podzbiory w sposób następujący: porównujemy parami liczby: pierwszą z drugą, trzecią z czwartą itd. (rys. 3.). Liczby mniejsze zapisujemy w jednym podzbiorze, a większe w drugim. W pierwszym podzbiorze, korzystając z algorytmu na znalezienie minimum, znajdujemy element najmniejszy. W drugim podzbiorze, korzystając z algorytmu na znalezienie maksimum, znajdujemy element największy.

Na rysunku 3. został przedstawiony przypadek, gdy liczba elementów zbioru jest parzysta (dzieli się przez 2). W przypadku nieparzystej liczby elementów ostatni element pozostanie wolny. Zapamiętuje się go w zmiennej pomocniczej i na koniec porównuje

z najmniejszym i największym znalezionym elementem. W szczególnym przypadku to właśnie ten element może być minimalny (bądź maksymalny).

Obliczmy liczbę porównań elementów w zbiorze n-elementowym w omówionych metodach.



Podział zbioru na dwa podzbiory:
- elementy mniejsze – w polach z pomarańczowym tłem,
- elementy większe – w polach z pomarańczowymym obramowaniem.

Rys. 3. *Metoda „dziel i zwyciężaj" w algorytmie znajdowania równocześnie minimum i maksimum*

W algorytmie naiwnym wykonywanych jest $n-1$ porównań podczas znajdowania minimum i $n-2$ porównań przy znajdowaniu maksimum, czyli razem wykonywane są:

$$n-1+n-2=2n-3 \text{ porównania.}$$

W metodzie „dziel i zwyciężaj" (w przypadku parzystej liczby elementów) wykonujemy najpierw $\frac{n}{2}$ porównań, aby podzielić zbiór na dwie części. W każdej części składającej się z $\frac{n}{2}$ elementów wykonujemy $\frac{n}{2}-1$ porównań, czyli razem:

$$\frac{n}{2}+\frac{n}{2}-1+\frac{n}{2}-1=\frac{3n}{2}-2 \text{ porównań.}$$

Stosując metodę „dziel i zwyciężaj", wykonujemy mniejszą liczbę porównań niż w algorytmie naiwnym. Na przykład dla zbioru składającego się ze 100 elementów, stosując metodę „dziel i zwyciężaj", wykonamy 148 porównań, a w algorytmie naiwnym – 197.

Ćwiczenie 12.

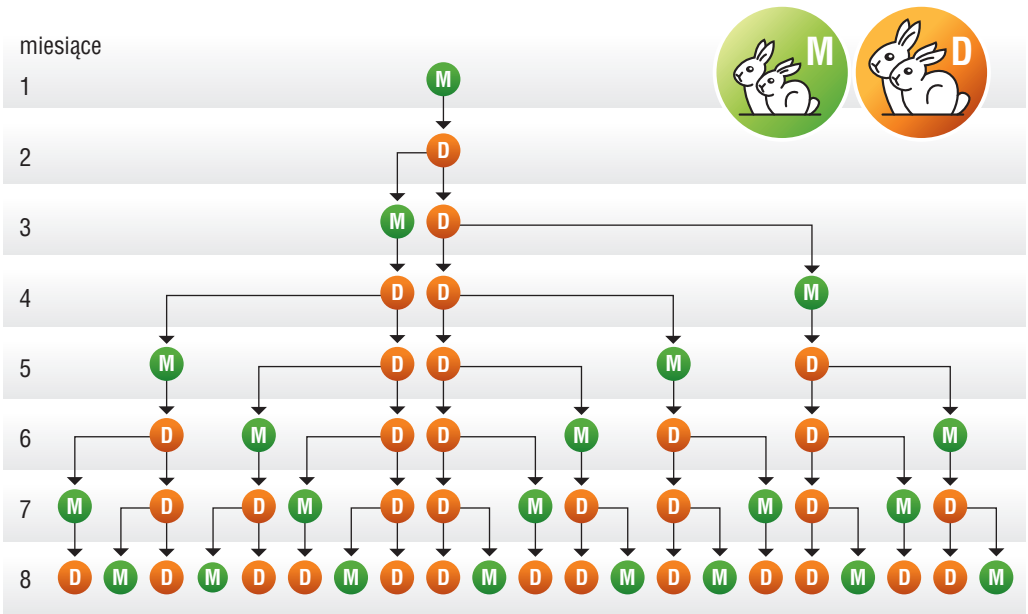
Przedstaw na rysunku algorytm jednoczesnego znajdowania minimum i maksimum dla zbioru liczb innego niż podany na rysunku 3. Przyjmij nieparzystą liczbę elementów. Ile porównań elementów wykonuje się w przypadku nieparzystej liczby elementów?

Aby zapisać algorytm w językach Pascal lub C++, można wykorzystać w programie dwie zdefiniowane wcześniej funkcje: minimum i maksimum. Elementy zbioru trzeba wczytać do tablicy. Podzbiory elementów mniejszych i większych można zapamiętywać w dwóch dodatkowo zadeklarowanych tablicach. Ten sposób nie jest optymalny (duża liczba deklarowanych zmiennych), ale nietrudny do wykonania. Można też wykonać to zadanie, nie deklarując dodatkowych tablic.

5. Liczby Fibonacciego

W roku 1202 wybitny włoski matematyk okresu średniowiecza, Leonardo Pisano, zwany Fibonaccim, przedstawił problem, który do tej pory dostarcza natchnienia rzeszom matematyków. Dotyczy on liczebności populacji królików.

Opis algorytmu: Początkowo rodzi się para królików, która po miesiącu osiąga dojrzałość i po następnym miesiącu wydaje na świat parę królików, po miesiącu następną parę i tak dalej, tzn. pierwsza para co miesiąc wydaje na świat potomstwo (parę królików). Z kolei każda nowa para po miesiącu osiąga dojrzałość i po każdym następnym miesiącu również wydaje na świat kolejną parę królików. Proces ten nie ma końca. Zakładamy, że króliki żyją wiecznie.



Rys. 4. *Schemat przedstawia liczebność populacji królików w pierwszych ośmiu miesiącach doświadczenia (M – para królików młodych, D – para królików dojrzałych); ósma liczba Fibonacciego to 21 par*

Pytanie brzmi: jaka jest liczba królików po n miesiącach?

Zauważmy, że liczba królików w danym miesiącu (z wyjątkiem pierwszego i drugiego miesiąca) zależy od dwóch wartości: liczby par królików dorosłych i liczby par królików młodych w poprzednim miesiącu. Tych ostatnich jest z kolei tyle, ile było par królików dorosłych dwa miesiące wcześniej.

Jeśli oznaczymy całkowitą liczbę królików w n-tym miesiącu jako F_n, to przyjmie ona postać:

$$F_n = \begin{cases} 1 & \text{dla } n \in \{1, 2\} \\ F_{n-1} + F_{n-2} & \text{dla } n \geq 3. \end{cases}$$

Ćwiczenie 13.

Oblicz na kartce trzecią i czwartą liczbę ciągu Fibonacciego. Czy możesz obliczyć piątą liczbę bez wcześniejszego obliczenia trzeciej i czwartej?

Liczba F_n opisuje całkowitą wielkość populacji królików po upływie n miesięcy i nazywana jest **liczbą Fibonacciego**. Natomiast ciąg liczb naturalnych, w którym każda liczba (z wyjątkiem pierwszej i drugiej) jest sumą dwóch poprzednich, nazywamy **ciągiem Fibonacciego**.

Oto kilka początkowych liczb tego ciągu: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233…

By obliczyć poszczególne liczby Fibonacciego, można posłużyć się algorytmem iteracyjnym. W postaci iteracyjnej obliczenia zaczynamy „od początku", przyjmując, że wartość dwóch pierwszych liczb Fibonacciego wynosi 1.

Ćwiczenie 14.

Otwórz plik *T5_c14.pas* lub *T5_c14.cpp* (CD). W programie jest zdefiniowana funkcja iteracyjnego sposobu obliczania liczb Fibonacciego – z jednym parametrem formalnym *n*. Wykonaj program dla kilku różnych danych.

6. Schemat Hornera

Schemat Hornera jest algorytmem służącym do szybkiego obliczania wartości wielomianu, a także przeliczania na postać dziesiętną liczb zapisanych w innym systemie liczbowym oraz szybkiego podnoszenia do potęgi.

W.G. Horner
– matematyk angielski;
żył w latach 1786-1837.

Przykład 8. Zastosowanie schematu Hornera do obliczania wartości wielomianu

Weźmy funkcję kwadratową $y = a_0x^2 + a_1x + a_2$. Jest ona wielomianem stopnia drugiego o współczynnikach a_0, a_1, a_2 (a_0, a_1, a_2, $\in R$ i $a_0 \neq 0$). Zwykle, gdy chcemy obliczyć jego wartość, wykonujemy trzy działania mnożenia i dwa dodawania. Natomiast gdy wielomian uporządkujemy w ten sposób, że zgrupujemy dwa pierwsze wyrazy i wyciągniemy wspólny czynnik x przed nawias, to otrzymamy wielomian o postaci: $y = (a_0x + a_1)x + a_2$. Dla obliczenia wartości tak zapisanego wielomianu wystarczy wykonanie dwóch działań mnożenia i dwóch dodawania.

Wielomian trzeciego stopnia $W(x) = a_0x^3 + a_1x^2 + a_2x + a_3$ po rozpisaniu według powyższej zasady przyjmie postać: $W(x) = ((a_0x + a_1)x + a_2)x + a_3$.

Dla obliczenia jego wartości wystarczy wykonać trzy działania mnożenia i trzy dodawania. Natomiast w postaci klasycznej dla obliczenia wartości wielomianu wykonujemy sześć działań mnożenia i trzy dodawania.

Uogólniając, wielomian n-tego stopnia, który ma postać:

$W(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + ... + a_{n-1}x + a_n$ dla $n \geq 0$, [1]

można uporządkować następująco:

$W(x) = (...(((a_0x + a_1)x + a_2)x + a_2)x + ... + a_{n-1})x + a_n$.

W celu obliczenia jego wartości będziemy wykonywać obliczenia zgodnie ze znaną zasadą kolejności wykonywania działań, poczynając od najbardziej „zagnieżdżonych" nawiasów. Za początkową wartość wielomianu należy przyjąć wartość współczynnika a_0 przy najwyższej potędze zmiennej. Za każdym razem należy aktualną wartość wielomianu pomnożyć przez x i dodać kolejny współczynnik.

Algorytm ten zwany jest schematem Hornera.

Algorytm obliczania wartości wielomianu, zwany **schematem Hornera**, można rozpisać następująco:
$W(x) := a_0$ (początkowa wartość wielomianu)
$W(x) := W(x)x + a_i$ dla $i = 1, 2, 3,..., n$ (iteracja).

Ćwiczenie 15.

Oblicz, ile należy wykonać działań mnożenia i dodawania dla obliczenia wartości wielomianu $W(x)$, posługując się wzorem [1].

Obliczenie wartości wielomianu stopnia n za pomocą schematu Hornera wymaga wykonania n mnożeń i n dodawań. Porównując liczbę operacji arytmetycznych, jakie należy wykonać w celu obliczenia wartości wielomianu $W(x)$ za pomocą wzoru [1] i za pomocą schematu Hornera, dochodzimy do wniosku, że drugi sposób jest mniej pracochłonny.

Ćwiczenie 16.

Rozpisz wielomian czwartego stopnia tak, aby można było obliczyć jego wartość z zastosowaniem schematu Hornera.

7. Wydawanie reszty metodą zachłanną

Algorytm zachłanny – algorytm, który w celu wyznaczenia rozwiązania w każdym kroku dokonuje zachłannego, tj. najkorzystniejszego w danym kroku, rozwiązania częściowego.

Z problemem wydawania reszty każdy spotkał się wielokrotnie. Resztę wydajemy, używając jak najmniejszej możliwej liczby banknotów lub bilonu. Metoda zachłanna zakłada użycie do wydania reszty największych dostępnych nominałów (ale mniejszych od lub równych wydawanej wartości).

Zanim zaczniemy wydawać resztę, musimy określić dostępne nominały. Pisząc program w językach Pascal lub C++, listę dostępnych nominałów umieścimy w tablicy, której elementy powinny być posortowane malejąco.

Opis algorytmu:

Algorytm zaczynamy od sprawdzenia, czy w tabeli nominałów jest wartość mniejsza lub równa reszcie (oznaczmy ją R). Jeśli odnaleziony nominał N jest równy R, to algorytm się kończy. Jeśli nie, to wybieramy największą możliwą liczbę L sztuk znalezionego nominału N mniejszego od R (zakładamy, że tablica nominałów jest posortowana malejąco). Liczbę tę otrzymujemy, dzieląc R przez wartość odnalezionego nominału (liczba nominałów to oczywiście część całkowita z dzielenia). Następnie od reszty R odejmujemy wartość znalezionego nominału pomnożoną przez liczbę nominałów wynikającą z dzielenia. Szukanie powtarzamy wśród kolejnych nominałów dla pomniejszonej wartości R do czasu aż wypłacimy całą sumę.

Przykład 9. Zastosowanie metody zachłannej do wydawania reszty

Korzystając z podanego opisu, znajdziemy nominały, jakimi zostanie wydana reszta w wysokości 69 zł.
Zbiór dostępnych nominałów to: {50, 20, 10, 5, 2, 1}.

Sposób wykonania algorytmu dla *R* równego 69.

Krok 1.
Największy nominał, nie większy niż *R* to 50 zł.
69 **div** 50 = 1 wypłacamy jeden banknot o nominale 50 zł.
R := 69 – 50 · 1 = 19 do wydania zostało 19 zł.

Krok 2.
Największy nominał, nie większy niż aktualne *R* to 10 zł.
19 **div** 10 = 1 wypłacamy jeden banknot o nominale 10 zł.
R := 19 – 10 · 1 = 9 do wydania zostało 9 zł.

Krok 3.
Największy nominał, nie większy niż aktualne *R* to 5 zł.
9 **div** 5 =1 wypłacamy jedną monetę o nominale 5 zł.
R := 9 – 5 · 1 = 4 do wydania zostały 4 zł.

Krok 4.
Największy nominał, nie większy niż aktualne *R* to 2 zł.
4 **div** 2 = 2 wypłacamy dwie monety o nominale 2 zł.
R := 4 – 2 · 2 = 0 wypłaciliśmy całą kwotę.
Reszta została wydana następująco: 1 · 50 zł + 1 · 10 zł + 1 · 5 zł + 2 · 2 zł = 69 zł.

Uwaga: Operator **div** oznacza całkowitą część z dzielenia w języku Pascal.
W języku C++ należy zastosować operator / , którego działanie zależy od typu danych (w przypadku liczb całkowitych wynik również będzie całkowity).

Ćwiczenie 17.

Na podstawie opisu algorytmu wydawania reszty metodą zachłanną oraz przykładu 9. napisz listę kroków oraz narysuj schemat blokowy do podanej specyfikacji.
Dane: Zbiór nominałów {50, 20, 10, 5, 2, 1}, reszta *R*.
Wyniki: Liczba nominałów składających się na resztę *R*.

Warto zapamiętać

- W każdym języku programowania występują instrukcje, które umożliwiają zapis algorytmów iteracyjnych. W językach Pascal i C++ poznaliśmy trzy takie instrukcje:
  - **for**,
  - **while**,
  - **repeat** (Pascal) i **do** (C++).

  Instrukcję **for** stosujemy, gdy mamy z góry określoną liczbę iteracji; dwie pozostałe – gdy liczba iteracji zależy od warunku.
- Algorytmy: znajdowania elementu minimalnego i maksymalnego, Euklidesa, wyznaczania liczb Fibonacciego, obliczania wartości wielomianu za pomocą schematu Hornera i wydawania reszty metodą zachłanną, można zrealizować, stosując technikę iteracji.
- Aby jednocześnie znaleźć element najmniejszy i największy, możemy zastosować algorytm naiwny, ale efektywniejsza jest metoda „dziel i zwyciężaj". Korzystając z niej, wykonuje się mniej operacji porównania elementów.

Pytania, problemy

1. Czym różni się działanie instrukcji iteracyjnej **while** od działania instrukcji **repeat** w języku Pascal i **while** od **do** w języku C++? Pokaż na przykładach.
2. Omów na przykładzie ciągu liczb (-1, 0, 80, 12, 12, -4, 9) algorytm znajdowania najmniejszego elementu.
3. Wymień przykładowe zastosowania algorytmu Euklidesa.
4. Omów iteracyjną realizację algorytmu Euklidesa znajdowania NWD. Wyjaśnij różnicę między wersją z odejmowaniem a wersją z dzieleniem.
5. Na czym polega metoda „dziel i zwyciężaj"?
6. Wyjaśnij zastosowanie metody „dziel i zwyciężaj" w jednoczesnym znajdowaniu najmniejszej i największej liczby punktów uzyskanych ze sprawdzianu.
7. Dlaczego algorytm naiwny jest mniej optymalny od metody „dziel i zwyciężaj" w jednoczesnym szukaniu elementu najmniejszego i największego?
8. Przedstaw na odręcznym rysunku algorytm wyznaczania liczb Fibonacciego.
9. Pokaż zastosowanie schematu Hornera w obliczaniu wartości wielomianu trzeciego stopnia.
10. Omów metodę zachłanną wydawania reszty wynoszącej 178 zł. Dostępne są nominały [100 zł, 50 zł, 20 zł, 10 zł, 5 zł, 2 zł, 1 zł].

Zadania

1. Zmodyfikuj program utworzony w ćwiczeniu 2, dodając możliwość zliczania liczb nieparzystych.
2. Napisz program, który będzie wczytywał liczby całkowite, wprowadzane z klawiatury, aż do wystąpienia liczby niedodatniej, oraz obliczał i wyprowadzał średnią arytmetyczną tych liczb. Jaka liczba będzie tu znacznikiem końca danych?
3. Ile razy będą wykonane pętle pokazane w ramkach? Rozważ różne przypadki w zależności od wprowadzonych danych.

Pascal

```
a) Readln(a,b);
   while b = a do Writeln('liczby rowne');
b) Readln(x);
   repeat
          Writeln(x)
   until x = 2;
```

C++

```
a) cin >> a >> b;
   while(b==a) cout << "liczby rowne" << endl;
b) cin >> x;
   do cout << x << endl; while(x!=2);
```

4. Zmodyfikuj program zapisany w zadaniu 10. z tematu 4. tak, aby wczytywał znaki wprowadzane z klawiatury aż do wprowadzenia znaku „@”.
5. Napisz program, który będzie wczytywał liczby aż do wystąpienia zera i dzielił je na dwa podzbiory: liczby parzyste i liczby nieparzyste.
6. Przedstaw w postaci listy kroków algorytm wyboru najwyższego z uczniów w klasie.
7. Napisz program znajdowania maksimum z *n* liczb. Zdefiniuj funkcję *MaxN*.
8. Narysuj schemat blokowy algorytmu obliczania silni.
9. Napisz program realizujący algorytm Euklidesa w wersji z dzieleniem.
10. Zapisz w postaci listy kroków algorytm znajdowania elementu najmniejszego i największego w zbiorze, wykorzystując metodę „dziel i zwyciężaj”.
11. Napisz program realizujący algorytm jednoczesnego znajdowania elementu najmniejszego i największego, wykorzystując metodę „dziel i zwyciężaj”. Dobierz odpowiednie struktury danych.
12. Utwórz schemat blokowy realizujący schemat Hornera dla wielomianu stopnia *n* o zadanych współczynnikach a_0, a_1, a_2, a_3, ..., a_n. Prześledź jego działanie dla $n = 7$, $a_0 = -2$, $a_1 = 4$, $a_2 = -5$, $a_3 = 3$, $a_4 = 9$, $a_5 = 3$, $a_6 = -1$, $a_7 = 6$.

Dla zainteresowanych

13. Napisz program, który wczytuje *n*-elementowy zbiór liczb całkowitych i wyprowadza różnicę między elementem minimalnym i maksymalnym.
14. Napisz program, który wczytuje *n*-elementowy zbiór liczb całkowitych i wyprowadza element najbliższy wartości średniej tego zbioru.
15. Zastosuj algorytm Euklidesa do obliczenia NWW dwóch liczb.
16. Zastosuj algorytm Euklidesa do dodawania ułamków zwykłych.
17. Napisz program realizujący algorytm wydawania reszty metodą zachłanną.
18. Napisz program umożliwiający wprowadzenie dwóch liczb całkowitych większych od 1 i wyprowadzający tę z nich, której suma cyfr jest większa.
    Wskazówka: Zadanie można rozwiązać, korzystając z instrukcji **`while`** i operatorów: **`mod`** oraz **`div`** w języku Pascal lub instrukcji **`while`** i operatorów `%` i `/` w języku C++.
19. Napisz program sprawdzający, czy wprowadzona liczba jest liczbą pierwszą.
    Wskazówka: Algorytm ten jest zrealizowany w arkuszu kalkulacyjnym w przykładzie 4. w temacie 13. Jakie powinny być wartości sprawdzanych dzielników?

20. Liczbami bliźniaczymi nazywamy dwie liczby pierwsze p i q, takie że $q = p + 2$ (np. 3 i 5, 5 i 7). Napisz program znajdujący 20 pierwszych par liczb bliźniaczych.
21. Napisz program realizujący obliczanie wartości wielomianu za pomocą schematu Hornera.

Przeczytaj, jeśli chcesz wiedzieć więcej...

W **algorytmie Euklidesa** celem jest znalezienie największej wspólnej miary dwóch odcinków o długościach a i b (lub największego wspólnego dzielnika dwóch liczb a i b). Euklides opisał ten algorytm w swoim dziele zatytułowanym *Elementy* ok. 300 lat p.n.e., gdy jeszcze nie było znane pojęcie *algorytm*.

Przydomek Fibonacci oznacza: Filius (z łac. „syn") Bonacciego. Fibonacci nazywał się w rzeczywistości Leonardo Pisano (Leonardo z Pizzy). Żył w latach ok. 1170-1240. Studiował księgi Muhammada Al-Chuwarizmiego i w 1202 roku opublikował dzieło *Liber Abaci*, dzięki któremu Europejczycy poznali cyfry arabskie i dziesiętny system liczenia. Właśnie w tym dziele Fibonacci zawarł rozważania na temat liczebności populacji królików...

Ciąg Fibonacciego, z pozoru prosty, ma wiele interesujących własności. Istnieje nawet towarzystwo Fibonacci Association, zajmujące się publikowaniem wyników licznych prac badawczych dotyczących tego zagadnienia.

Ciekawą własność mają ilorazy kolejnych wyrazów ciągu Fibonacciego. W tabeli 1. zostały one przedstawione z dokładnością do sześciu miejsc po przecinku. Łatwo zaobserwować, jak zbliżają się do liczby $1{,}618033988749894848820... = \frac{1+\sqrt{5}}{2}$,

zwanej **złotą średnią** lub **złotym podziałem** (z łac. *divina proportio* – „boski podział"). Liczba ta jest proporcją najczęściej spotykaną w przyrodzie i stosowaną już przez starożytnych mistrzów. W ciele ludzkim – i to zarówno w całej jego postaci, jak i w wielu jego częściach – występują złote proporcje. Złoty podział można także odnaleźć chociażby w układzie listków na gałęzi.

n	F_n	F_{n+1}/F_n
1	1	1,000000
2	1	2,000000
3	2	1,500000
4	3	1,666667
5	5	1,600000
6	8	1,625000
7	13	1,615385
8	21	1,619048
9	34	1,617647
10	55	1,618182
11	89	1,617978
12	144	1,618056
13	233	1,618026
14	377	1,618037
15	610	1,618033

Tabela 1. *Ilorazy kolejnych liczb Fibonacciego*