10. Poszukujemy liczby

Na pewno zdarzyło ci się kiedyś szukać konkretnej rzeczy w jakimś zbiorze, np. książki na regale lub ubrania w szafie. Dużo łatwiej to zrobić, gdy zbiór jest uporządkowany zgodnie z pewną zasadą, np. książki według alfabetu, a ubrania według kategorii. Możesz wtedy zawęzić poszukiwania np. do litery P albo do spodni, zamiast przeglądać każdą rzecz po kolei. Czasami jednak mamy do czynienia ze zbiorami, których elementy nie są uporządkowane, lub nie wiemy, czy są uporządkowane. Dlatego w tym temacie zajmiemy się wyszukiwaniem liczb w zbiorach zarówno uporządkowanych, jak i nieuporządkowanych.

Cele lekcji

- Wyszukasz element w zbiorze nieuporządkowanym i uporządkowanym.
- Zastosujesz algorytmy przeszukiwania liniowego i binarnego.
- Poznasz kolejne sposoby przekazania parametru do funkcji – przez wskaźnik oraz przez referencję.
- Znajdziesz jednocześnie minimum i maksimum w zbiorze.

10.1. Sformułowanie problemu wyszukiwania

W programach wyszukujących określony element w zbiorze danymi wejściowymi będzie ciąg n losowych liczb całkowitych: $a_0, a_1, \ldots, a_{n-1}$. Liczby zapamiętamy w **tablicy**.

Tablica, s. 142

Funkcja `main` w programach, które napiszemy (niezależnie od zastosowanej metody wyszukiwania), powinna uwzględniać następujące etapy:

1. Wylosowanie elementów tablicy.
2. Wypisanie elementów tablicy.
3. Wyszukanie elementu w tablicy.
4. Wypisanie wyniku wyszukiwania.

Aby zrealizować dwa pierwsze etapy, zdefiniujemy funkcje `Losuj` i `Wypisz`. Pierwsza z nich wylosuje elementy tablicy, a druga wypisze wylosowane liczby.

Liczba elementów tablicy, czyli rozmiar tablicy, powinna być znana w momencie kompilacji programu. Dlatego rozmiar określimy za pomocą **stałej**, którą zdefiniujemy przed funkcjami. Jeśli chcielibyśmy sprawdzić działanie programu dla tablicy o innej liczbie elementów, wystarczy wówczas wprowadzić zmianę tylko w jednej linii kodu – w **definicji stałej** podać inny rozmiar tablicy.

Stała, s. 125

Definicja stałej, s. 125

Fragment kodu źródłowego programu wyszukującego element tablicy może wyglądać następująco. Fragment ten zawiera definicje stałej oraz funkcji losującej i wypisującej tablicę liczb.

Fragment kodu źródłowego programu wyszukującego element w tablicy – definicje stałej oraz funkcji losującej i wypisującej elementy tablicy

```
1.  #include <iostream>
2.  #include <cstdlib>
3.  #include <ctime>
4.
5.  using namespace std;
6.
7.  const int N=10;
8.
9.  void Losuj(int A[])
10. {
11.     for (int i=0;i<N;i++) A[i]=rand()%100;
12. }
13.
14. void Wypisz(int A[])
15. {
16.     for (int i=0;i<N;i++) cout<<A[i]<<" ";
17.     cout<<endl;
18. }
```

Warto wiedzieć

W środowisku Code::Blocks z kompilatorem GCC można określić rozmiar tablicy za pomocą zmiennej:

```
int n;
cout<<"Rozmiar tablicy: ";
cin>>n;
int A[n];
```

Nie jest to jednak rozwiązanie zgodne ze standardem języka C++ może nie działać w innych środowiskach lub gdy użyje się innego kompilatora.

W linii 7 została zdefiniowana stała `N` o wartości 10, określająca rozmiar tablicy. Parametrem każdej z funkcji `Losuj` i `Wypisz` jest tablica. Instrukcja w linii 11 wypełnia tablicę losowymi liczbami z zakresu od 0 do 99. Zakres losowania został ograniczony, żeby łatwo było zinterpretować działanie programu. Otrzymanie losowej liczby całkowitej nieujemnej umożliwia **funkcja** `rand`. Aby móc z niej skorzystać, trzeba dołączyć do programu bibliotekę `cstdlib`. Zakres losowania ograniczyliśmy, biorąc resztę z dzielenia wylosowanej liczby przez 100.

Funkcja `rand`

Przypomnijmy, że **przekazanie parametru przez wartość** oznacza, że parametr funkcji jest chroniony, tzn. wszystkie operacje są wykonywane na kopii parametru, traktowanego jak **zmienna lokalna**. W przypadku, gdy parametrem funkcji jest tablica, w rzeczywistości nie jest nim sama tablica, ale jej adres – tablice w języku C++ są reprezentowane przez adresy (wskaźniki). Oznacza to, że wszystkie operacje są wykonywane na oryginalnej tablicy, a dokonane w niej zmiany zapamiętywane po zakończeniu działania funkcji. Taki sposób przekazania parametru nazywamy **przekazaniem parametru przez wskaźnik**. Tak więc wartości zwracane przez funkcję `Losuj` zostaną zachowane po zakończeniu jej działania.

Przekazanie parametru przez wartość, s. 108

Zmienna lokalna, s. 108

Przekazanie parametru przez wskaźnik

Fragment kodu źródłowego funkcji `main` wywołującej funkcje `Losuj` i `Wypisz` może wyglądać następująco:

Fragment kodu

W linii 1 zadeklarowana jest tablica `N` liczb całkowitych. Instrukcja w linii 2 inicjuje generator liczb losowych i uzależnia losowanie liczb od czasu wskazywanego przez zegar komputera. Jeśli nie dodalibyśmy tej instrukcji, po każdym uruchomieniu programu wynikiem funkcji `Losuj` byłby ten sam ciąg liczb. Ustawienie punktu startowego dla mechanizmu losującego umożliwia **funkcja** `srand`. Z kolei dzięki **funkcji** `time` zwrócimy aktualny czas w postaci liczby całkowitej. Korzystanie z funkcji `srand` wymaga dołączenia biblioteki `cstdlib`, a korzystanie z funkcji `time` – biblioteki `ctime`.

Warto wiedzieć

Żeby wylosować liczbę z przedziału [*a*; *b*], należy użyć wyrażenia:
`rand()%(b-a+1)+a`

Funkcja `srand`, funkcja `time`

Ćwiczenie 1

Napisz program losujący i wypisujący tablicę *n* liczb całkowitych nieujemnych, uwzględniający funkcje `Losuj` i `Wypisz`. Sprawdź działanie programu dla różnych wartości stałej określającej rozmiar tablicy.

Warto wiedzieć

Dzięki temu, że podczas przekazania parametru przez wskaźnik nie jest tworzona kopia tablicy, oszczędza się pamięć komputera.

Zapamiętaj

W języku C++ rozmiar tablicy określamy najczęściej za pomocą stałej. Jeśli tablica jest parametrem funkcji, to jest ona przekazywana przez wskaźnik (adres). Wszystkie operacje na takim parametrze są wykonywane na oryginalnej tablicy i zmiany są pamiętane po zakończeniu działania funkcji.

10.2. Poszukujemy liczby w zbiorze nieuporządkowanym

Jeżeli szukamy danego elementu w zbiorze i nie wiemy, czy zbiór jest uporządkowany, musimy sprawdzać każdy element po kolei. Jeśli pierwszy element nie jest równy szukanemu, sprawdzamy drugi itd. Czynności powtarzamy, dopóki nie odnajdziemy elementu lub nie przejrzymy wszystkich elementów zbioru. Taki sposób postępowania nazywamy **przeszukiwaniem liniowym** lub **sekwencyjnym**.

Przeszukiwanie liniowe (sekwencyjne)

Oto specyfikacja problemu wyszukiwania elementu w zbiorze nieuporządkowanym:

Specyfikacja

Dane: `A[0..n-1]` – tablica *n* liczb całkowitych,
x – liczba całkowita.

Wynik: wartość **`prawda`**, gdy liczba *x* znajduje się w tablicy, **`fałsz`** – w przeciwnym przypadku.

Funkcję sprawdzającą, czy liczba x znajduje się w n-elementowej tablicy A, można zapisać w pseudokodzie następująco:

```
funkcja SzukajLin(A[],x)
    dla i ← 0, 1, ..., n – 1 wykonuj
        jeśli A[i] = x to zwróć prawda i zakończ
    zwróć fałsz i zakończ
```

Dobra rada

Możesz też ustalić, że jeśli szukana liczba jest w tablicy, wynikiem funkcji `SzukajLin` będzie pozycja tej liczby. Natomiast jeśli liczby nie ma w tablicy, wynikiem funkcji może być wartość n (indeks spoza zakresu).

Ćwiczenie 2

Napisz program, który wylosuje i wypisze tablicę liczb całkowitych, a następnie wczyta liczbę całkowitą i wypisze informację, czy ta liczba znajduje się w tablicy. Wyszukiwanie liczby powinna realizować funkcja `SzukajLin`, działająca zgodnie z algorytmem zapisanym w powyższym pseudokodzie.

Zapamiętaj

Jeśli nie wiemy, czy zbiór, w którym szukamy elementu, jest uporządkowany, stosujemy przeszukiwanie liniowe (sekwencyjne). Przeglądamy wówczas kolejne elementy zbioru tak długo, aż znajdziemy szukany element lub przejrzymy wszystkie elementy.

10.3. Poszukujemy liczby w zbiorze uporządkowanym

Gdy szukamy elementu w zbiorze uporządkowanym, możemy ograniczyć liczbę porównań. Zrobimy to poprzez stopniowe zawężanie zbioru, który przeszukujemy. Aby zilustrować sposób postępowania, rozważmy jedną z wersji gry w 20 pytań: pierwszy gracz wybiera liczbę całkowitą z określonego przedziału, natomiast drugi musi odgadnąć, jaką liczbę pomyślał ten pierwszy. Zgadujący może pytać, czy pomyślana liczba znajduje się w podanym przez niego przedziale.

Załóżmy, że liczba całkowita pomyślana przez pierwszego gracza znajduje się w przedziale $[a; b]$. Dla zgadującego najskuteczniejszą strategią będzie podzielenie tego przedziału na dwa równoliczne przedziały (z dokładnością do 1 dla nieparzystej długości przedziału). Najpierw wyznacza środek przedziału (c) ze wzoru $c = (a + b)$ **div** 2. Następnie pyta, czy szukana liczba znajduje się w przedziale $[a; c]$. Ponieważ poszukiwana liczba musi należeć do jednego z przedziałów $[a; c]$ lub $[c + 1; b]$, w zależności od odpowiedzi pierwszego gracza zgadujący odrzuca jeden z przedziałów, czyli połowę liczb. W kolejnych krokach postępuje podobnie. Gdy zawęzi przedział poszukiwań do jednej liczby, ta liczba jest rozwiązaniem.

Przeszukiwanie binarne

Opisany sposób wyszukiwania elementu nosi nazwę **przeszukiwania**

Tabela 10.1 przedstawia przykład wyszukiwania w przedziale [1; 100] liczby pomyślanej przez pierwszego gracza (tu liczby 65).

Iteracja pętli	Początek przedziału (a)	Koniec przedziału (b)	Środek przedziału (c)	Przedział, o który pyta zgadujący	Uc od
1	1	100	50	[1; 50]	
2	51	100	75	[51; 75]	
3	51	75	63	[51; 63]	
4	64	75	69	[64; 69]	
5	64	69	66	[64; 66]	
6	64	66	65	[64; 65]	
7	64	65	64	[64; 64]	
–	65	65	–	–	

Tabela 10.1. Przykład wyszukiwania liczby 65 w przedziale [1; 100]

Poszukiwanie elementu w uporządkowanej tablicy

Rozważmy teraz problem poszukiwania elementu w uporządkowanej tablicy liczb. W tym przypadku, kiedy zawężamy przedział poszukiwań, operujemy indeksami elementów tablicy, a nie wartościami elementów. Może się również zdarzyć, że szukanej liczby nie ma w tablicy. Oto specyfikacja problemu i zapis algorytmu w pseudokodzie:

Specyfikacja

Dane: `A[0..n–1]` – tablica n uporządkowanych niemalejąco liczb całkowitych,
x – liczba całkowita.

Wynik: wartość **prawda**, gdy liczba x znajduje się w tablicy, **fałsz** – w przeciwnym przypadku.

```
funkcja SzukajBin(A[],x)
    lewy ← 0
    prawy ← n – 1
    dopóki lewy < prawy wykonuj
        srodek ← (lewy + prawy) div 2
        jeśli A[srodek] < x to lewy ← srodek + 1
        w przeciwnym przypadku prawy ← srodek
    zwróć A[lewy] = x i zakończ
```

W zmiennych `lewy` i `prawy` przechowywane są indeksy początku i końca przeszukiwanego przedziału, a w zmiennej `srodek` – indeks środka tego przedziału. Przy wyznaczaniu środka przedziału nie spraw-

Warto w

Do określeni funkcji nie tr instrukcji wa
`jeśli A[lew`

Nie uzyskalibyśmy znaczącej poprawy, a w przypadku nieznalezienia liczby byłoby wykonywanych dwa razy więcej porównań. Wynikiem funkcji jest wartość logiczna – wynik porównania `A[lewy] = x`.

Kod źródłowy funkcji SzukajBin może wyglądać następująco:

Fragment kodu źródłowego programu wyszukującego element w tablicy – funkcja realizująca algorytm przeszukiwania binarnego

```
1.  bool SzukajBin(int A[], int x)
2.  {
3.      int lewy=0, prawy=N-1, srodek;
4.      while (lewy<prawy)
5.      {
6.          srodek=(lewy+prawy)/2;
7.          if (A[srodek]<x) lewy=srodek+1;
8.          else prawy=srodek;
9.      }
10.     return (A[lewy]==x);
11. }
```

Tabele 10.2 oraz 10.3 ilustrują działanie algorytmu poszukiwania odpowiednio liczb 3 i 5 w tablicy, w której na kolejnych pozycjach znajdują się liczby: 1, 4, 5, 6, 9. W pierwszym przykładzie uzyskamy odpowiedź, że liczby 3 nie ma w tablicy (pętla wykona się trzy razy), a w drugim – że liczba 5 jest w tablicy (pętla wykona się dwa razy).

Iteracja pętli	Wartość zmiennej lewy	Wartość zmiennej prawy	Wartość zmiennej srodek	Element A[srodek]	Odpowiedź na pytanie, czy A[srodek] < 3
1	0	4	2	5	Nie
2	0	2	1	4	Nie
3	0	1	0	1	Tak
–	1	1	–	–	–

Tabela 10.2. Przykład wyszukiwania liczby 3 w tablicy z liczbami: 1, 4, 5, 6, 9

A to ciekawe

20 pytań sztucznej inteligencji

Gra w 20 pytań, polegająca na zgadywaniu, o jakim obiekcie pomyślała dana osoba, ma wiele wersji. W 1988 r. Robin Burgener na podstawie tej popularnej zabawy stworzył grę 20Q, która do odgadywania wykorzystuje sztuczną inteligencję. W 20Q można zagrać online. Według właścicieli gry sztuczna inteligencja zgaduje pomyślany przez użytkownika

Iteracja pętli	Wartość zmiennej lewy	Wartość zmiennej prawy	Wartość zmiennej srodek	Element A[srodek]	Odpowiedź na pytanie, czy A[srodek] < 5
1	0	4	2	5	Nie
2	0	2	1	4	Tak
–	2	2	–	–	–

Tabela 10.3. Przykład wyszukiwania liczby 5 w tablicy z liczbami: 1, 4, 5, 6, 9

Parametrem funkcji SzukajBin jest uporządkowana niemalejąco tablica liczb. W związku z tym na potrzeby programu realizującego algorytm przeszukiwania binarnego zmodyfikujemy funkcję Losuj tak, aby losowała tak uporządkowaną tablicę liczb. Zakres losowania kolejnego elementu uzależnimy od wartości elementu poprzedniego – dodamy do niego wartość losową. Pierwszy element tablicy może być dowolną liczbą. Oto kod źródłowy zmodyfikowanej funkcji Losuj:

Fragment kodu źródłowego programu wyszukującego element w tablicy – funkcja losująca uporządkowaną niemalejąco tablicę liczb

```
1.  void Losuj(int A[])
2.  {
3.      A[0]=rand()%20;
4.      for (int i=1;i<N;i++) A[i]=A[i-1]+rand()%20;
5.  }
```

Kod źródłowy funkcji main programu poszukującego binarnie danego elementu w tablicy może być następujący:

Fragment kodu źródłowego programu wyszukującego binarnie dany element w tablicy – funkcja main

```
1.  int main()
2.  {
3.      int x;
4.      int A[N];
5.      srand(time(NULL));
6.      Losuj(A);
7.      Wypisz(A);
8.      cout<<"Podaj liczbe do wyszukania: ";
9.      cin>>x;
10.     if (SzukajBin(A,x))
11.         cout<<"Liczba jest w tablicy";
12.     else
13.         cout<<"Liczby nie ma w tablicy";
14.     return 0;
15. }
```

Ćwiczenie 3

Napisz program, który sprawdzi, czy w uporządkowanej tablicy *n* liczb

Tabela 10.4 przedstawia liczbę powtórzeń pętli algorytmu realizującego przeszukiwanie binarne w zależności od liczby elementów w zbiorze.

Liczba elementów w zbiorze (n)	Liczba powtórzeń pętli (k)
10	4
16	4
100	7
128	7
1 000	10
1 024	10
10 000	14
100 000	17
1 000 000	20
1 048 576	20

Tabela 10.4. Liczba iteracji pętli w algorytmie przeszukiwania binarnego w zależności od liczby elementów zbioru

Warto wiedzieć

lgorytm przeszukiwania inarnego opiera się metodzie „dziel zwyciężaj", czyli podziale roblemu na podproblemy go samego typu dla niejszej liczby danych. ięcej o metodzie „dziel zwyciężaj" powiemy tematcie 20.

Liczba potrzebnych powtórzeń pętli rośnie bardzo powoli mimo znacznego wzrostu liczby elementów w zbiorze. Dla liczby elementów rzędu miliona potrzebnych jest tylko 20 powtórzeń. Gdyby wykorzystać przeszukiwanie liniowe, potrzebne byłoby średnio pół miliona powtórzeń.

Na podstawie tabeli 10.4 można wyznaczyć wzór opisujący zależność pomiędzy liczbą powtórzeń k a liczbą danych n w algorytmie przeszukiwania binarnego. Dla liczby danych będących potęgą dwójki liczba powtórzeń jest równa wykładnikowi potęgi:

$$n = 2^k$$

Zatem:

$$k = \log_2 n$$

Dla wartości n, które nie są potęgami liczby 2, wartość logarytmu o podstawie 2 nie będzie liczbą całkowitą. Liczba iteracji pętli jest wówczas równa wartości zaokrąglonej w górę. Zaokrąglenie w górę, nazywane często sufitem, oznaczamy $\lceil\ \rceil$.

$$k = \lceil \log_2 n \rceil$$

Zapamiętaj

Algorytm przeszukiwania binarnego służy do znajdowania elementu w zbiorze uporządkowanym. Jego działanie polega na podzieleniu przeszukiwanego zbioru na połowy i kontynowaniu poszukiwań w części, w której może występować szukany element. Czynności powtarza się tak długo, aż pozostanie jeden element – wówczas sprawdza się, czy to szukany wynik.

10.4. Jednoczesne znajdowanie minimum i maksimum

Czasami zachodzi potrzeba znalezienia w zbiorze danych zarówno elementu najmniejszego, jak i największego. Dzieje się tak na przykład wtedy, gdy chcemy poznać **rozpiętość zbioru**, czyli różnicę pomiędzy elementem największym (maksimum) i elementem najmniejszym (minimum).

Rozpiętość zbioru

Oto specyfikacja problemu, którym się teraz zajmiemy:

Specyfikacja

Dane: `A[0..n-1]` – tablica n liczb całkowitych.

Wynik: `mini` – najmniejszy element w tablicy A,
`maks` – największy element w tablicy A.

Ponieważ nie wiemy, czy zbiór jest uporządkowany, narzucającym się rozwiązaniem jest zastosowanie algorytmu przeszukiwania liniowego: minimum i maksimum szukamy wtedy niezależnie. Zapis tego algorytmu w pseudokodzie może być następujący:

```
mini ← A[0]
dla i ← 1, 2, ..., n - 1 wykonuj
    jeśli A[i] < mini to mini ← A[i]
maks ← A[0]
dla i ← 1, 2, ..., n - 1 wykonuj
    jeśli A[i] > maks to maks ← A[i]
```

Obydwie instrukcje warunkowe można umieścić w jednej pętli, nie zmienia to jednak liczby porównań wykonywanych przez algorytm:

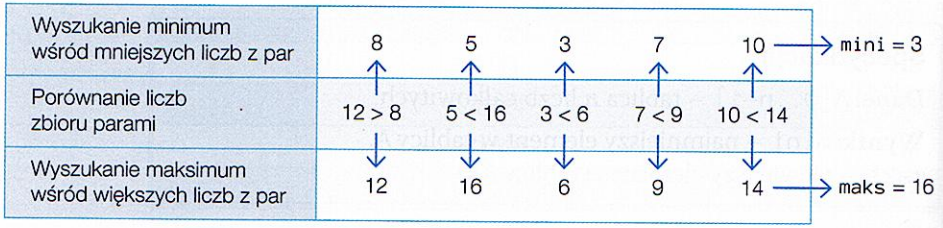
```
mini ← A[0]
maks ← A[0]
dla i ← 1, 2, ..., n - 1 wykonuj
    jeśli A[i] < mini to mini ← A[i]
    jeśli A[i] > maks to maks ← A[i]
```

Zauważ, że drugą instrukcję warunkową wystarczy wykonywać tylko wtedy, gdy nie była modyfikowana wartość zmiennej `mini`. Dlatego algorytm zmodyfikujemy następująco:

```
mini ← A[0]
maks ← A[0]
dla i ← 1, 2, ..., n - 1 wykonuj
    jeśli A[i] < mini to mini ← A[i]
    w przeciwnym przypadku
        jeśli A[i] > maks to maks ← A[i]
```

Jeśli zastosujemy ostatni algorytm, mamy szansę zmniejszyć liczbę porównań. W najgorszym przypadku zostanie wykonanych tyle samo porównań co w dwóch wcześniejszych algorytmach: $2 \cdot (n-1)$.

Zastanówmy się, jak znaleźć minimum i maksimum zbioru, wykonując mniej operacji porównania niezależnie od położenia szukanych wartości w tablicy. W tym celu można porównywać elementy parami: pierwszy z drugim, trzeci z czwartym itd. Następnie porównujemy mniejszy element z pary z aktualnym minimum, a większy – z aktualnym maksimum. Działanie algorytmu dla ciągu 12, 8, 5, 16, 3, 6, 7, 9, 10, 14 pokazuje rysunek 10.1.



Rys. 10.1. Wyznaczenie minimum i maksimum zbioru liczb: 12, 8, 5, 16, 3, 6, 7, 9, 10, 14

Warto wiedzieć

Algorytm jednoczesnego wyszukiwania minimum i maksimum w zbiorze opiera się na metodzie „dziel i zwyciężaj". Problem został podzielony na dwa mniejsze problemy: znalezienie minimum wśród mniejszych liczb z par oraz znalezienie maksimum wśród większych liczb z par.

Algorytm porównujący parami sąsiednie elementy wygodniej będzie zapisać, jeśli założymy, że liczba elementów tablicy jest parzysta.

Dla zbioru n-elementowego liczba porównań elementów parami jest równa $n/2$. Aby znaleźć minimum wśród mniejszych elementów z par, trzeba wykonać $n/2 - 1$ porównań, aby znaleźć maksimum wśród większych elementów z par – również $n/2 - 1$ porównań. Łącznie, kiedy szukamy i minimum, i maksimum, liczba porównań wynosi:

$$n/2 + (n/2 - 1) + (n/2 - 1) = 3n/2 - 2$$

Oto zmodyfikowana specyfikacja problemu:

Specyfikacja

Dane: `A[0..n-1]` – tablica n liczb całkowitych, n jest liczbą parzystą.

Wynik: `mini` – najmniejszy element w tablicy A,
`maks` – największy element w tablicy A.

Dobra rada

Jeśli chcesz zastosować podany algorytm jednoczesnego wyszukiwania minimum i maksimum w tablicy o nieparzystej liczbie elementów, możesz powiększyć rozmiar tablicy o jeden i skopiować na ostatnią pozycję wartość ostatniego elementu.

Zapis algorytmu w pseudokodzie może być następujący:

```
jeśli A[0] > A[1] to
    mini ← A[1]
    maks ← A[0]
w przeciwnym przypadku
    mini ← A[0]
    maks ← A[1]
dla i ← 2, 4, 6, ..., n - 2 wykonuj
    jeśli A[i] > A[i+1] to
        jeśli A[i+1] < mini to mini ← A[i+1]
        jeśli A[i] > maks to maks ← A[i]
    w przeciwnym przypadku
        jeśli A[i] < mini to mini ← A[i]
        jeśli A[i+1] > maks to maks ← A[i+1]
```

Instrukcja warunkowa przed pętlą wyznacza wartości początkowe zmiennych `mini` i `maks`, porównując dwa pierwsze elementy tablicy. Wykonywane jest w tym celu jedno porównanie. Pętla wykona się $(n - 2) / 2$ razy. Po porównaniu dwóch sąsiednich elementów mniejszy z nich jest porównywany z aktualnym minimum, a większy – z aktualnym maksimum. W jednym obrocie pętli wykonywane są zatem trzy porównania. Ponieważ jedno porównanie jest wykonywane przed pętlą, w sumie powyższy algorytm wykona $1 + 3 \cdot (n - 2)/2 = 3 \cdot n/2 - 2$ porównań. Oto kod źródłowy funkcji realizującej ten algorytm:

Fragment kodu źródłowego programu wyszukującego jednocześnie minimum i maksimum w tablicy – funkcja `MiniMaks`

```
1.  void MiniMaks(int A[], int &mini, int &maks)
2.  {
3.      if (A[0]>A[1])
4.      {
5.          mini=A[1]; maks=A[0];
6.      }
7.      else
8.      {
9.          mini=A[0]; maks=A[1];
10.     }
11.     for (int i=2;i<N-1;i+=2)
12.         if (A[i]>A[i+1])
13.         {
14.             if (A[i+1]<mini) mini=A[i+1];
15.             if (A[i]>maks) maks=A[i];
16.         }
17.         else
18.         {
19.             if (A[i]<mini) mini=A[i];
20.             if (A[i+1]>maks) maks=A[i+1];
21.         }
22. }
```

Funkcja `MiniMaks` ma zwrócić dwie liczby całkowite: minimum i maksimum. Wartości są zwracane za pośrednictwem parametrów, dlatego funkcja jest typu `void`. Domyślnie parametry **typu prostego przekazywane są przez wartość**, czyli operacje wewnątrz funkcji wykonywane są na kopii parametru, a zmiany nie są zachowywane po zakończeniu działania funkcji. Żeby zachować zmiany w parametrze będącym zmienną typu prostego, działania powinny być wykonywane bezpośrednio na zmiennej, czyli parametrem aktualnym musi być adres zmiennej. Sygnalizuje to znak `&` (linia 1) umieszczony przed nazwą parametru formalnego. Taki sposób przekazania parametru w języku C++ nazywamy **przekazaniem parametru przez referencję**.

Typ prosty, s. 142
Przekazanie parametru przez wartość, s. 108

Przekazanie parametru przez referencję

Ćwiczenie 4

Napisz program wyszukujący jednocześnie największy i najmniejszy element tablicy. Wykorzystaj przedstawioną powyżej funkcję `MiniMaks`.

Podsumowanie

- Algorytm przeszukiwania liniowego wykorzystuje się do poszukiwania elementu w zbiorze nieuporządkowanym.
- W algorytmie przeszukiwania liniowego sprawdzamy, czy kolejne elementy zbioru są równe szukanemu – czynności powtarzamy do momentu, gdy znajdziemy element lub przejrzymy wszystkie elementy zbioru.
- Algorytm przeszukiwania binarnego stosuje się do poszukiwania elementu w zbiorze uporządkowanym.
- Kiedy stosujemy algorytm przeszukiwania binarnego, w kolejnych krokach dzielimy zbiór na pół i odrzucamy część, w której na pewno nie ma szukanego elementu. Czynności te powtarzamy, dopóki nie pozostanie jeden element.
- Żeby znaleźć równocześnie minimum i maksimum zbioru, można porównywać elementy zbioru parami: pierwszy z drugim, trzeci z czwartym itd. Po każdym porównaniu elementów w parze porównujemy mniejszy element z pary z aktualnym minimum zbioru, a większy z aktualnym maksimum.
- W przypadku, gdy parametrem funkcji jest tablica, do funkcji przekazywany jest adres tablicy, więc zmiany dokonane na takim parametrze są zachowywane po zakończeniu działania funkcji. Taki sposób przekazania parametru nazywamy przekazaniem przez wskaźnik.
- Jeśli chcemy zachować zmiany dokonane w funkcji na parametrze typu prostego, to powinniśmy w nagłówku funkcji przed nazwą parametru formalnego wstawić znak &. Wtedy do funkcji zostanie przekazany adres zmiennej będącej parametrem aktualnym. Taki sposób przekazania parametru nazywamy przekazaniem przez referencję.

Zadania

* **1** Napisz program, który sprawdzi, czy dana liczba znajduje się w tablicy n liczb całkowitych. Wykorzystaj funkcję realizującą algorytm przeszukiwania liniowego. Wynikiem funkcji powinna być pozycja szukanego elementu lub wartość n, jeśli elementu nie ma w tablicy.

* **2** Napisz program, który sprawdzi, czy dana liczba znajduje się w uporządkowanej tablicy n liczb całkowitych. Wykorzystaj funkcję realizującą algorytm przeszukiwania binarnego. Wynikiem funkcji powinna być pozycja szukanego elementu lub wartość n, jeśli szukanego elementu nie ma w tablicy.

* **3** Napisz program wyznaczający rozpiętość zbioru danych znajdujących się w tablicy n liczb całkowitych.

** **4** Napisz program, który w tablicy n liczb całkowitych z zakresu od 0 do 999 999 999 znajdzie liczbę o największej sumie cyfr.

** **5** Napisz program, który w tablicy n liczb całkowitych znajdzie pierwszy element parzysty. Przyjmij, że w tablicy najpierw występują liczby nieparzyste, potem parzyste oraz że tablica zawiera co najmniej jedną liczbę nieparzystą i jedną parzystą. Zastosuj algorytm przeszukiwania binarnego.

** **6** Przeanalizuj podany pseudokod i zastanów się, czy dla każdych danych realizuje on algorytm przeszukiwania binarnego. Odpowiedź uzasadnij.

```
funkcja S(A[],x)
    lewy ← 0
    prawy ← n - 1
    dopóki lewy < prawy wykonuj
        srodek ← (lewy + prawy) div 2
        jeśli A[srodek] > x to prawy ← srodek - 1
        w przeciwnym przypadku lewy ← srodek
    zwróć A[lewy] = x i zakończ
```

** **7** Napisz funkcję, która dla danej liczby całkowitej dodatniej znajdzie jej pierwiastek kwadratowy dyskretny. Zastosuj algorytm przeszukiwania binarnego.
Uwaga: Pierwiastek dyskretny z danej liczby to największa liczba całkowita mniejsza lub równa pierwiastkowi z tej liczby.

*** **8** Napisz funkcję, której wynikiem będzie liczba elementów z przedziału $[a; b]$ (a, b – liczby całkowite, $a < b$), znajdujących się w uporządkowanej niemalejąco tablicy n liczb całkowitych. Zastosuj algorytm przeszukiwania binarnego.

*** **9** Zapisz w wybranej przez siebie notacji algorytm zgadywania liczby w grze w 20 pytań przy założeniu, że jedna odpowiedź może być błędna. Jaka jest minimalna liczba pytań potrzebnych do odgadnięcia liczby?