1. Algorytmy na tekstach

Błędy językowe, literówki i inne pomyłki w tekście zwykle świadczą o niewiedzy lub nieuwadze piszącego. Czasami jednak mogą mieć naprawdę poważne konsekwencje. Na przykład skład leku i jego dawkowanie powinny być zawsze szczegółowo i precyzyjnie opisane w ulotce. W takim wypadku nawet przypadkowa zmiana znaków w tekście może być kluczowa. Programy przetwarzające tekst oferują wiele narzędzi, m.in. wyszukiwanie, automatyczne podkreślanie powtarzających się wyrazów czy możliwość wielokrotnej zmiany podanej frazy na inną. W tym temacie nauczysz się przetwarzać teksty z wykorzystaniem programów napisanych w języku C++.

Cele lekcji

- Zrozumiesz, jak komputer zapisuje informacje tekstowe.
- Poznasz i zastosujesz algorytmy przetwarzania tekstów: porównywania tekstów oraz wyszukiwania wzorca.
- Nauczysz się używać zmiennych typu **`char`** i typu **`string`** w języku C++.
- Dowiesz się, jak przetwarzać teksty w języku C++.
- Poznasz funkcje z biblioteki `string`: `length`, **`find`** i `rfind`.

1.1. Litery i inne znaki jako liczby

Emoji

W serwisach społecznościowych często używa się piktogramów zwanych **emoji**, czyli symboli rysunkowych uzupełniających tekst o graficzne wyrażenie emocji lub przedstawienie jakiegoś obiektu – przedmiotu, miejsca, rośliny itp.

Kod liczbowy znaku

Tablica Unicode

Jeśli umieścisz na portalu społecznościowym emoji 🌻, to widzisz je dlatego, że piktogram ten znajduje się w zbiorze znaków czcionki na twoim urządzeniu. Emoji 🌻 jest w takim zbiorze na pozycji o numerze 127 803. Mówimy, że jest to **kod liczbowy znaku** (w tym przypadku piktogramu). Kody wszystkich piktogramów, a także wszelkich innych znaków (liter oraz symboli m.in. matematycznych i muzycznych) znajdują się w specjalnej **tablicy znaków Unicode**. W roku 2020 liczba znaków zakodowanych w tablicy Unicode przekraczała 280 000.

Warto wiedzieć

Emoji wywodzą się z japońskich telefonów komórkowych z końca lat 90. XX w. Nazwa pochodzi od japońskich słów: 絵 (obraz) i 文字 (litera).

Ćwiczenie 1

a. Uruchom edytor tekstu i wybierz czcionkę Segoe UI Emoji lub Segoe UI Symbol. Następnie, przytrzymując wciśnięty lewy klawisz **Alt**, wybierz w części numerycznej klawiatury kolejno cyfry: **128190**. Jaki symbol kryje się pod tym kodem?

b. Korzystając z dostępnych serwisów internetowych, sprawdź, jaki kod ma emoji z piłką nożną ⚽.

Tabela 1.1 przedstawia fragment tablicy Unicode dla emoji o kodach od 127 800 do 127 819.

	0	1	2	3	4	5	6	7	8	9
127 80_	🌸	🌹	🌺	🌼	🌻	🌷	🌿	🍁	🍀	🍁
127 81_	🍃	🍂	🍄	🍎	🍐	🍇	🥥	🥝	🍉	🥭

Tabela 1.1. Fragment tablicy znaków Unicode zawierający wybrane piktogramy

Dobra rada

Większość laptopów ma klawisze, które symulują numeryczną część tradycyjnej klawiatury QWERTY. Zazwyczaj są to klawisze **7**, **8**, **9**, **U**, **I**, **O**, **J**, **K**, **L** oraz **M**. Klawisz **Num Lock** znajdziesz zwykle w prawej górnej części klawiatury laptopa.

Podstawowe znaki, takie jak litery alfabetu łacińskiego, cyfry, znaki interpunkcyjne i symbole (np. @ i ^), które możemy wprowadzić do komputera bezpośrednio z klawiatury, znajdziemy w początkowej części tablicy Unicode (na pozycjach od 0 do 127). Jest ona nazywana podstawowym zestawem znaków ASCII (skrót od ang. *American Standard Code for Information Interchange*), a w skrócie – **tablicą ASCII**. System kodowania znaków ASCII opracowano w latach 60. XX w. na potrzeby telekomunikacji. Powstałe później systemy kodowania znaków są w pełni kompatybilne z ASCII.

Tablica ASCII

Typ znakowy w języku C++

W języku C++ możemy deklarować zmienne przechowujące znaki. Służy do tego **typ `char`**, nazywany **znakowym typem danych** (ang. *character data type*). Zmienna tego typu to liczba z zakresu od 0 do 127, która jest kodem liczbowym odpowiedniego znaku w tablicy ASCII. Fragment tej tablicy przedstawia tabela 1.2.

Typ `char` (typ znakowy)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Tabela 1.2. Fragment tablicy ASCII

Warto wiedzieć

Kod ASCII jest czasami wykorzystywany w filmach fabularnych. Na przykład w filmie *Marsjanin* główny bohater otrzymał z NASA zakodowaną wiadomość: 48 4F 57 41 4C 49 56 45. Korzystając z tablicy kodów ASCII, odczytał ją jako `HOW ALIVE`, czyli pytanie o to, jak udało mu się przeżyć na Marsie.

Zazwyczaj tablicę ASCII przedstawia się tak, że znaki w niej zapisane są pogrupowane po 16. W opisie kolumn pojawiają się kolejne cyfry systemu szesnastkowego (od 0 do F). Aby odczytać kod liczbowy, czyli **kod ASCII** znaku, parę liczb, czyli numer wiersza i numer kolumny, traktuje się jako kod znaku zapisany w systemie szesnastkowym. Na przykład znak @ ma kod 40_{16}, czyli 64 dziesiętnie, a znak [ma kod $5B_{16}$, czyli 91 dziesiętnie.

Kod ASCII

Napiszemy teraz program, który wyświetli fragment tablicy ASCII pokazany w tabeli 1.2 na s. 11. Kody widocznych tam znaków mieszczą się między 48 a 126. Program powinien wyświetlić znaki z odstępami w wierszach po 16 znaków (nie licząc spacji).

Oto kod źródłowy programu *Znaki ASCII*, który wykonuje opisane zadanie:

Kod źródłowy programu *Znaki ASCII*

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.

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  int main()
6.  {
7.      char znak;
8.
9.      for (int i=48; i<=126; i++)
10.     {
11.         znak = i;
12.         cout << znak << " ";
13.         if (i%16 == 15)
14.             cout << endl;
15.     }
16.     cout << endl;
17.     return 0;
18. }
```

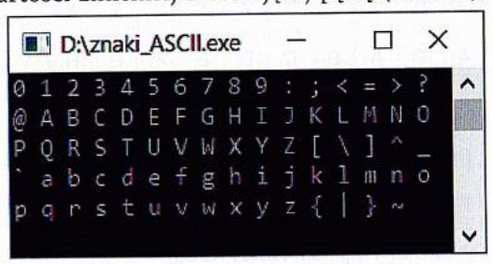
W linii 7 znajduje się deklaracja zmiennej znak typu **char**. W liniach 9–15 wykonywana jest **pętla for**, która realizuje wyświetlanie znaków w następujący sposób: zmiennej znak przypisywana jest liczba (kod ASCII znaku) równa wartości zmiennej i sterującej pętlą (linia 11), następnie wyświetlany jest znak, a po nim pojedyncza spacja, zapisana w cudzysłowie (linia 12). Zapisy w liniach 13 i 14 służą temu, aby znaki wyświetlić w wierszach po 16 znaków.

Pętla **for**, s. 247

Warto wiedzieć

Deklaracja zmiennej typu **char** skutkuje zarezerwowaniem w pamięci jednego bajta (czyli 8 bitów), w którym znajdzie się binarny zapis liczby będącej kodem ASCII danego znaku. Oto zapis binarny kodu litery A (wynoszącego 65):

0	1	0	0	0	0	0	1



Rys. 1.1. Wywołanie programu *znaki_ASCII.exe*

Na rysunku 1.1 przedstawiono wywołanie programu *Znaki ASCII*. Zauważ, że układ tabeli kodów ASCII jest starannie przemyślany. Na przykład pary wielkich i małych liter („A" i „a", „B" i „b" itd.) znajdują się w tych samych kolumnach i są oddalone o dwa wiersze. Dlatego łatwo z kodu małej litery przejść do kodu odpowiadającej jej wielkiej litery i na odwrót. Różnica w kodzie liczbowym odpowiadająca odległości dwóch wierszy wynosi 32.

Ćwiczenie 2

a. Zapisz kod źródłowy programu w pliku *znaki_ASCII.cpp*. Skompiluj kod i sprawdź działanie programu.

b. Zmodyfikuj powyższy kod źródłowy tak, aby program wyświetlił znaki, które w tablicy ASCII mają kody liczbowe między 32 a 47.

Przypisanie kodu znaku z wykorzystaniem apostrofów

Aby przypisać wartość zmiennej typu **char**, nie trzeba znać liczb przyporządkowanych znakom w tablicy ASCII. Zmiennej można nadać wartość liczbową niejawnie, podając wybrany znak między apostrofami, np.:

```
char znak = '~';
```

Dobra rada

Apostrof znajdziesz na tym samym klawiszu co znak cudzysłowu.

Gdybyśmy od razu wiedzieli, że program ma wyświetlić wszystkie znaki znajdujące się w tablicy ASCII między cyfrą 0 a znakiem ~ (patrz rys. 1.1), to kod źródłowy mógłby wyglądać następująco:

Kod źródłowy programu *Znaki ASCII 2*

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  int main()
6.  {
7.      char znak = '0';
8.
9.      while (znak <= '~')
10.     {
11.         cout << znak << " ";
12.         if (znak%16 == 15)
13.             cout << endl;
14.         znak = znak + 1;
15.     }
16.     cout << endl;
17.     return 0;
18. }
```

Warto wiedzieć

Kod ASCII był pierwotnie przeznaczony na potrzeby usługi telegraficznej zwanej teleksem. Nadawca zapisywał tekst na klawiaturze urządzenia zwanego dalekopisem, a odbiorca otrzymywał wydruk. Dlatego tablica ASCII na pozycjach od 0 do 31 i na pozycji 127 zawiera kody przeznaczone do sterowania drukarką.

W wierszu 7 deklarujemy zmienną znak i przypisujemy jej cyfrę 0. Wygodnie będzie zastosować **pętlę while**, która będzie wykonywana, dopóki liczba określająca kod ASCII danego znaku będzie mniejsza od kodu ASCII znaku ~ lub jemu równa.

W pętli wypisywane są znak, którego kod ASCII jest przechowywany w zmiennej znak, oraz spacja (wiersz 11). Zapisy w wierszach 12–13 odpowiadają za wypisywanie znaków po szesnaście, a w linii 14 zmiennej znak przypisywany jest kod ASCII kolejnego znaku. Zapisy w wierszach 9 i 14 są poprawne dlatego, że w pamięci komputera znaki są zapisane jako kolejne liczby całkowite, a więc są w ustalonym porządku.

Pętla **while**, s. 248

Ćwiczenie 3

a. Zapisz kod źródłowy programu *Znaki ASCII 2* pod nazwą *znaki_ASCII_2.cpp*. Skompiluj kod i sprawdź działanie programu.
b. Napisz program, który wyświetli litery od A do Z w dwóch wierszach w następujący sposób:

```
ABCDEFGHIJKLM
ZYXWVUTSRQPON
```

1.2. Szukamy literówek. Łańcuchy znaków w języku C++

Zapobiegać powstawaniu literówek można na różne sposoby. Na przykład w edytorze tekstu słowa zapisywane przez użytkownika zwykle są porównywane ze słowami znajdującymi się w słowniku. Innym sposobem jest wymaganie od użytkownika dwukrotnego wpisania tekstu, np. PIN-u, adresu e-mail lub hasła.

Warto wiedzieć

Edytory tekstu, a także gry słowne (np. Scrabble, Literaki czy Słowotok) zazwyczaj mają swoje oficjalne słowniki, w których sprawdzane są podane wyrazy.

Poprawność adresu e-mail – porównanie dwóch adresów

Podczas zakładania konta w serwisie internetowym często wymaga się od nas, abyśmy dwukrotnie podali adres poczty elektronicznej. Napiszemy program, który będzie symulował taką rejestrację w serwisie internetowym.

Oto specyfikacja problemu:

Specyfikacja

Dane: cztery napisy oznaczające kolejno: imię, nazwisko oraz adres e-mail (podany dwukrotnie).

Wynik: komunikat „Hasło do serwisu wysłaliśmy na adres: <adres_e-mail>", jeśli wpisane adresy są identyczne, w przeciwnym wypadku komunikat „Podane adresy są różne!".

Sprawdzenie, czy dwa adresy są identyczne, sprowadza się do porównania wartości dwóch zmiennych, w których są one przechowywane.

Typ `string` — W poprzednich tematach stosowaliśmy już zmienne **typu `string`**, czyli zmienne typu tekstowego, w których można zapamiętywać napisy składające się ze znaków ASCII.

Biblioteka `string` — Aby móc deklarować zmienne typu **`string`**, należy użyć **biblioteki** `string`, czyli wpisać na początku kodu źródłowego programu dyrektywę: `#include <string>`. Zmienne typu tekstowego nazywać będziemy **napisami, łańcuchami znaków** (ang. *strings of characters*) lub po prostu **łańcuchami**.

Napis (łańcuch znaków, łańcuch)

Podobnie jak w przypadku wypisywania napisów na ekranie, aby przypisać zmiennej typu **`string`** wartość (w postaci napisu), umieszcza się ją w cudzysłowie.

Oto kod źródłowy funkcji `main` programu *Porównanie adresów*:

```
1.  int main()
2.  {
3.      const string INF_OK =
4.          "Haslo do serwisu wyslalismy na adres: ";
5.      const string INF_BLAD =
6.          "Podane adresy e-mail sa rozne!";
7.
8.      string imie, nazwisko, adres_1, adres_2;
9.
10.     cout << "Podaj imie: "; cin >> imie;
11.     cout << "Podaj nazwisko: "; cin >> nazwisko;
12.
13.     cout << "Podaj adres e-mail: "; cin >> adres_1;
14.     cout << "Powtorz adres e-mail: "; cin >> adres_2;
15.     cout << endl;
16.
17.     if (adres_1 == adres_2)
18.         cout << INF_OK + adres_1 << endl;
19.     else
20.         cout << INF_BLAD << endl;
21.
22.     return 0;
23. }
```

Fragment kodu źródłowego programu *Porównanie adresów*

Dobra rada

Aby kod był czytelniejszy, długie napisy przypisywane stałym umieszczaj w tekście programu od nowej linii. Podczas kompilacji znak nowej linii (podobnie jak spacja) po operatorze przypisania jest ignorowany.

Warto wiedzieć

Wszelkiego rodzaju dane zapisane bezpośrednio w kodzie programu nazywa się literałami. Liczba 23 w instrukcji `int suma = 23;` to literał całkowity, a napis ABC w instrukcji `string napis = "ABC";` to literał napisowy.

Ponieważ oba komunikaty ustalone w specyfikacji są dla użytkownika niezmienne, możemy je zdefiniować jako **stałe** typu **`string`** (linie 3–6). Zauważ, że słowo kluczowe **`const`** znajduje się przed nazwą i typem zapamiętywanej wartości. Kompilator potraktuje ją jako wartość tylko do odczytu (ang. *read-only*) i nie pozwoli na jej zmianę. Dobrą praktyką jest zapisywanie nazw stałych wielkimi literami. Dzięki temu nie będą się one mylić z nazwami zmiennych.

Stała

Dobra rada

Pamiętaj, że w języku C++ rozróżniana jest wielkość liter. Nazwy zadeklarowanej zmiennej używaj dokładnie tak samo.

W wierszu 8 deklarujemy zmienne typu **`string`**, w których będą zapamiętane dane wpisane z klawiatury przez użytkownika (pobierane w liniach 10–14). Zauważ, że napisy jako wartości zmiennych i stałych typu **`string`** deklarujemy pomiędzy parą znaków **" "**.

W **warunku logicznym** (wiersz 17) sprawdza się, czy zachodzi równość między wartościami dwóch zmiennych typu **`string`**. Jeśli tak, wypisywany jest komunikat o poprawnej rejestracji (INF_OK) uzupełniony o podany adres e-mail (linia 18). Do złączenia (scalenia) napisów wykorzystujemy **operator** +. Operację tę nazywamy **złączeniem** lub **konkatenacją** (ang. *concatenation*). Jeśli podane napisy są różne, program wypisze na ekranie komunikat zapisany w stałej INF_BLAD (linia 20).

Logika matematyczna i operatory logiczne, s. 242–244

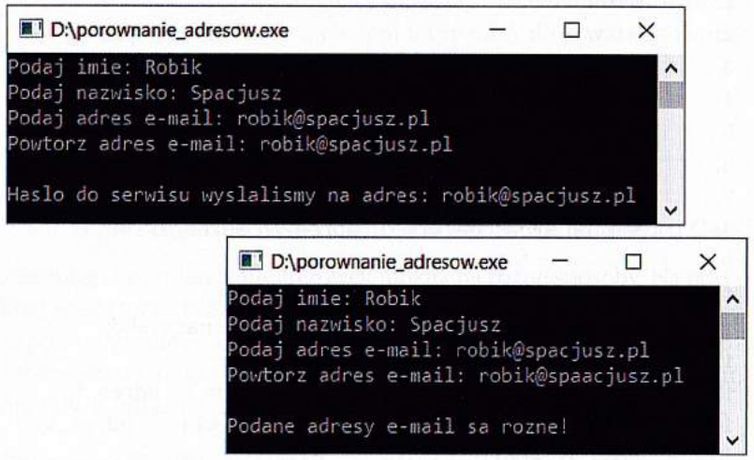
Operator +

Operacja złączenia (konkatenacja)

Warto wiedzieć

W teorii literatury konkatenacją nazywa się konstrukcję, która polega na powieleniu tego samego słowa lub wyrażenia (niekoniecznie w tej samej formie fleksyjnej) na końcu jednego wersu i na początku następnego. Przykład zastosowania konkatenacji można znaleźć w wierszu Juliana Tuwima *Rzepka*.

Rysunek 1.2 przedstawia przykładowe wywołania programu dla dwóch sytuacji: poprawnego i niepoprawnego podania adresów e-mail.



Rys. 1.2. Wywołania programu *porownanie_adresow.exe*

Ćwiczenie 4

Zapisz kod źródłowy przedstawionego wcześniej programu pod nazwą *porownanie_adresow.cpp*. Skompiluj kod i sprawdź działanie programu dla takich samych oraz różnych adresów e-mail.

Poprawność adresu e-mail – sprawdzenie struktury adresu

Poprzedni program można rozbudować o sprawdzanie, czy napis podany przez użytkownika w polu „Podaj adres e-mail:" spełnia warunki poprawnego adresu poczty elektronicznej. Napiszemy program *Sprawdzenie adresu*, który będzie wykonywał to zadanie.

Dla uproszczenia przyjmiemy, że za poprawny adres e-mail będzie uznany napis, w którym znajdują się: dokładnie jeden znak @, który jest co najmniej trzecim znakiem napisu, oraz kropka, która jest czwartym lub trzecim znakiem od końca napisu i nie występuje bezpośrednio po znaku @.

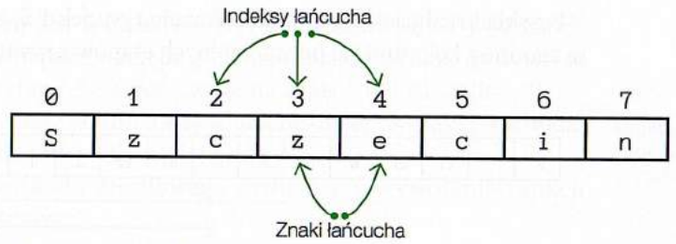
Warto wiedzieć

Domeny najwyższego rzędu zazwyczaj są dwu- lub trzyliterowe. Należą do nich m.in. domeny .pl, .us, .org, .com, .gov.

Przykładami poprawnych adresów będą: nowak@abcd.pl oraz nk@ab.com. Niepoprawne adresy to np.: n@abc.pl, nk@w@.pl, n.@pl, a także nk@.com. W poprzednim programie porównywaliśmy całe napisy – tym razem będziemy porównywać pojedyncze znaki.

Na łańcuch znaków można patrzeć jak na ciąg komórek w pamięci komputera. Pierwszy znak jest przechowywany w komórce o indeksie 0, drugi – w komórce o indeksie 1, a ostatni – w komórce o numerze $n - 1$, gdzie n to liczba znaków łańcucha (rys. 1.3). Zauważ, że w **tablicach** stosuje się podobny sposób indeksacji.

Tablice w języku C++, s. 246

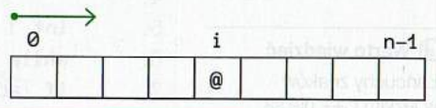


Rys. 1.3. Interpretacja graficzna łańcucha znaków

Poprawność adresu e-mail możemy sprawdzić w czterech etapach. Opisuje je schemat z rysunku 1.4.

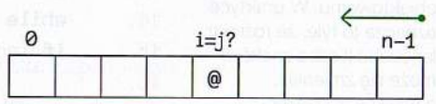
Etap 1

Szukamy w napisie (zaczynając od lewej strony) znaku @. Jeśli go nie znajdziemy albo będzie on pierwszym lub drugim znakiem napisu, sprawdzanie kończy się niepowodzeniem. W przeciwnym razie zapamiętujemy w zmiennej i pozycję znaku @ w napisie. Przechodzimy do etapu 2.



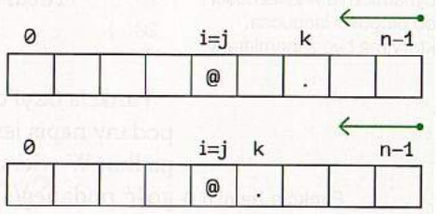
Etap 2

Sprawdzamy, czy wewnątrz napisu nie ma innego znaku @. Szukamy pierwszego wystąpienia znaku @, począwszy od prawego końca napisu. Zapamiętujemy jego pozycję w zmiennej j i sprawdzamy, czy i jest równe j. Jeśli wartości są różne, sprawdzanie kończy się niepowodzeniem. W przeciwnym razie przechodzimy do etapu 3.



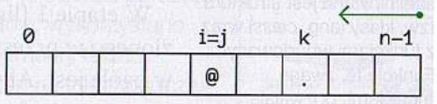
Etap 3

Sprawdzamy, czy znak kropki występuje w napisie na pozycji trzeciej lub czwartej od końca. Jeśli nie, sprawdzanie kończy się niepowodzeniem. Jeśli tak, zapamiętujemy pozycję kropki, np. w zmiennej k, i przechodzimy do etapu 4.



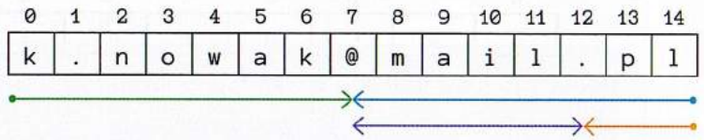
Etap 4

Sprawdzamy, czy znak @ znajduje się po lewej stronie znaku kropki w odstępie co najmniej jednego znaku. Jeśli nie, sprawdzanie kończy się niepowodzeniem. Jeśli tak, podany adres uznajemy za poprawny.



Rys. 1.4. Etapy sprawdzania poprawności podanego adresu e-mail

Przykład realizacji algorytmu pokazuje rysunek 1.5. Kolory strzałek są zgodne z kolorami tła poszczególnych etapów z rysunku 1.4 ze s. 17.



Rys. 1.5. Sprawdzanie poprawności adresu

Algorytm sprawdzający poprawność adresu e-mail wygodnie będzie zapisać w kodzie źródłowym w formie funkcji:

Kod źródłowy funkcji CzyPoprawnyAdres w programie *Sprawdzenie adresu*

```
1.  bool CzyPoprawnyAdres(string adres)
2.  {
3.      int dl = adres.length();
4.
5.      int i = 0; // etap 1
6.      while (adres[i]!='@' && i<dl-1) i++;
7.      if (adres[i]!='@' || i<2) return false;
8.
9.      int j = dl-1; // etap 2
10.     while (adres[j]!='@') j--;
11.     if (i!=j) return false;
12.
13.     int k = dl-1; // etap 3
14.     while (adres[k]!='.' && k>0) k--;
15.     if (adres[k]!='.'||!(k==dl-3||k==dl-4)) return false;
16.
17.     if (k-i<=1) return false; // etap 4
18.
19.     return true;
20. }
```

Warto wiedzieć

Łańcuchy znaków w języku C++ nie są zwykłymi tablicami znaków, czyli tablicami zmiennych typu **char**. Projektując bibliotekę `string`, zastosowano podejście właściwe tzw. programowaniu obiektowemu. W praktyce oznacza to tyle, że rozmiar łańcucha (liczba znaków) może się zmieniać, a miejsce w pamięci komputera dla zmiennej jest przydzielane dynamicznie w zależności od długości łańcucha, który ma być zapamiętany.

Funkcja CzyPoprawnyAdres zwraca wartość logiczną – **true**, gdy podany napis jest adresem e-mail, lub **false** – w przeciwnym przypadku. W wierszu 3 deklarujemy zmienną `dl`, która przechowa długość podanego napisu. Wykorzystaliśmy do tego **funkcję** `length` zdefiniowaną w bibliotece `string`. Funkcja ta zwraca długość napisu. Zapisujemy ją po nazwie zmiennej i kropce. Kolejne etapy sprawdzania poprawności adresu zaznaczono w komentarzach.

Funkcja `length`

Warto wiedzieć

W bibliotece `string` zdefiniowana jest struktura tzw. klasy (ang. *class*) wraz z funkcjami wbudowanymi. Funkcje te, zwane metodami, pozwalają wykonywać operacje na obiektach danej klasy.

W etapie 1 (linie 5–7) zmienna `i` określa pozycję pierwszego znalezionego w przeszukiwaniu od lewej znaku @, o ile taki znak w napisie w ogóle jest. Aby uzyskać dostęp do znaku w napisie, stosuje się zapis z parą nawiasów kwadratowych. Używaliśmy go już podczas korzystania ze struktury tablicy.

Etap 2 zapisany jest w liniach 9–11. Jeżeli `i` różni się od `j` (warunek z linii 11), to w napisie występują co najmniej dwa znaki @. Wiersze 13–15 dotyczą etapu 3. Zwróć uwagę na zapis `!(k==dl-3||k==dl-4)` w linii 15 – oznacza on **formułę logiczną** „Nieprawda, że zachodzi jeden z przypadków `k==dl-3` lub `k==dl-4`".

Logika matematyczna i operatory logiczne, s. 242–244

Oto fragment kodu źródłowego zawierający wywołanie funkcji CzyPoprawnyAdres:

Fragment kodu źródłowego funkcji `main` programu *Sprawdzenie adresu*

```
1.  if (CzyPoprawnyAdres(adres_1))
2.  {
3.      cout << "Powtorz adres e-mail: "; cin >> adres_2;
4.      cout << endl;
5.      if (adres_1 == adres_2)
6.          cout << INF_OK + adres_1 << endl;
7.      else
8.          cout << INF_ROZNE_ADRESY << endl;
9.  }
10. else
11.     cout << endl << INF_BLAD_ADRESU << endl;
```

Zwróć uwagę, że funkcja CzyPoprawnyAdres jest typu logicznego, a jej wywołanie znajduje się w miejscu wyrażenia logicznego instrukcji warunkowej (linia 1). W ten sposób funkcję typu logicznego można wykorzystać jako warunek logiczny, np. w instrukcji warunkowej lub pętli.

Warto wiedzieć

Zapis w linii 1 może być również następujący:

```
if (CzyPoprawnyAdres
   (adres_1) == 1)
```

Ćwiczenie 5

a. Zapisz specyfikację do programu *Sprawdzenie adresu*. Zaproponuj brzmienie wykorzystywanych w nim komunikatów.
b. Zapisz pełny kod źródłowy programu *Sprawdzenie adresu*. Skompiluj kod i uruchom program.

A to ciekawe

Jak odróżnić człowieka od maszyny?

W wielu serwisach internetowych stosuje się rozwiązanie nazywane CAPTCHA, którego zadaniem jest upewnienie się, że formularz uzupełnia człowiek, a nie spamujący robot. Test polega zazwyczaj na przepisaniu słowa z obrazka. Fakt, że czynność tę wykonują regularnie miliony internautów, wykorzystano w projekcie reCAPTCHA. Użytkownikom wyświetla się fragmenty tekstu nieczytelne dla oprogramowania OCR (zamieniającego skan na tekst) podczas digitalizacji zasobów na użytek bibliotek cyfrowych. Jeśli wystarczająco dużo osób tak samo odczyta dane słowo, można je uznać za poprawnie rozpoznane.

Poprawność adresu e-mail – sprawdzenie struktury adresu z wykorzystaniem funkcji z biblioteki `string`

Obiekty, klasy i metody w języku C++, s. 251

Biblioteka `string` poza **funkcją (metodą)** `length` oferuje kilkanaście innych funkcji, nazywanych również metodami, które mogą usprawnić pisanie kodu źródłowego oraz znacznie go skrócić.

Funkcja find
Funkcja rfind

Zapiszemy teraz poznany wcześniej algorytm z wykorzystaniem funkcji **find** oraz `rfind`. **Funkcja find** zwraca indeks pierwszego wystąpienia poszukiwanej frazy. **Funkcja** `rfind` zwraca indeks ostatniego wystąpienia poszukiwanej frazy, czyli szuka frazy, zaczynając od końca (od prawej strony) analizowanego łańcucha znaków.

Oto zapis funkcji `CzyPoprawnyAdres` z użyciem powyższych funkcji:

Kod źródłowy funkcji CzyPoprawnyAdres wykorzystujący funkcje z biblioteki `string` w programie *Sprawdzenie adresu 2*

```
1.  bool CzyPoprawnyAdres(string adres)
2.  {
3.      int dl = adres.length();
4.
5.      if (adres.find("@")<2 || adres.find("@")>dl)
6.          return false; // etap 1
7.      int i = adres.find("@");
8.
9.      int j = adres.rfind("@"); // etap 2
10.     if (i!=j) return false;
11.
12.     if (adres.rfind(".")>dl) return false; // etap 3
13.     int k = adres.rfind(".");
14.     if (!(k==dl-3 || k==dl-4)) return false;
15.
16.     if (k-i<=1) return false; // etap 4
17.
18.     return true;
19. }
```

Warto wiedzieć

Zapis nazwy funkcji po kropce jest właściwy dla tzw. programowania obiektowego.

Funkcję **find** zastosowaliśmy w linii 7 do odszukania indeksu pierwszego wystąpienia znaku @. Warunki `adres.find("@")>dl` z linii 5 oraz `adres.rfind(".")>dl` z linii 12 odnoszą się do sytuacji, w której funkcje **find** i `rfind` nie znajdą poszukiwanych znaków. Wówczas funkcje te zwracają pewną ustaloną wartość, która jest większa niż długość napisu. W obu przypadkach użyto zapisu z nazwą funkcji po kropce.

Zapamiętaj

W języku C++ rozróżniamy zmienne znakowe typu **char** i zmienne łańcuchowe typu **string**. Pierwsze służą do zapamiętania pojedynczego znaku (np. litery, cyfry), a drugie pozwalają na przechowanie całego łańcucha znaków.

1.3. Lista zakupów. Usuwanie duplikatów w tekście

Stworzymy teraz program, którego zadaniem będzie usunąć zbędne powtórzenia produktów z listy zakupów. Dla ułatwienia przyjmijmy, że napisy będą podawane w porządku alfabetycznym w oddzielnych wierszach, bez żadnych znaków interpunkcyjnych.

Warto wiedzieć

Jeśli dane wejściowe są uporządkowane, znacznie prościej jest rozwiązać wiele zadań. Na przykład porządek alfabetyczny w indeksie książki lub w słowniku pozwala szybko odnaleźć hasło.

Specyfikacja

Dane: lista słów wpisywanych alfabetycznie, jedno pod drugim, zakończona trzema gwiazdkami (***).

Wynik: lista maksymalnie 20 słów po usunięciu powtórzeń występujących w sąsiednich wierszach.

Sposób rozwiązywania tego problemu można sformułować następująco: dopóki na wejściu pojawiają się nowe wyrazy, należy porównywać ostatnio wczytany wyraz z poprzednim i w razie potrzeby usuwać powtórzenia, a nieusunięte wyrazy dołączać do listy wynikowej.

Precyzyjniej ten algorytm można zapisać jako listę kroków:

1. Wczytaj pierwszy wyraz. Zapamiętaj go jako pierwszy na liście wynikowej i wczytaj kolejny wyraz.
2. Dopóki ostatnio wczytany wyraz nie jest ciągiem znaków ***, wykonuj kolejno kroki 3 i 4.
   3. Jeśli ten wyraz jest różny od poprzedniego, to dodaj go jako kolejny na listę wynikową.
   4. Wczytaj kolejny wyraz.

Przeanalizujemy teraz działanie algorytmu dla listy słów. Wygodnie będzie to zrobić w tabeli:

Napis	Czy dublet?	Czy koniec listy?	Lista wynikowa
chleb	–	nie	chleb
cytryna	nie	nie	chleb cytryna
mleko	nie	nie	chleb cytryna mleko
ser	nie	nie	chleb cytryna mleko ser
ser	tak	nie	chleb cytryna mleko ser
szynka	nie	nie	chleb cytryna mleko ser szynka
***	nie	tak	chleb cytryna mleko ser szynka

Tabela 1.3. Analiza działania algorytmu usuwającego powtórzenia

Tabelę 1.3 uzupełniamy od lewej do prawej, według kolejnych kroków algorytmu. Kolorem zaznaczono etap, w którym nowy wyraz był powtórzeniem.

Warto wiedzieć

Wyszukiwanie duplikatów jest operacją szczególnie istotną w pracy analityków danych. Niektóre aplikacje wspomagające ich pracę, np. arkusze kalkulacyjne, zawierają wbudowane funkcje pozwalające usuwać zduplikowane wartości.

Ćwiczenie 6

Przeanalizuj działanie algorytmu dla listy zakupów składającej się z ośmiu wyrazów, wśród których są dwa powtórzenia, a jedno pojawia się na końcu.

Oto fragment kodu źródłowego programu *Usuwanie powtórzeń*:

Fragment kodu źródłowego programu *Usuwanie powtórzeń*

```
1.  const int N = 20;
2.
3.  int main()
4.  {
5.      string wynik[N];
6.      string nowy, stary;
7.      int i = 0;
8.
9.      cin >> nowy;
10.     wynik[0] = nowy;
11.
12.     stary = nowy;
13.     cin >> nowy;
14.     while (nowy != "***" && i < N-1)
15.     {
16.         if (nowy != stary)
17.         {
18.             i++;
19.             wynik[i] = nowy;
20.         }
21.         stary = nowy;
22.         cin >> nowy;
23.     }
24.
25.     for (int j=0; j<=i; j++)
26.         cout << wynik[j] << endl;
27.
28.     return 0;
29. }
```

Dobra rada

Łańcuch *** w programie *Usuwanie powtórzeń* pełni funkcję wartownika. Używaj wartowników wtedy, kiedy chcesz oznaczyć warunki brzegowe realizacji programu, np. nie chcesz wyjść poza listę.

W wierszu 1 zadeklarowano stałą N i przypisano jej wartość 20. W linii 5 zadeklarowano tablicę N łańcuchów, w której będzie przechowywana lista produktów. W linii 6 zadeklarowano łańcuchy znaków, które będą służyły do porównania dwóch podanych po sobie napisów. Zmienna i deklarowana w linii 7 będzie licznikiem. W wierszach 9–10 wczytujemy pierwszy, a w wierszu 13 drugi napis. W pętli (linie 14–23) sprawdza się, czy podany ostatnio napis jest różny od ***

oraz czy licznik napisów nie przekracza wartości stałej N. Jeśli oba warunki są spełnione, kod w wierszach 16–20 zapobiega wstawianiu powtórzonej nazwy do tablicy. W liniach 25–26 wypisywana jest lista produktów.

Ćwiczenie 7

Zapisz kod źródłowy programu *Usuwanie powtórzeń*. Skompiluj kod i sprawdź działanie programu dla różnych danych wejściowych.

1.4. Wyszukiwanie wzorca w tekście

Wyszukiwanie wzorca w tekście, nazywane też dopasowywaniem wzorca (ang. *pattern matching*), to szukanie wystąpienia danego łańcucha znaków w tekście. Celem jest określenie, w którym miejscu w tekście znajduje się wzorzec, o ile w ogóle w nim występuje.

Wyszukiwanie wzorca w tekście

Wzorca szukamy np. wtedy, gdy chcemy znaleźć konkretne pojęcie w spisie treści lub indeksie podręcznika. Wówczas nasze oko stara się dostrzec dany ciąg znaków.

Warto wiedzieć

Od rozwiązania problemu wyszukiwania wzorca w tekście można też oczekiwać, że wskaże ono wszystkie wystąpienia wzorca, jeśli jest ich więcej.

Wykreślanka

Na rysunku 1.6 znajduje się przykład zadania nazywanego wykreślanką. Polega ono na odnajdowaniu ukrytych wyrazów – w tym wypadku imion. Mogą one być zapisane poziomo, pionowo lub na ukos, także wspak. Wyszukiwanie imion pośród porozrzucanych na planszy liter jest przykładem szukania wzorca.

R	O	M	A	N	A	L	I	C	J	A	Z	A	P	M	G	P	R
T	E	L	H	C	E	I	C	J	O	W	S	I	A	I	S	A	K
O	A	I	G	O	Z	U	Z	A	N	N	A	L	W	C	O	W	A
M	A	R	I	A	J	U	L	I	A	I	E	E	A	H	N	E	R
E	W	D	O	M	I	N	I	K	T	N	N	M	W	A	I	Ł	O
K	B	O	G	U	M	I	Ł	O	A	E	Ł	A	E	Ł	A	I	L

Rys. 1.6. Wykreślanka z zaznaczonymi imionami

Ćwiczenie 8

a. W pliku, który otrzymasz od nauczyciela (np. *wykreslanka.xlsx*), znajduje się wykreślanka. Znajdź jak najwięcej imion, które w niej ukryto.

b. Przygotuj własną wykreślankę, w której umieścisz jak najwięcej imion osób z twojej klasy.

Program *Szukaj wzorca*

Napiszemy teraz program, który będzie poszukiwał podanego wzorca w tekście. Przyjmiemy pewne uproszczenia. Oto specyfikacja problemu:

Specyfikacja

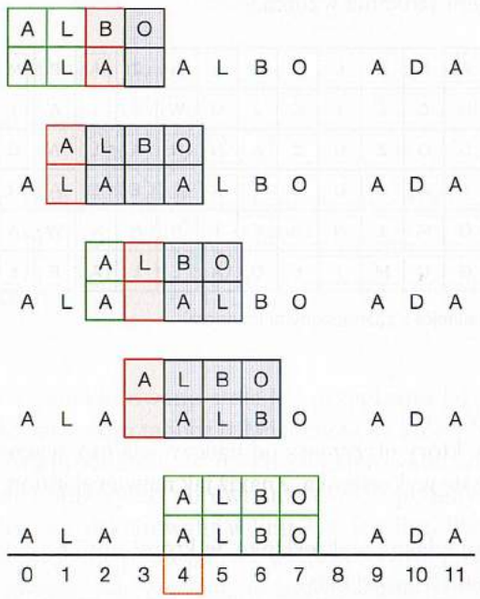
Dane: tekst zapisany wielkimi literami oraz wzorzec – łańcuch zapisany wielkimi literami.

Wynik: pozycja pierwszego wystąpienia wzorca w tekście, rozumiana jako liczba znaków tekstu poprzedzających wystąpienie wzorca, lub liczba –1, jeśli wzorca nie ma w tekście.

Algorytm wyszukiwania wzorca w tekście

Przedstawimy działanie prostego **algorytmu wyszukiwania wzorca w tekście**, tzw. algorytmu naiwnego. Wzorzec będzie przesuwany stopniowo wzdłuż tekstu o jeden znak. Za każdym razem wzorzec będzie ustalał początek fragmentu tekstu, który będzie z nim porównywany znak po znaku (od pierwszego znaku wzorca). Gdy analizowany znak jest zgodny ze znakiem wzorca, porównuje się zgodność kolejnego znaku fragmentu tekstu z kolejnym znakiem wzorca. Jeśli analizowany fragment tekstu jest identyczny ze wzorcem, to wyszukiwanie się kończy i jako odpowiedź zwracana jest pozycja dopasowania, czyli liczba znaków tekstu poprzedzających wystąpienie wzorca.

W tekście ALA ALBO ADA poszukamy wzorca ALBO. Rysunek 1.7 pokazuje działanie algorytmu. Zwracana jest wartość 4.



Rys. 1.7. Naiwne wyszukiwanie wzorca w tekście

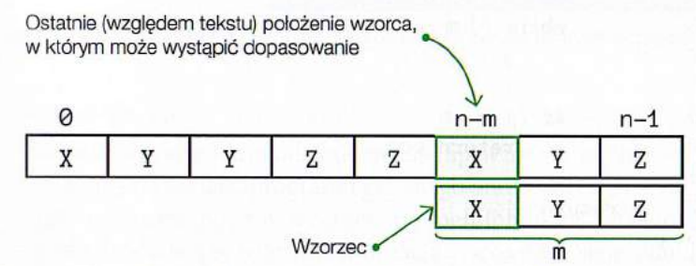
Ćwiczenie 9

a. Zapisz przebieg działania algorytmu poszukiwania w tekście ZA SZCZYTEM ZASZCZYT wzorca ZASZCZYT. Zapisz go w podobny sposób jak na rysunku 1.7.

b. Ile prób dopasowania wzorca trzeba by było wykonać w powyższym ćwiczeniu, gdyby wyszukiwanie zacząć od końca?

Liczbę znaków tekstu do przeszukiwania oznaczmy literą n, a liczbę znaków wzorca – literą m. Aby stwierdzić, czy wzorzec występuje w tekście, dla każdego znaku tekstu, aż do znaku na pozycji o numerze $n - m$, należy sprawdzać zgodność kolejnych znaków tekstu ze znakami tworzącymi wzorzec.

Dlaczego sprawdzamy do znaku o numerze $n - m$? Wyjaśnia to rysunek 1.8. Jeśli tekst ma długość $n = 9$, a wzorzec $m = 3$, to ostatni indeks ma wartość 6 (znaki indeksujemy od 0).



Rys. 1.8. Wyznaczenie indeksu ostatniego znaku tekstu, z którym porównujemy wzorzec

Oto lista kroków algorytmu:

1. Wczytaj tekst i wzorzec.
2. Oznacz długość tekstu literą n, a długość wzorca - literą m.
3. Dla każdej liczby z zakresu od 0 do n - m wykonuj kolejno kroki 4 i 5.
   4. Wybierz fragment tekstu długości m, począwszy od znaku na pozycji równej licznikowi pętli, i porównaj go ze wzorcem.
   5. Jeśli fragment i wzorzec są identyczne, to zwróć jako odpowiedź liczbę równą licznikowi pętli i zakończ algorytm.
6. Zwróć wartość -1.

Zauważ, że w powyższym algorytmie licznik pętli, w momencie znalezienia wzorca w tekście, wskazuje jednocześnie indeks (pozycję) pierwszego wystąpienia wzorca w tekście. Jest to możliwe, ponieważ wartości licznika zaczynają się od 0, czyli tej wartości, od której indeksuje się znaki w łańcuchach.

Warto wiedzieć

W teorii algorytmów metodę rozwiązywania problemu, która wymaga sprawdzenia wszystkich przypadków, nazywa się metodą wyczerpującego wyszukiwania (ang. *exhaustive search*) albo metodą siłową (ang. *brute-force method*). O algorytmie mówi się wówczas, że jest naiwny.

Oto kod źródłowy programu *Szukaj wzorca*:

Kod źródłowy programu *Szukaj wzorca*

```
1.  #include <iostream>
2.  #include <string>
3.
4.  using namespace std;
5.
6.  const string TEKST = "ALA ALBO ADA";
7.
8.  int Porownaj(string wzorzec)
9.  {
10.     int n = TEKST.length();
11.     int m = wzorzec.length();
12.
13.     for (int poz=0; poz<=n-m; poz++)
14.     {
15.         int j = 0;
16.         while (j<m && TEKST[poz+j] == wzorzec[j])
17.             j = j + 1;
18.
19.         if (j == m)
20.             return poz;
21.     }
22.     return -1;
23. }
24.
25. int main()
26. {
27.     cout << "Tekst: " << TEKST << endl;
28.     string wzorzec;
29.     cout << "\nPodaj wzorzec: "; cin >> wzorzec;
30.     cout << "Pozycja: " << Porownaj(wzorzec) << endl;
31.     return 0;
32. }
```

Stała globalna

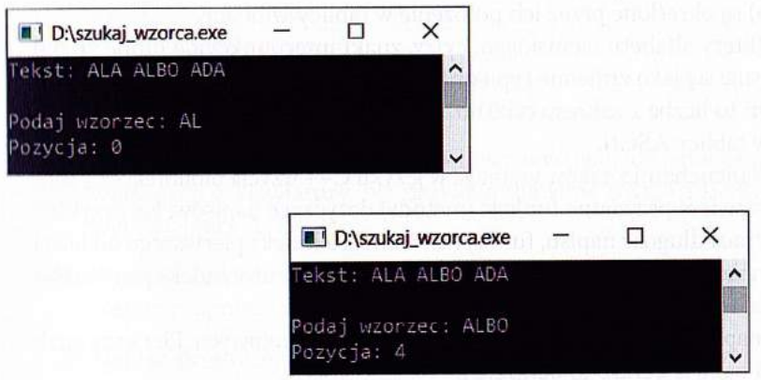
W linii 6 zastosowano **stałą globalną** o nazwie TEKST. Jest to stała, do której dostęp jest możliwy z dowolnego miejsca kodu programu, również z funkcji Porownaj, którą definiujemy w liniach 8–23. Stała globalna musi być zdefiniowana przed wystąpieniem funkcji głównej i każdej innej funkcji lub zmiennej, które mogłyby ją wykorzystywać. W liniach 10 i 11 zadeklarowano zmienne n i m, które będą pamiętały długość odpowiednio tekstu i wzorca.

Pętle zagnieżdżone

Wewnątrz funkcji Porownaj zastosowano **pętle zagnieżdżone** (ang. *nested loops*), czyli pętlę umieszczoną wewnątrz innej pętli. Zewnętrzna pętla (**for**), zapisana w wierszach 13–21, służy do przesuwania wzorca po kolejnych znakach tekstu, a wewnętrzna (**while**), w liniach 16–17, przechodzi przez kolejne znaki wzorca (krok 4 z listy kroków).

Instrukcja warunkowa w linii 19 sprawdza, czy w napisie występuje wzorzec – wartość zmiennej j będzie równa długości wzorca. Wówczas funkcja kończy działanie i zwraca wartość zmiennej poz (linia 20).

Rysunek 1.9 przedstawia przykładowe wywołania programu.



Rys. 1.9. Wywołania programu *szukaj_wzorca.exe* dla dwóch różnych wzorców

Dobra rada

Pamiętaj, że wielkie i małe litery mają różne kody ASCII. Jeśli tekst wyjściowy jest zapisany wielkimi literami, wówczas wzorzec należy zapisać również wielkimi literami.

Ćwiczenie 10

a. W pliku otrzymanym od nauczyciela (np. *wzorzec_w_tekscie.cpp*) znajdują się szkielet programu głównego oraz tekst, w którym należy wyszukać podany wzorzec. Uzupełnij brakujące fragmenty kodu źródłowego, włącznie z funkcją Porownaj. Skompiluj kod i uruchom program dla wzorców: AGA oraz ALBO.

b. Sprawdź działanie programu dla własnego tekstu i wzorca.

Naiwnego algorytmu wyszukiwania wzorca nie można by było z powodzeniem zaimplementować np. w czytniku PDF. Okazuje się, że dla dłuższych tekstów jest on bardzo powolny. Znane są bardzo efektywne algorytmy dopasowywania wzorca w tekście i to z nich korzysta się w praktyce przy przetwarzaniu tekstów.

Warto wiedzieć

Efektywna metoda wyszukania wzorca w tekście jest zaimplementowana w języku C++ jako funkcja (metoda) **find**. Używaliśmy jej w programie sprawdzającym poprawność adresu e-mail.

A to ciekawe

Alfabet Braille'a, czyli szukanie wzorca

Alfabet Braille'a to system zapisu informacji z wykorzystaniem dwóch stanów: punkt wypukły i niewypukły. Znaki przedstawione są jako sześciopunkty, czyli kombinacje punktów umieszczonych w dwóch kolumnach, po trzy punkty w każdej. Czytanie książek brajlowskich oznacza w praktyce wielokrotne dopasowywanie wzorca opuszkami palców w celu odczytania kolejnych znaków tekstu.

Podsumowanie

- W pamięci komputera znaki, jak każdy inny rodzaj informacji, są zapisywane jako liczby. Liczby odpowiadające konkretnym znakom (literom, cyfrom, znakom interpunkcyjnym i innym symbolom) są określone przez ich położenie w tablicy Unicode.
- Pojedyncze znaki (litery alfabetu łacińskiego, cyfry, znaki interpunkcyjne i inne znaki) w języku C++ zapisuje się jako zmienne typu **char**.
- Zmienna typu **char** to liczba z zakresu od 0 do 127, która jest kodem liczbowym odpowiedniego znaku w tablicy ASCII.
- Praca z tekstem i z łańcuchami znaków wymaga w języku C++ użycia biblioteki `string`.
- Biblioteka `string` zawiera przydatne funkcje (metody) dotyczące napisów. Na przykład funkcja `length` zwraca długość napisu, funkcja **find** zwraca indeks pierwszego od lewej wystąpienia szukanego wzorca w danym łańcuchu, a funkcja `rfind` – indeks pierwszego wystąpienia wzorca od końca łańcucha.
- Dostęp do znaku w napisie uzyskuje się za pomocą nawiasów kwadratowych. Pierwszy znak łańcucha znaków o nazwie `tekst` to `tekst[0]`.
- Łańcuchy znaków (lub łańcuch i pojedynczy znak) można scalać, używając operatora `+`.

Zadania

* **1** Napisz program, który wczyta z klawiatury nazwę miejscowości (bez spacji) i wypisze co drugą jej literę. Litery powinny być rozdzielone spacjami.

* **2** Napisz program, który wczyta z klawiatury 10 nazw kolorów i wypisze najdłuższą z nich. Jeśli takich nazw jest więcej, to program powinien wyświetlić nazwę podaną najpóźniej.

* **3** Napisz program, który wczyta 10 imion osób z twojej klasy i wypisze je w postaci tzw. trójkąta Floyda. Oznacza to, że w pierwszym wierszu będzie jedno imię, a w kolejnych odpowiednio dwa, trzy i cztery.

```
1
2 3
4 5 6
7 8 9 10
```

* **4** Napisz program, który wczyta z klawiatury łańcuch znaków (składający się z kombinacji cyfr oraz małych i wielkich liter) i wypisze tylko wielkie litery, które pojawiły się w tekście.

* **5** Napisz program, który wczyta wyraz wzorzec, a następnie będzie wczytywać kolejne wyrazy aż do pojawienia się ciągu znaków ***. Następnie program powinien wyświetlić liczbę powtórzeń wzorca pośród wczytanych wyrazów.

* **6** Napisz program, który dla podanego wyrazu wyświetli jego zapis wspak (od ostatniej do pierwszej litery).

** **7** Napisz program, który będzie zliczał liczbę samogłosek wewnątrz łańcucha znaków.

** **8** Taumatawhakatangihangakoauauotamateaturipukakapikimaungahoronukupokaiwhenuakitanatahu to nazwa jednego ze wzgórz w Nowej Zelandii i jedna z najdłuższych nazw geograficznych na świecie. Korzystając z pliku, który przekaże ci nauczyciel (np. *wzgorze.txt*), sprawdź:
   a. długość tej nazwy,
   b. liczbę występujących w niej liter: a, e, i, o, u, y,
   c. czy w nazwie występuje fraza „eaturip".

** **9** Napisz program, który wyświetli litery występujące jednocześnie w dwóch słowach wpisanych z klawiatury. Na przykład dla słów ARKA i BARAK program powinien wypisać litery A, K i R.

** **10** Wyrazy różniące się dowolną jedną literą nazywa się metagramami, np.: kasa i kara. Napisz program sprawdzający, czy podana para wyrazów to metagramy.

** **11** Napisz program, który z łańcucha znaków będzie usuwał wszystkie samogłoski i zastępował je znakiem podkreślenia (_).

** **12** Napisz funkcję o nagłówku `int ile(string wzorzec)`, która zwróci liczbę całkowitą oznaczającą liczbę wystąpień wzorca w tekście. Przyjmij, że tekst jest zapisany w stałej globalnej o nazwie TEKST.

** **13** Napisz program, który po podaniu daty urodzenia (dzień i miesiąc) wypisze odpowiadający jej znak zodiaku. Na przykład dla danych `21.07` wynikiem będzie napis `rak`.

** **14** Pod kodem $7F_{16}$ (binarnie 0111 1111) w tablicy ASCII kryje się kod znaku sterującego DEL (od ang. *delete*, czyli usunąć). Poszukaj wiarygodnej informacji wyjaśniającej przyczynę umieszczenia kodu tego znaku w tym miejscu. Jaki miało to związek z zapisem na taśmie dziurkowanej używanej w drukarkach dalekopisów? Przygotuj krótką prezentację na temat ewolucji kodów stosowanych w telegrafii i teleksach. Uwzględnij kody Baudota–Murraya, ITA2 oraz ASCII.

** **15** Napisz funkcję, która kompresuje tekst prostą metodą opartą na zliczaniu powtarzających się liter (jedna litera może się powtarzać maksymalnie 9 razy). Na przykład wynikiem dla łańcucha `aaabbcaa` będzie `a3b2c1a2`. Jeśli wynik miałby być łańcuchem dłuższym niż ten dany na wejściu, to funkcja powinna zwracać pierwotny łańcuch znaków.

*** **16** Napisz program, który poda liczbę wystąpień każdej litery alfabetu łacińskiego w tekście wprowadzonym do stałej globalnej o nazwie TEKST. Program nie powinien rozróżniać wielkości liter.

*** **17** Napisz program, który podane przez użytkownika zdanie (w języku angielskim) zapisze w kodzie Morse'a. Oprócz liter program powinien też kodować cyfry i kropkę.
Wskazówka: Aby w języku C++ wczytać do zmiennej o nazwie `tekst` ciąg znaków rozdzielony spacjami wpisywany z klawiatury, użyj instrukcji `getline(cin,tekst)`.