Napisy i operacje na plikach

Ciągi znaków (ang. *strings*) stanowią drugi, po liczbach, ważny rodzaj informacji przetwarzanych przez komputery. W języku C łańcuch znaków (napis) jest zapisywany w kodzie programu jako ciąg znaków zawarty pomiędzy dwoma cudzysłowami. W pamięci taki łańcuch jest następującym po sobie ciągiem znaków (`char`), który kończy się znakiem `null` zapisywanym jako `'\0'`. Dlatego, możemy interpretować napis jako o tablicę znaków. W konsekwencji, jeśli mamy napis, do poszczególnych znaków odwołujemy się jak w każdej tablicy (pierwszy znak ma indeks 0):

```
char* text1 = "modyficable string"; // możemy modyfikować litery w tym napisie
const char* text2 = "example of a character string"; // tego napisu nie możemy modyfikować
char text3 [] = {'a', 'n' ,'o', 't', 'h', 'e', 'r', ' ', 's', 't', 'r', 'i', 'n', 'g', '\0'};
// możemy stworzyć napis jako tablicę
cout << text2[1]; // wypisze na ekranie literę x
```

Słowo `const` użyte w deklaracji napisu `text2` oznacza iż nie pozwalamy na modyfikowanie treści tego napisu. Jednak okazuje się, że bezpośrednie operowanie na łańcuchach znaków jest wysoce niedogodne. Na przykład, aby połączyć dwa napisy w jedną całość należy wykonać kilka czynności takich jak alokacja pamięci dla nowego napisu, kopiowanie znaków z obu napisów do nowej tablicy znaków oraz zwolnienie już nie używanej pamięci. Dlatego w języku C++ wprowadzono specjalną klasę o nazwie `string`, która znacząco ułatwia posługiwanie się napisami w programie gdyż zawiera w sobie wiele przydatnych metod, które implementują funkcjonalności potrzebne przy operowaniu napisami. Poniżej kilka przydatnych metod z klasy `string`:

`string (const char* s)`	konstruktor
`size_t size()`	zwraca długość napisu
`void clear()`	czyści zawartość napisu
`[size_t pos]`	Zwraca znak w napisie o indeksie `pos`
`+=, +`	łączy dwa napisy
`==`	zwraca `true` gdy dwa napisy są identyczne
`substr(size_t pos = 0, size_t len = npos)`	Zwraca podciąg w napisie
`find (const string& str, size_t pos = 0)`	poszukuje podciąg `str` w napisie
`insert (size_t pos, const string& str);`	wstawia str do bieżącego stringu na pozycji `pos`
`erase (size_t pos = 0, size_t len = npos);`	usuwa ze stringu podciąg na pozycji `pos` i długości `len`
`replace (size_t pos,size_t len,const string& str);`	zastępuje podciąg na pozycji `pos` i długości `len` stringiem `str`

Występująca w powyższej tabeli stała `npos` oznacza koniec stringu.

Poniżej przykład użycia klasy `string` oraz operatora `+` do połączenia dwóch napisów:

```
#include <iostream>
#include <string> // załączamy plik string
using namespace std;
int main()
{  string str1 = "hello";
    string str2 = " world";
    string str3 = str1 + str2;
    cout<< str3; // wypisze na ekranie hello world
    if (str1 == str2 ) { cout << "str1 equals str2";} // porównujemy dwa napisy
    return 0; }
```

W języku C++ mamy również dostępną funkcję `to_string()`, która umożliwia konwersję typów podstawowych (np. `int`, `double`, etc.) na obiekt klasy string. Na przykład: `double x=1.0; string str=to_string(x);`. Konwersję w drugą stronę, czyli z obiektu klasy string na jeden z typów podstawowych, można dokonać za pomocą jednej z następujących funkcji: `stoi()`, `stol()`, `stoul()`, `stoll()`, `stoull()`, `stof()`, `stod()`, `stold()`. Na przykład konwersja stringu zawirającego liczbę dziesiętną na typ `int` ma postać: `string str_dec = "1987520"; int value=stoi(str_dec,nullptr,10);`. Ostatni argument tej funkcji (10) to podstawa systemu w którym zapisana jest konwertowana liczba. Dla systemu binarnego będzie to 2 zaś dla szesnastkowego 16.

Standardowo znaki przechowywane w obiektach klasy `string` są typu `char` czyli pojedynczy znak może reprezentować maksymalnie 265 różnych wartości kodowych ze standardu ASCII. Jeśli w programie potrzebujemy operować na znakach spoza tego standardu jak na przykład, litery specyficzne dla jakiegoś języka, emotikony lub inne znaki graficzne to musimy posłużyć się kodowaniem *unicode*. Wartości kodowe znaków w standardzie unicode znacznie przekraczają wartość 256 i dlatego w języku C++ stworzono specjalną klasę `wstring` (ang. *wide character string*) umożliwiającą przechowywanie znaków kodowanych w *unicode*. Klasa `wstring` przechowuje znaki jako typ `wchar_t`, którego rozmiar wynosi dwa lub więcej bajtów i dlatego umożliwia przechowywanie znaków o wartościach kodowych znacznie większych niż 255. Jeśli chcemy stworzyć literał złożony ze znaków typu `wchar_t` to należy go poprzedzić znakiem `L`. Aby wypisać na ekranie, lub wprowadzić z klawiatury znaki typu `wchar_t` należy posłużyć się odpowiednio strumieniami `wcout` oraz `wcin`. Pojedyncze znaki *unicode* można przedstawić za pomocą wartości kodowej danego znaku poprzedzonej przedrostkiem `\u`. Na przykład, `L'\u2713'` reprezentuje znak ✓. Kody *unicode* dla każdego znaku można znaleźć na stronie unicode-table.com.

```
#include <iostream>
#include <string>
#include <locale>
#include <clocale>
using namespace std;
int main()
{
  // konieczne aby znaki poprawnie wyświetlały się w konsoli
  setlocale(LC_ALL, "");//jedynie w przypadku gdy używamy funkcji języka C (np. printf, etc)
  locale::global(locale("")); // gdy używamy wcin lub wcout w C++
  wcout.imbue(locale());

  wstring name; // napis złożony ze znaków typu wchar_t
  wcout << L"\u2713 wprowadź imię: ";
  wcin >> name;
  wcout << L"Witaj, " << name << L'\u263a' << endl;
  return 0; }
```

Tekst w unicode wyświetli się poprawnie jedynie w przypadku gdy użyta przez konsolę czcionka zawiera odpowiednie znaki.

*

W języku C++ do komunikacji z plikami wykorzystuje się tak zwane strumienie, które są specjalnymi obiektami umożliwiającymi zapisywanie lub odczytywanie danych do lub z pliku. Aby móc użyć w programie strumienie operujące na plikach należy dołączyć plik nagłówkowy `fstream`. Pracę z plikiem rozpoczyna się od jego otwarcia. W tym celu należy utworzyć obiekt strumienia plikowego `fstream`. Opcjonalnie można użyć klasy `ifstream`, która otwiera plik do odczytu lub klasy `ofstream`, która otwiera plik do zapisu. Gdy plik jest otwarty, dalsze czynności wykonuje się na ogół tak samo, jak przy użyciu strumieni `cin` i `cout`, to znaczy za pomocą operatorów << i >>. Jeśli chcemy operować na kilku plikach to dla każdego pliku należy utworzyć oddzielny strumień. Koniecznie należy pamiętać o tym, że gdy otwarty plik nie jest nam już dalej w programie potrzeby to należy go zamknąć. Poniżej kilka przydatnych metod z klasy `fstream`:

`void open( const char* file_path,  int mode = ios_base::in | ios_base::out)`	otwiera plik wskazany przez `file_path`
`bool is_open()`	zwraca `true` gdy plik jest otwarty
`void close()`	zamyka plik
`seekg( streampos pos )`	ustawia bieżącą pozycję w pliku na `pos`
`streampos tellg()`	zwraca bieżącą pozycję w pliku
`get (char& c)`	pobiera kolejny znak z pliku
`bool eof()`	zwraca true gdy osiągnięto koniec pliku
`>> oraz <<`	operatory wejścia i wyjścia

W funkcji `open` jako ścieżkę do pliku można podać tak zwaną ścieżkę bezwzględną (absolutną) lub względną. Ścieżka bezwzględna zawsze zaczyna się od korzenia w systemie plików a następnie poprzez kolejne katalogi prowadzi do docelowego pliku. W systemach z rodziny Windows korzeniem będzie litera oznaczająca zamontowany dysk zaś w systemach z rodziny Linux będzie to katalog nadrzędny oznaczony znakiem `/`. Dla przykładu, ścieżka absolutna w Windows może mieć postać `C:\Users\Ewa\plik.txt`. Ścieżka względna odnosi się zawsze do tak zwanego bieżącego katalogu roboczego. Na ogół bieżący katalog roboczy to folder w którym znajduje się skompilowany i uruchomiony plik binarny (plik .exe w Windows). Jeśli na przykład nasz katalog roboczy to `C:\Windows\Users\Ewa` to ścieżka względna `plik.txt` wskazuje na plik `C:\Windows\Users\Ewa\plik.txt`. Warto również pamiętać, że systemach Windows jako separator katalogów standardowo używa się znaku ukośnika wstecznego `\` zaś w systemach Linux znaku ukośnika `/`. W języku C++ znak ukośnika wstecznego jest znakiem specjalnym dlatego w kodzie źródłowym należy go zapisać w następujący sposób `\\`. Na ogół najnowsze systemy Windows potrafią również poprawnie zinterpretować znak ukośnika i dlatego dla uproszczenia można zawsze posługiwać się tym znakiem jako separatorem katalogów. Poniżej przykładowy program demonstrujący otwarcie pliku do odczytu i zapisu oraz sposób odczytania i zapisania zmiennej typu całkowitego do/z pliku.

```
#include <fstream> // załączamy odpowiedni plik nagłówkowy
using namespace std;
int main()
{
    int x = 5; int y;
    string line;
    fstream file ("C:\\scores.txt"); // Utworzenie strumienia do zapisu i odczytu do/z pliku
C:\\scores.txt
    if ( file.is_open() ) // sprawdzamy czy plik został poprawnie otwarty
    {
         file >> y; // wczytaj liczbę całkowitą z pliku i zapisz ją w zmiennej y
         file << "x variable is " << x ; // zapisz tekst i wartość zmiennej x do pliku
         getline(file, line); // odczytaj jedną linię tekstu z pliku
         file.close(); // koniecznie pamiętajmy o zamknięciu otwartego pliku
     }
    return 0;
}
```

Dostępne są również modyfikatory umożliwiające modyfikowanie formatu w którym wpisywane są dane za pomocą strumienia. Aby posłużyć się modyfikatorami należy załączyć plik nagłówkowy `iomanip`. W poniższej tabeli podano kilka przykładowych modyfikatorów:

`std::dec`	wypisz liczbę w postaci dziesiętnej
`std::hex`	wypisz liczbę w postaci szesnastkowej
`std::fixed`	wypisz liczbę w notacji zwykłej np. 1.3
`std::scientific`	wypisz liczbę w notacji naukowej np. 1.3e-5
`std::left, std::right`	wyrównaj do lewej lub prawej
`std::setprecision(n)`	wypisz liczbę z precyzją `n` cyfr
`std::setw(n)`	ustal szerokość pola na którym będzie wypisana liczba na `n` znaków

Na przykład wypisanie liczby w polu o szerokości 30 znaków z precyzją 2 cyfr z wyrównaniem do prawej będzie mieć postać: `cout<<std::setw(30)<<std::setprecision(10)<<std::right<<std::acos(-1.0L);`
W wyniku otrzymamy napis: `3.1`

zadania

1. Napisz program, który wczyta jedną linię tekstu z klawiatury i wypisze na ekranie ilość cyfr występujących we wprowadzonym tekście.

2. Napisz program, który wczyta z pliku jedną linię tekstu a następnie zapisze ją w nowym pliku w odwrotnej kolejności. Posłuż się obiektami klasy `string`, `ifstream` oraz `ofstream`. Stwórz w programie funkcję o nazwie `string string_reverse(const string& in)`, która zwraca obiekt klasy `string` z odwróconą kolejnością liter.

3. Napisz program, który na początku tworzy napis o treści "`Congratulations Mrs. <name>, you and Mr. <name> are the lucky recipients of a trip for two to XXXXXX. Your trip to XXX is already scheduled `". Wykonaj następujące operacje na tym stringu:

    a. Zastąp każde wystąpienie "<name>" słowem "Smith",

    b. zastąp każde wystąpienie XXXX (niezależnie od ilości liter X) słowem "Siberia"

    c. dodaj słowo "un" bezpośrednio przed słowem "lucky"

    d. dodaj słowo "in December" na końcu stringu.

    e. wypisz powstały string na ekranie.

4. W badaniach na strukturą genomu ważną rolę odgrywają sekwencje DNA będące cyklicznymi przesunięciami innych sekwencji. Ciąg liter `s` jest przesunięciem cyklicznym ciągu `t` wtedy gdy oba ciągi pokrywają się przy przesunięciu ciągu `s` o dowolną ilość liter. W wyniku przesunięcia cyklicznego, litery wychodzące poza koniec napisu są przenoszone na jego początek. Na przykład, ciąg liter `ACTGACG` jest cyklicznym przesunięciem ciągu `TGACGAC`. Napisz program, który wczyta z klawiatury dwie linie tekstu i wypisze informację czy są one wzajemnie przesunięciem cyklicznym.

5. Napisz program będący prostym kalkulatorem walut. Dla uproszczenia przyjmij, że program będzie operował na walutach: PLN, USD oraz EUR. Stwórz jeden plik tekstowy w którym w oddzielnej linii będzie zapisany kurs kupna 1 USD w każdej z dostępnych walut. W programie uwzględnij również tak zwany spread walutowy w wysokości 5% (dla przypomnienia, spread walutowy to różnica pomiędzy ceną kupna i sprzedaży waluty). Po uruchomieniu, program powinien wczytać dane z tego pliku. Następnie powinien wyświetlić użytkownikowi menu umożliwiające wybór dwóch walut oraz kierunek konwersji. W następnej

kolejności użytkownik wprowadza sumę w walucie źródłowej po czym program wyświetla wartość w walucie docelowej. Na koniec program umożliwia powrót do początkowego menu.

6. Zaprojektuj i zaimplementuj prostą bazę danych do przechowywania informacji o filmach i muzyce w twojej bibliotece. Zastanów się jaka powinna być struktura rekordu i jak powinna być zaimplementowana? Zaimplementuj podstawowe funkcjonalności: dodawanie i usuwanie rekordu, zapis i odczyt z pliku.