Politechnika Wrocławska

Wydział Elektroniki,
Fotoniki i Mikrosystemów

Laboratorium informatyki

Ćwiczenie nr 7. Operacje na plikach

Zagadnienia do opracowania:

- standardowe wejście i wyjście
- tryby otwarcia plików
- zapis do pliku i odczyt z pliku w języku *C*
- wskaźniki plikowe
- zapis do pliku i odczyt z pliku w języku *C++*

Spis treści

1. Cel ćwiczenia

Celem ćwiczenia jest opanowanie umiejętności odczytu i zapisu danych do pliku z wykorzystaniem aplikacji konsolowej, napisanej w języku *C* lub *C++*.

2. Wprowadzenie

2.1. Odczyt i zapis do pliku w języku *C*

Niezbędną umiejętnością, wykorzystywaną w tworzeniu aplikacji w językach *C* i *C++*, jest obsługa plików systemowych. Pliki mogą być otwierane przez program komputerowy w dwóch trybach: ***tekstowym*** i ***binarnym***. W ***trybie binarnym*** każdy bajt danych jest dosłownie interpretowany przez aplikację. W ***trybie tekstowym*** dane zawarte w pliku mogą się różnić od danych przetwarzanych przez program. Różnica wynika z *modelu pliku tekstowego w języku C*. Warto odnieść się w tym miejscu do znaków ASCII (patrz: rys. 2.4, ćw. 6). Standard dzieli znaki na ***drukowalne*** – interpretowane jako tekst oraz na ***sterujące*** – niewyświetlane, określające sposób obsługi tekstu. Przykładem tych drugich mogą być symbole **końca pliku** czy **końca linii**. W systemie *Windows* koniec linii zapisywany jest za pomocą sekwencji $\boldsymbol{\backslash r\backslash n}$, a w systemach *Unix* – $\boldsymbol{\backslash n}$. **Model pliku w języku *C*** jest zgodny z formatowaniem systemu *Unix*. Z tego powodu otwarcie pliku w ***trybie tekstowym*** przez aplikację działającą w systemie *Windows* poskutkuje niejawnym zastąpieniem wszystkich sekwencji $\boldsymbol{\backslash r\backslash n}$ na $\boldsymbol{\backslash n}$ (analogicznie usunięte zostaną symbole końca pliku *Windowsa* – Ctrl-Z (\x1A)). Otwarcie tego samego pliku w ***trybie binarnym*** spowoduje, że znaki $\boldsymbol{\backslash r}$ i ***Ctrl-Z*** będą interpretowane tak, jak znaki drukowalne, a w związku z tym możliwa będzie obsługa tychże znaków w kodzie programu. Warto nadmienić, że system *Unix* przewiduje tylko jeden format pliku. Obsługa plików w ***trybie tekstowym*** i ***binarnym*** w tym systemie da identyczny rezultat.

W momencie uruchomienia aplikacji języka *C* otwarte zostają (automatycznie) trzy ***pliki standardowe***: ***standardowe wejście***, ***standardowe wyjście*** oraz ***standardowe wyjście dla błędów***. Są to przykłady **standardowego wejścia/wyjścia wysokiego poziomu** (*ang. standard high-level I/O*), tj. niezależnego od systemu operacyjnego. Funkcje biblioteczne obsługi **standardowego wejścia/wyjścia wysokiego poziomu** zawarte są w nagłówkach ***stdio.h*** (dla języka *C*) oraz ***iostream*** (dla języka *C++*) [2]. Plik ***standardowego wejścia*** zapewnia najczęściej transfer danych z klawiatury do programu, natomiast ***standardowego wyjścia*** oraz ***standardowego wyjścia dla błędów*** transfer z programu na ekran komputera. Wykorzystując **przekierowanie** można wskazać inne pliki jako **pliki standardowego wejścia/wyjścia**. **Standardowe wejście** dostarcza dane do funkcji ***scanf()***. Analogicznie, **standardowe wyjście** jest plikiem, do którego zapisuje dane funkcja ***printf()***. ***Strumienie standardowego wejścia i wyjścia są buforowane***. Oznacza to, że dane przetwarzane przez *funkcje obsługi wejścia/wyjścia* są automatycznie (niejawnie) gromadzone w tymczasowo zaalokowanym obszarze pamięci (buforze). Przesył danych między **buforem** a **standardowym wejściem/wyjściem** odbywa się porcjami (najczęściej o rozmiarze wielokrotności 512 B), co skutkuje zwiększeniem prędkości transferu [2].

W języku *C* do otwarcia pliku wykorzystuje się funkcję ***fopen()***. Funkcja ta przyjmuje **dwa łańcuchy znakowe**: pierwszy określa nazwę pliku, który ma zostać otwarty (w szczególności względną lub bezwzględną ścieżkę do pliku), drugi to tryb otwarcia pliku. W tabeli 1 zestawiono tryby otwarcia pliku wspierane przez funkcję ***fopen()***. Tryby można łączyć. Dodatkowe dwa opcjonalne łańcuchy określają kolejno tryb tekstowy - **"t"** i binarny - **"b"**. Jeżeli żaden z nich nie zostanie przekazany do funkcji ***fopen()***, domyślnie wybrany zostanie **tryb tekstowy** (***nie ma to znaczenia w systemach Unix***). Przykłady przedstawiono na listingu 1. (bieżąca lokalizacja stanowi miejsce wywołania programu).

Tabela 1. Tryby otwarcia pliku wspierane przez funkcję *fopen()* [1]

Łańcuch znakowy	Objaśnienie	Gdy plik istnieje	Gdy plik nie istnieje
”r”	otwarcie pliku do odczytu	odczyt od początku pliku	błąd otwarcia
”w”	utworzenie pliku do zapisu	usuń zawartość pliku	utworzenie nowego
”a”	dopisywanie do pliku	dopisz do końca pliku	utworzenie nowego
”r+”	otwarcie pliku do odczytu/zapisu	odczyt od początku pliku	błąd otwarcia
”w+”	utworzenie pliku do odczytu/zapisu	usuń zawartość pliku	utworzenie nowego
”a+”	otwarcie pliku do odczytu/dopisywania	dopisz do końca pliku	utworzenie nowego

```
1 // Otwarcie do odczytu pliku results.txt w trybie
     tekstowym
2 fopen("results.txt", "r");
3 // Utworzenie i zapis do pliku tekstowego lezacego
     pod wzgledna sciezka data\test.txt
4 fopen("data\test.txt", "wt");
5 // Otwarcie pliku do dopisywania i odczytu w trybie
     binarnym lezacego pod bezwgledna sciezka C:\Users
     \pwr\students.dat
6 fopen("C:\Users\pwr\students.dat", "a+b");
```

Listing 1. Przykłady wywołania funkcji *fopen()*

Funkcja ***fopen()***, oprócz otwarcia pliku, tworzy potrzebne bufory oraz strukturę danych, zawierającą informacje o pliku i buforach. Najczęściej są to: identyfikator pliku, wskaźnik do początku bufora, licznik bajtów skopiowanych do bufora, wskaźnik bieżącego położenia w strumieniu, sygnalizatory błędów i końca pliku [2]. Funkcja ***fopen()*** zwraca wskaźnik na utworzony pakiet danych. Jest to tzw. ***wskaźnik plikowy*** (*ang. file pointer*) – wskaźnik na typ pochodny ***FILE***, zdeklarowany w nagłówku ***stdio.h***. W przypadku niepowodzenia otwarcia pliku (niedozwolony dostęp, błędna ścieżka, itp.)

zwracana jest wartość ***NULL***. Również **pliki standardowe** posiadają swoje (predefiniowane) wskaźniki plikowe zdeklarowane w nagłówku ***stdio.h***:

- standardowe wejście – ***stdin***,
- standardowe wyjście – ***stdout***,
- standardowe wyjście dla błędów – ***stderr***.

Przykładowo plik ***data.txt*** można otworzyć w trybie zapisu/odczytu w sposób następujący:

```
1 FILE * fPtr = fopen("data.txt", "w+");
2 if (fPtr == NULL) {
3     printf("Failed to open the file\n");
4     exit(EXIT_FAILURE);
5 }
```

Wskaźnik plikowy zwrócony przez funkcję *fopen()* zapisywany jest w zmiennej ***fPtr***. Jeżeli nie udało się otworzyć/utworzyć pliku ***data.txt*** wyświetlony zostaje komunikat i następuje wcześniejsze zakończenie programu. Funkcja ***exit()*** zamyka aplikację przeprowadzając standardowe sprzątanie zasobów – m.in. gwarantuje zamknięcie otwartych plików. Argument przekazany do funkcji *exit()* działa analogicznie, jak wartość zwracana przez funkcję *main()*. Przyjęło się, że ***exit(0)*** oznacza poprawne wykonanie programu, a każda inna wartość określa błędne zakończenie aplikacji. Różne systemy operacyjne mogą implementować różne zakresy zwracanych wartości, dlatego dla pełnej przenośności kodu programu warto stosować makrodefinicje ***EXIT_SUCCESS*** (poprawne wykonanie aplikacji) oraz ***EXIT_FAILURE*** (błędne wykonanie aplikacji).

Każdy otwarty w programie plik należy zamknąć, aby nie dopuścić do ***wycieku zasobów*** i nie blokować (niecelowo) innym procesom/wątkom jego obsługi. W języku *C* służy do tego funkcja ***fclose()***, przyjmująca ***wskaźnik plikowy*** zwrócony przez funkcję ***fopen()***. Funkcja ***fclose()*** zwalnia

pamięć zaalokowaną na obsługę bufora wykorzystywanego przez funkcję ***fopen()***. W przypadku powodzenia zwracana jest wartość **0**, w innym wypadku – stała **EOF** (*ang. end of file*) o ujemnej wartości (najczęściej -1):

```
if (fclose(fPtr) != 0) {
    printf("Failed to close the file\n");
    exit(EXIT_FAILURE);
}
```

Zapis danych do pliku odbywa się z wykorzystaniem funkcji ***fprintf()***. Posiada ona nagłówek i funkcjonalność zbliżoną do funkcji *printf()*, z tą różnicą, że jako pierwszy argument przyjmuje ***wskaźnik plikowy do strumienia wyjściowego***, do którego ma przesłać dane:

```
int fprintf(FILE * stream, const char * format, ...)
```

Podobnie, jak funkcja *printf()*, również ta funkcja, w przypadku poprawnego wykonania, zwraca liczbę znaków wpisanych do strumienia. W przypadku niepowodzenia zwracana jest liczba ujemna. Przekazując do funkcji ***fprintf()*** jako pierwszy argument wskaźnik plikowy ***standardowego wyjścia*** osiąga się funkcjonalność funkcji *printf()*:

```
// Rownowazne zapisy
fprintf(stdout, "Hello, world!\n");
printf("Hello, world!\n");
```

Odczyt zapisanych danych można zrealizować z wykorzystaniem funkcji ***fscanf()***. Podobnie, jak w przypadku funkcji ***fprintf()***, widoczna jest tu analogia do funkcji *scanf()* z rozszerzeniem o przyjmowany ***wskaźnik plikowy do strumienia wejściowego***. Funkcja zwraca liczbę zmiennych, do których udało się przypisać wartość w przypadku powodzenia, albo wartość ***EOF*** w przeciwnym wypadku. Przekazując do funkcji ***fscanf()*** jako

pierwszy argument wskaźnik plikowy ***standardowego wejścia*** osiąga się funkcjonalność funkcji *scanf()*:

```
char buffer[100];
// Rownowazne zapisy
fscanf(stdin, "%s", buffer);
scanf("%s", buffer);
```

Kompletny program, przeprowadzający operacje zapisu, a następnie odczytu z pliku przedstawiono na listingu 2. Funkcja ***rewind()*** przyjmuje jako argument wskaźnik plikowy zwrócony przez funkcję ***fopen()*** i przesuwa wskaźnik bieżącego położenia w strumieniu na początek pliku. Dzięki temu możliwe jest odczytanie wcześniej wpisanej zawartości pliku przez funkcję ***fscanf()***. Zamknięcie pliku realizowane jest z wykorzystaniem funkcji ***fclose()***.

```
// Standardowe wejscie/wyjscie
#include <stdio.h>
// Deklaracja funkcji exit()
#include <stdlib.h>
// Deklaracja funkcji strlen()
#include <string.h>

int main() {
    // Otworz plik
    FILE * fPtr = fopen("data.txt", "w+");
    if (fPtr == NULL) {
        printf("Failed to open the file\n");
        exit(EXIT_FAILURE);
    }

```

```
16      // Dane do zapisania do pliku
17      const char * text = "Ala ma kota";
18      const unsigned int textLength = strlen(text);
19
20      // Zapisanie danych do pliku
21      if (fprintf(fPtr, text) != textLength)
22          printf("Failed to write to the file\n");
23
24      // Wroc do poczatku pliku
25      rewind(fPtr);
26
27      // Bufor na dane odczytane (+1 na '\0')
28      char buffer[textLength + 1];
29
30      // Odczyt zapisanych danych
31      if (fscanf(fPtr, "%s", buffer) != 1)
32          printf("Failed to read from the file\n");
33      else
34          printf("Data read from file: %s\n", buffer);
35
36      // Zamkniecie pliku
37      if (fclose(fPtr) != 0) {
38          printf("Failed to close the file\n");
39          exit(EXIT_FAILURE);
40      }
41
42      return EXIT_SUCCESS;
43  }
```

Listing 2. Przykład zapisu i odczytu z pliku tekstowego

Funkcje *scanf()* oraz *fscanf()* domyślnie kończą pobieranie danych ze *strumienia wejściowego* w momencie napotkania znaku białego (spacja, tabulator, itp.). Z tego powodu wynikiem działania programu z listingu 2 będzie wypisanie na ekranie łańcucha **"Ala"**. Do pobierania i zapisu całych łańcuchów znakowych można posłużyć się wygodniejszymi funkcjami ***fgets()*** i ***fputs()***. [*Uwaga: można spotkać się również z funkcją* ***gets()****, pobierającą dane ze standardowego wejścia. Jednakże została ona usunięta w standardzie C11 z powodu niebezpieczeństwa użycia, wynikającego z jej podatności na przepełnienie bufora.*] Funkcje ***fgets()*** i ***fputs()*** zdeklarowane są w pliku nagłówkowym ***stdio.h***. Ich deklaracje wyglądają następująco:

```
1 int fputs(const char * str, FILE * stream)
2 char * fgets(char * str, int num, FILE * stream)
```

Funkcja ***fputs()*** zapisuje łańcuch znakowy wskazywany przez ***str*** do pliku wskazywanego przez wskaźnik plikowy ***stream***, bez umieszczania znaku nowej linii. W przypadku sukcesu zwracana jest nieujemna wartość, a w przeciwnym razie wartość ***EOF***.

Funkcja ***fgets()*** pobiera trzy argumenty: **adres**, pod którym mają zostać zapisane odczytane dane – ***str***, **maksymalny rozmiar łańcucha wejściowego** – ***num*** oraz **wskaźnik plikowy do strumienia wejściowego** – ***stream***. W przypadku powodzenia funkcja zwraca łańcuch ***str***, w przeciwnym wypadku zwracana jest wartość ***NULL***. Zmodyfikowany program wykonujący odczyt i zapis do pliku z wykorzystaniem funkcji *fgets()* i *fputs()* przedstawiono na listingu 3.

```
1 // Standardowe wejscie/wyjscie
2 #include <stdio.h>
3 // Deklaracja funkcji exit()
4 #include <stdlib.h>
5 // Deklaracja funkcji strlen()
```

```c
#include <string.h>

int main() {
    // Otworz plik
    FILE * fPtr = fopen("data.txt", "w+");
    if (fPtr == NULL) {
        printf("Failed to open the file\n");
        exit(EXIT_FAILURE);
    }

    // Dane do zapisania do pliku
    const char * text = "Ala ma kota";
    const unsigned int textLength = strlen(text);

    // Zapisanie danych do pliku
    if (fputs(text, fPtr) == EOF)
        printf("Failed to write to the file\n");

    // Wroc do poczatku pliku
    rewind(fPtr);

    // Bufor na dane odczytane (+1 na '\0')
    char buffer[textLength + 1];

    // Odczyt zapisanych danych
    if (fgets(buffer, textLength + 1, fPtr) == NULL)
        printf("Failed to read from the file\n");
    else
        printf("Data read from file: %s\n", buffer);

    // Zamkniecie pliku
```

```
37      if (fclose(fPtr) != 0) {
38          printf("Failed to close the file\n");
39          exit(EXIT_FAILURE);
40      }
41
42      return EXIT_SUCCESS;
43  }
```

Listing 3. Przykład zapisu i odczytu z pliku tekstowego z wykorzystaniem funkcji *fgets()* i *fputs()*

2.2. Odczyt i zapis do pliku w języku *C++*

Język *C++* wprowadził *obiektowe mechanizmy obsługi plików*, analogiczne do *obiektowego wejścia/wyjścia*, zdeklarowane w nagłówku ***fstream***. Są to ***klasy std::ofstream*** (obsługa wyjściowego strumienia plikowego) oraz ***std::ifstream*** (obsługa wejściowego strumienia plikowego). W przeciwieństwie do nagłówka ***iostream*** nie zostały przygotowane predefiniowane globalne instancje (*obiekty*) tych klas (jak miało to miejsce w przypadku *std::cout* oraz *std::cin*). Otwarcie i zamknięcie pliku realizowane jest za pomocą ***metod***[1] ***open()*** oraz ***close()***. Funkcja ***open()*** przyjmuje dwa argumenty – **nazwę pliku** (w szczególności względną lub bezwzględną ścieżkę do pliku) oraz **tryb otwarcia pliku**:

```
// Naglowek metody open() klasy std::ofstream
void open(const char * filename, std::ios_base::
   openmode mode = std::ios_base::out);
// Naglowek metody open() klasy std::ifstream
void open(const char * filename, std::ios_base::
   openmode mode = std::ios_base::in);
```

[1] funkcji wywoływanych na obiekcie

Wspierane tryby otwarcia pliku zestawiono w tabeli 2. Tryb posiada wartość domyślną: std::ios_base::out dla klasy ***std::ofstream*** oraz std::ios_base::in dla klasy ***std::ifstream***. Tryby można łączyć za pomocą operatora alternatywy bitowej – ***operator|***. Różnica między trybem std::ios_base::app i std::ios_base::ate polega na tym, że std::ios_base::app umożliwia wprowadzanie danych wyłącznie na końcu pliku, a std::ios_base::ate jedynie przesuwa wskaźnik bieżącego położenia w strumieniu na koniec pliku (działanie odwrotne do funkcji *rewind()* z języka *C*) [3].

Tabela 2. Tryby otwarcia pliku wspierane przez funkcję *open()* [4]

Tryb	**Objaśnienie**
std::ios_base::app	dopisywanie do pliku
std::ios_base::binary	otwarcie pliku w trybie binarnym
std::ios_base::in	otwarcie pliku do odczytu
std::ios_base::out	otwarcie pliku do zapisu
std::ios_base::trunc	nadpisywanie pliku
std::ios_base::ate	przesunięcie wskaźnika położenia w strumieniu na koniec pliku

W tabeli 3. zestawiono porównanie trybów otwarcia pliku wspieranych przez funkcję ***open()*** z ich odpowiednikami z języka *C*.

Tabela 3. Tryby otwarcia pliku w języku *C++* i *C* [3]

Tryb w języku *C++*	**Tryb w języku *C***
std::ios_base::in	"r"
std::ios_base::out oraz std::ios_base::out | std::ios_base::trunc	"w"
std::ios_base::out | std::ios_base::app	"a"
std::ios_base::in | std::ios_base::out	"r+"
std::ios_base::in | std::ios_base::out | std::ios_base::trunc	"w+"
std::ios_base::binary	"b"

Funkcja ***close()*** jest bezargumentowa. Transfer danych do wejściowego strumienia plikowego może być realizowany (analogicznie jak w przypadku obiektu *std::cout*) za pomocą przeciążonego operatora przesunięcia bitowego w lewo – **operator«**. Metoda ***is_open()*** zwraca wartość ***true***, jeżeli otwarcie pliku się powiodło. Przykład zapisu do pliku tekstowego przedstawiono na listingu 4. ***Metody wywoływane są na obiektach z wykorzystaniem operatora dostępu do składowych – operator.***.

```
#include <iostream>
#include <fstream>

int main() {
  std::ofstream file;
  // Otwarcie pliku
  file.open("data.txt");
  if (!file.is_open()) {
    std::cout << "Failed to open the file" << std::
   endl;
    exit(EXIT_FAILURE);
  }
  // Wpisanie danych do pliku
  file << "Grades:" << std::endl
    << "Jan Kowalski | " << 2.0 << std::endl
    << "Adam Nowak | " << 4.5 << std::endl
    << "Robert Malinowski | " << 3.5 << std::endl;
  // Zamkniecie pliku
  file.close();
  return EXIT_SUCCESS;
}
```

Listing 4. Przykład zapisu do pliku tekstowego z wykorzystaniem klasy *std::ofstream*

Warto nadmienić, że klasa ***std::ofstream*** może korzystać z tych samych funkcji formatujących, co klasa ***std::ostream***, której instancją jest obiekt ***std::cout***. Są to metody takie, jak ***setf()*** czy ***precision()*** [3].

W przypadku klasy ***std::ifstream***, obsługującej ***wejściowy strumień plikowy***, przeciążony został operator przesunięcia bitowego w prawo – **operator»** (w analogii do obiektu *std::cin*). Przykład odczytu danych z pliku przedstawiono na listingu 5. W pliku ***grades.txt*** zapisane są („w słupku") oceny studentów z pewnej grupy laboratoryjnej. Zadaniem programu jest obliczenie średniej arytmetycznej ocen dla całej grupy. Metoda ***eof()*** sprawdza czy napotkano znak końca pliku (***EOF***). Metoda ***fail()*** zwraca wartość ***true***, jeżeli odczyt pliku został przerwany w wyniku błędu. ***Operator»*** może zostać użyty wewnątrz warunku pętli *while*, ponieważ zwraca on oryginał obiektu klasy ***std::ifstream*** (tu: obiekt ***file***), który z kolei, rzutowany na typ logiczny ***bool***, niesie informację czy ostatnia operacja odczytu zakończyła się poprawnie (operacja rzutowania ***std::ifstream*** na ***bool*** jest równoważna negacji wyniku metody ***fail()***).

```
1  #include <iostream>
2  #include <fstream>
3
4  int main() {
5    std::ifstream file;
6    // Otwarcie pliku
7    file.open("grades.txt");
8    if (!file.is_open()) {
9      std::cout << "Failed to open the file" << std::
     endl;
10     exit(EXIT_FAILURE);
11   }
12
13   float grade, sum = 0.0;
```

```
14    unsigned int count = 0;
15    // Wykonuj dopoki odczyt danych sie powiodl i nie
       napotkano EOF
16    /* Rownowazne do:
17       file >> grade;
18       while (!file.fail()) { ... }
19    */
20    while (file >> grade) {
21      ++count;
22      sum += grade;
23    }
24
25    // Sprawdzenie bledu odczytu
26    if (!file.eof() && file.fail())
27      std::cout << "Error occurred during reading the
       file" << std::endl;
28
29    // Obliczenie sredniej ocen
30    if (count != 0)
31      std::cout << "The group average is: " << sum /
       count << std::endl;
32    else
33      std::cout << "No data read" << std::endl;
34
35    // Zamkniecie pliku
36    file.close();
37    return EXIT_SUCCESS;
38  }
```

Listing 5. Przykład odczytu danych z pliku z wykorzystaniem klasy *std::ifstream*

Operator» zakończy odczyt danych z pliku, jeżeli natrafi na znak biały (co może być problematyczne przy wczytywaniu napisów). Aby temu zapobiec można posłużyć się metodą ***getline()***. Pobiera ona dwa argumenty – **adres bufora**, do którego mają zostać zapisane dane odczytane z pliku, oraz **maksymalną liczbę znaków** (łącznie z '\0'), jaka ma zostać do niego wpisana:

```
1 const unsigned int size = 16;
2 char buffer[size];
3
4 std::ifstream file;
5 file.open("data.txt");
6 if (file.is_open()) {
7     file.getline(buffer, size);
8 }
```

3. Program ćwiczenia

Zadanie 1. Korzystając z języka *C* (*kompilator gcc*) napisz program, który będzie pobierał z klawiatury znaki ASCII, do wystąpienia litery 'q' (lub 'Q'), a następnie zapisze je w postaci jednego łańcucha znakowego w pliku tekstowym *input.txt*. Znaki wprowadzane są niezależnie od siebie, jeden po drugim. Maksymalnie użytkownik może wprowadzić 20 znaków. Jeżeli wcześniej nie wystąpi litera 'q', program ma zapisać dotychczasowe znaki do pliku i przerwać swoje działanie.

Zadanie 2. Korzystając z języka *C* lub *C++* napisz program, który pobierze z klawiatury nazwę (lub ścieżkę) pliku tekstowego oraz pojedynczą literę, a następnie wyznaczy ile razy dana litera występuje w zawartości tego pliku. Dla ułatwienia można ograniczyć długość nazw (ścieżek) plików do 40 znaków.

Zadanie 3. Korzystając z języka *C++* (*kompilator g++*) i klas ***std::ofstream*** oraz ***std::ifstream*** napisz program, który odczyta pierwsze 20 znaków ze wszystkich wskazanych plików tekstowych, a następnie połączy odczytane znaki w jeden łańcuch i zapisze go w pliku *compilation.txt*. Nazwy (lub ścieżki) plików mają zostać przekazane do programu za pomocą argumentów funkcji *main()*.

4. Dodatek

4.1. Obsługa plików z poziomu konsoli systemowej

Wśród poleceń obsługiwanych przez konsolę systemu operacyjnego dostępny jest szeroki zakres komend, służących do przeprowadzania operacji na plikach. W tabeli 4. zestawiono podstawowe polecenia konsoli systemów Windows i Unix, w tym najważniejsze polecenia umożliwiające obsługę plików.

Tabela 4. Podstawowe polecenia konsoli systemów Windows i Unix

Windows	**Unix**	**Opis**
attrib	chmod	wyświetl atrybuty pliku
cd	cd	zmień katalog
cls	clear	wyczyść ekran
comp	diff	porównaj zawartość plików
copy	cp	kopiuj pliki
del	rm	usuń pliki
dir	ls	wylistuj zawartość katalogu
echo	echo	wyświetl tekst
fc	diff	porównaj pliki i wyświetl różnice
find	find	znajdź plik
mkdir	mkdir	utwórz katalog
move	mv	przenieś pliki
rd	rm -r	usuń katalog
rename	mv	zmień nazwę pliku
shutdown	shutdown now	zamknij system
taskkill	kill	zakończ proces
tasklist	ps	wylistuj uruchomione procesy
tree	ls -R	wylistuj zawartość katalogu w formie graficznej

Literatura

[1] *fopen, fopen_s*. URL: `https://en.cppreference.com/w/c/io/fopen`.

[2] S. Prata. *Język C. Szkoła programowania*. 6th ed. Helion, 2016. ISBN: 978-83-283-1470-2.

[3] S. Prata. *Język C++. Szkoła programowania*. 6th ed. Helion, 2013. ISBN: 978-83-246-4336-3.

[4] *std::basic_ofstream*. URL: `https://en.cppreference.com/w/cpp/io/basic_ofstream`.