

XVI. Obsługa plików

16.1. Biblioteka odpowiedzialna za obsługę plików

Pisząc nasze programy, prędzej czy później znajdzie potrzeba zapisywania danych na dysku. Z pomocą przychodzi tu biblioteka **fstream**, dzięki której uzyskujemy funkcje pozwalające nam zarówno zapisywać pliki jak i je odczytywać.

16.2. Typ zmiennej **fstream**

Zanim zaczniemy odczytywać, bądź zapisywać dane z/do pliku, musimy posiadać zmienną, dzięki której będziemy mogli wykonywać operacje na wybranym pliku. W tym celu utworzona została klasa **fstream**. Klasa ta jest umieszczona w przestrzeni nazw **std::**. Klasa ta udostępnia nam cały interfejs, dzięki któremu będziemy mogli obsłużyć dowolny plik znajdujący się na dysku lub innym nośniku danych.

```
std::fstream plik;
```

16.3. Otwieranie pliku

Zmienna, którą utworzyliśmy aktualnie nie wskazuje na żaden plik. Aby przypisać konkretny plik do zmiennej wywołujemy funkcję **open()**, której definicja wygląda następująco:

```
void open(const char* nazwa_pliku,ios_base::openmode tryb_otwarcia_pliku);
```

Pierwszy parametr funkcji (**nazwa_pliku**) określa ścieżkę dostępu i nazwę pliku do którego chcemy uzyskać dostęp. Drugi parametr funkcji, czyli **tryb_otwarcia_pliku** służy do poinformowania kompilatora w jakim trybie dany plik chcemy otworzyć. Lista dostępnych trybów wraz z opisami w poniższej tabeli.

Tryb	Opis trybu
ios::app	(append - dopisywanie danych do pliku) Ustawia wewnętrzny wskaźnik zapisu pliku na jego koniec. Plik otwarty w trybie tylko do zapisu. Dane mogą być zapisywane tylko i wyłącznie na końcu pliku.
ios::ate	(at end) Ustawia wewnętrzny wskaźnik pliku na jego koniec w chwili otwarcia pliku.
ios::binary	(binary) Informacja dla kompilatora, aby dane były traktowane jako strumień danych binarnych, a nie jako strumień danych tekstowych.
ios::in	(input - wejście/odczyt) Zezwolenie na odczytywanie danych z pliku.
ios::out	(output - wyjście/zapis) Zezwolenie na zapisywanie danych do pliku.
ios::trunc	(truncate) Zawartość pliku jest tracona, plik jest obcinany do 0 bajtów podczas otwierania.

Wszystkie wymienione tryby możemy łączyć ze sobą - oznacza to, że jeśli chcemy otrzymać plik do odczytu i zapisu wystarczy oddzielić je pojedynczym operatorem |.

```
std::fstream plik;  
plik.open("nazwa_pliku.txt",std::ios::in|std::ios::out);
```

16.3.1. Czy udało otworzyć się plik?

Zdecydowana większość kursów pomija ten bardzo ważny krok, jaki należy implementować we własnych kodach źródłowych - sprawdzanie czy plik został otwarty prawidłowo. Operacja jest bardzo prosta, jednak prawie zawsze zaniebywana nawet w książkach poświęconych programowaniu!

Po wykonaniu operacji otwarcia pliku, wewnątrz klasy ustawiane są odpowiednie flagi, które informują o tym, czy otrzymaliśmy dostęp do pliku czy też nie. Funkcje, jakie umożliwiają nam sprawdzenie tego stanu to `good()` oraz `is_open()`. Definicja tych funkcji wygląda następująco:

```
bool good();
bool is_open();
```

Obie funkcje zwrócą wartość `true`, jeśli uzyskano dostęp do pliku, w przeciwnym wypadku otrzymamy wartość `false`.

```
std::fstream plik;
plik.open("nazwa_pliku.txt",std::ios::in|std::ios::out);
if(plik.good() == true)
{
    std::cout<<"Uzyskano dostep do pliku!"<<std::endl;
    //tu operacje na pliku
}else std::cout<<"Dostep do pliku zostal zabroniony!"<<std::endl;
```

16.3.2. Kiedy nie uzyskamy dostępu do pliku

Próba odczytu:

- Plik nie istnieje na dysku;
- Nie posiadamy uprawnień odczytu do pliku.

Próba zapisu:

- Nie posiadamy uprawnień pozwalających nam modyfikować plik;
- Nie posiadamy uprawnień do katalogu w którym chcemy utworzyć plik;
- Nośnik, na którym chcemy dokonać zapisu jest tylko do odczytu.

16.4. Zamykanie pliku

Każdy plik należy zamykać po zakończeniu pracy z nim. Jeśli plik ma być używany tylko przez jednego użytkownika szkodliwość jest stosunkowo mała - klasa `fstream` sama zamknie plik przed usunięciem zmiennej z pamięci. Jeśli natomiast zapomnisz zamknąć plik, którego dane mają być współdzielone przez kilku użytkowników, automatycznie uniemożliwisz im dostęp do tego zasobu. Funkcja odpowiedzialna na zamykanie pliku nosi nazwę `close()`. Deklaracja wygląda następująco:

```
void close(void);
```

16.4.1. Przykład

Poniższy przykład pokazuje jak należy prawidłowo posługiwać się otwartym plikiem.

```
#include <fstream>
int main()
{
    std::fstream plik;
    plik.open("nazwa_pliku.txt",std::ios::in|std::ios::out);
    if(plik.good() == true)
    {
        //tu operacje na pliku (zapis/odczyt)
    }
}
```

```
plik.close();
}
return(0);
}
```

16.5. Odczytywanie danych z pliku

Jeśli uzyskamy już dostęp do pliku w trybie do odczytu, możemy rozpocząć odczytywanie danych z pliku. Język C++ oferuje więcej niż jedną metodę odczytu danych z pliku.

16.5.1. Pobieranie danych za pomocą strumienia

Pierwszą, a zarazem bardzo wygodną metodą odczytywania danych z pliku jest strumień. Ponieważ zapis jest analogiczny do strumienia `std::cin>>`, przedstawiam tylko formę zapisu.

```
nazwa_zmiennej_plikowej>>zmienna_do_ktorej_dane_maja_zostac_zapisane;
```

Co należy wiedzieć o strumieniu:

- Dane odczytywane za pomocą strumienia są zawsze traktowane jako tekst, niezależnie czy podczas otwierania użyliśmy trybu `ios::binary` czy nie.
- Strumień działa analogicznie do `std::cin>>`, co w konsekwencji oznacza, że za pomocą tej funkcji nie odczytamy żadnej informacji o białych znakach (tj. enter, tabulacja, spacja itp).

16.5.2. Pobieranie danych wierszami

Kolejną metodą na odczytanie danych, to użycie funkcji `getline()`. Funkcja ta została omówiona w rozdziale **XVIII. Biblioteka <string>**. Przykład:

```
std::fstream plik("nazwa_pliku.txt",std::ios::in);//zakładamy, że plik istnieje
std::string dane;
getline(plik,dane);//wczytanie CAŁEGO jednego wiersza danych
```

Istnieje również druga funkcja służąca do wczytywania danych wierszami, jednak wydaje się ona mniej wygodna w użyciu. Jest nią funkcja `getline()`, zaszyta wewnątrz klasy `fstream`.

```
istream& getline (char* odczytane_dane, streamsize ilosc_danych, char znak_konca_linii);
```

Parametry oznaczają kolejno:

- (`odczytane_dane`) wskaźnik zmiennej, do której mają zostać wczytane dane z pliku;
- (`ilosc_danych`) maksymalna ilość znaków jakie mogą zostać zapisane do zmiennej;
- (`znak_konca_linii`) parametr jest opcjonalny. Umożliwia zmianę znaku końca linii.

Przykład wykorzystania tej funkcji:

```
std::fstream plik("nazwa_pliku.txt",std::ios::in);//zakładamy, że plik istnieje
char dane[255];
plik.getline(dane,255);//wczytanie jednego wiersza danych (lub części wiersza jeśli się nie zmieści)
```

Co należy wiedzieć o obu funkcjach `getline()`:

- Dane odczytywane za pomocą funkcji `getline()` są zawsze traktowane jako tekst, niezależnie czy podczas otwierania użyliśmy trybu `ios::binary` czy nie.

16.5.3. Pobieranie danych blokami

Pobieranie danych blokami jest jedną z najszybszych metod na odczytywanie danych. Co więcej jest to bezpieczna metoda dla danych binarnych (pod warunkiem włączenia trybu `ios::binary`). Minusem tej metody jest niezbyt poręczna forma w jakiej otrzymujemy dane. Budowa tej funkcji wygląda następująco:

```
istream& read(char* bufor, streamsize rozmiar_bufora);
```

Pierwszym parametrem przekazywanym do funkcji jest wskaźnik do którego mają zostać wczytane dane. Drugi parametr określa rozmiar bufora. Pamiętaj, że bufor może nie być wypełniony do końca danymi. Aby sprawdzić ile bajtów danych zostało faktycznie wczytanych do bufora, należy posłużyć się tu funkcją `gcount()`. Przykład:

```
std::fstream plik("nazwa_pliku.txt",std::ios::in);//zakładamy, że plik istnieje
char bufor[1024];
plik.read (bufor,1024);//wczytuje tyle danych ile się zmieści do bufora
std::cout<<"Wczytano "<<plik.gcount()<<" bajtów do bufora"<<std::endl;
```

Co należy wiedzieć o funkcji `read()`:

- Dane odczytywane za pomocą funkcji `read()` są traktowane jako dane binarne, jeśli użyliśmy trybu `ios::binary`.

16.5.4. Inne metody na odczytywanie danych

C++ oferuje również inne metody odczytywania danych. Nie będą one jednak tu omówione ponieważ te, które zostały poruszone w tym rozdziale są wystarczające do pełnego wykorzystywania możliwości wczytywania danych z plików.

16.6. Zapisywanie danych do pliku

Zapisywanie danych do pliku jest równie proste jak ich odczytywanie. Po otwarciu pliku do zapisu możemy korzystać z kilku technik umożliwiających zapisywanie danych. Zanim jednak je poznasz musisz zdać sobie sprawę, że dane do pliku można albo dopisywać tylko i wyłącznie na końcu pliku albo **nadpisywać** dane jeśli nie jesteśmy na jego końcu. Nie można dopisywać tekstu pomiędzy istniejące dane jak to często robimy w edytorach tekstowych. Pamiętaj więc, jeśli chcesz otworzyć plik do zapisu zastanów się conajmniej dwa razy, bo tutaj błąd może kosztować nawet utratę całej zawartości pliku. Jeśli chcesz testować działanie funkcji służących do zapisu danych polecam utworzyć najpierw pusty plik i wpisać ręcznie do niego jakieś nieistotne dane lub pracować na kopii pliku, zawierającego ważne dane. W razie wykonania jakiegoś rażącego błędu będziesz mógł przywrócić szybko dane.

16.6.1. Zapisywanie danych za pomocą strumienia

Zapisywanie danych za pomocą strumienia jest analogicznym działaniem do `std::cout<<`. Jediną istotną różnicą, jaka ma tu miejsce to fakt, że wyjściem jest teraz plik, a nie konsola.

```
nazwa_zmiennej_plikowej<<zmienna_ktora_ma_zostac_zapisana_do_pliku;
```

Tak samo jak w przypadku odczytywania danych za pomocą strumienia, zapisywane dane tą techniką są zawsze traktowane jako tekst niezależnie od ustawienia trybu `ios::binary`. Każdorazowe zapisanie danych powoduje przesunięcie wskaźnika o tyle znaków ile zostało zapisanych do pliku.

16.6.2. Zapisywanie danych blokami

Gdy zapisywanie danych w postaci tekstu jest dla nas niewystarczające (a przy profesjonalnym podejściu do większości projektów tak właśnie jest) z pomocą przychodzi nam kolejna funkcja klasy **fstream** i jest to **write()**. Definicja tej funkcji wygląda następująco:

```
ostream& write(const char* bufor, streamsize ilosc_danych_do_zapisu );
```

Pierwszy parametr (**bufor**) to wskaźnik bufora, w którym znajdują się dane jakie chcemy zapisać do pliku. Drugim parametrem (**ilosc_danych_do_zapisu**) informujemy kompilator ile danych ma zostać zapisanych do pliku z bufora. Wraz z wykonaniem tej operacji wskaźnik wewnętrzny pliku przesuwają się do przodu o ilość bajtów zapisanych do pliku.

```
std::fstream plik("nazwa_pliku.txt",std::ios::out);//zakładamy, że nie wystąpił błąd (plik
otwarto/utworzono)
std::string napis;
getline(std::cin,napis);
plik.write (&napis[0],napis.length());//zapisuje dane poczynając od 0 indeksu
```

16.6.3. Zapisywanie danych w szczegółach

Jeśli napiszesz sobie program, który będzie zapisywał do pliku wczytywane wiersze z klawiatury aż do napotkania pustego wiersza pewnie zauważysz, że rozmiar pliku się nie zmienia zaraz po dopisaniu danych. Dzieje się tak dlatego, że klasa **fstream** ma wewnętrzny bufor, który ma na celu przyspieszenie operacji dyskowych. Każdorazowy dostęp do wybranego obszaru dysku wymaga bardzo dużego czasu w porównaniu do szybkości pamięci podręcznej. Dane zanim trafią na dysk są umieszczane najpierw w buforze, a następnie gdy bufor się zapełni zostają zapisywane na dysk. Dzięki takiemu podejściu do zapisywania danych w pliku proces jest dużo szybszy. Przykładowo, jeśli jednorazowe ustawienie głowicy dysku na określonej pozycji zajmuje np. 2ms, to zapisanie długiego zdania znak po znaku zajęłoby: 2ms*ilość_znaków czasu. Wbudowany system buforowania danych zamiast zapisywać tak często dane, zapisze je najpierw do bufora, a później wyśle je na dysk oszczędzając jednocześnie mnóstwo zasobów sprzętowych komputera. System ten jest zawsze sprawny niezależnie od tego czy skaczesz po pliku w różne miejsca, czy dopisujesz stale dane na jego końcu.

16.6.4. Kontrola bufora zapisu

Klasa **fstream** umożliwia nam 'kontrolowanie' wewnętrznego bufora zapisu. Cała ta kontrola sprowadza się do zmuszenia klasy **fstream**, aby zapisała całą obecną zawartość bufora na dysk bez względu na to czy jest on zapełniony czy nie. W tym celu utworzono funkcję **flush()**. Poniżej zamieszczam przykład demonstrujący użycie tej funkcji.

```
#include <fstream>
using namespace std;
int main ()
{
    fstream plik("plik.txt",ios::out);
    if(plik.good())
    {
        for(int i=1;i<=100;i++)
        {
            plik<<i<<" ";
            plik.flush();
        }
        plik.close();
    }
    return(0);
}
```

Pamiętaj jednak, że takie zapisywanie danych jak tu zostało zaprezentowane nie jest wydajne. Funkcja **flush()** pomimo iż wydaje się w obecnym świetle dla Ciebie bezużyteczna znajduje ona swoje praktyczne zastosowanie chociażby w serwerach profesjonalnych baz danych.

16.7. Poruszanie się po pliku z danymi

Do tej pory odczytywaliśmy (zapisywaliśmy) dane z (do) pliku zawsze od tego miejsca na którym skończyliśmy operację odczytu (zapisu) ostatnim razem. Taka forma odczytu i zapisu danych jest bardzo wygodna, jednak czasem zachodzi potrzeba poruszania się po pliku w bardziej nietypowy sposób. Z pomocą przychodzą tu funkcje **seekg()** i **seekp()**. Obie funkcje służą do ustawiania nowej pozycji wewnętrznego wskaźnika pliku. Jest jednak między nimi jedna zasadnicza różnica:

- **seekg()** ustawia wewnętrzny wskaźnik pliku dla funkcji odczytujących dane;
- **seekp()** ustawia wewnętrzny wskaźnik pliku dla funkcji zapisujących dane.

Parametry obu tych funkcji są analogiczne:

```
istream& seekg (streamoff offset, ios_base::seekdir kierunek);  
ostream& seekp (streamoff offset, ios_base::seekdir kierunek);
```

Pierwszy parametr (**offset**) to przesunięcie, które informuje o ile bajtów ma zostać przesunięty wewnętrzny wskaźnik pliku. Drugi parametr (**kierunek**) jest opcjonalny i informuje klasę **fstream** względem czego ma zostać dokonane przesunięcie wskaźnika. Domyślną wartością, jaka jest przyjmowana za zmienną **kierunek**, to **ios_base::beg**. Kierunki jakie mamy do wyboru to:

Kierunek	Opis
ios_base::beg	Przesunięcie względem początku pliku (domyślne)
ios_base::cur	Przesunięcie względem aktualnej pozycji
ios_base::end	Przesunięcie względem końca pliku

16.7.1. Odczytywanie aktualnej pozycji wewnętrznego wskaźnika pliku

Jeśli będziemy mieli potrzebę odczytania aktualnej pozycji wewnętrznego wskaźnika pliku, możemy to zrobić za pomocą funkcji **tellg()** i **tellp()**. Definicja obu funkcji wygląda następująco:

```
streampos tellg();  
streampos tellp();
```

Obie funkcje wyglądają tak samo, różnią się jednak działaniem.

- Funkcja **tellg()** zwraca aktualną pozycję wewnętrznego wskaźnika pliku od której będzie następowało wczytywanie danych z pliku.
- Funkcja **tellp()** zwraca aktualną pozycję wewnętrznego wskaźnika pliku od której będzie następowało zapisywanie danych do pliku.

16.7.2. Gdy wyjdziemy poza zasięg pliku

Aby sprawdzić, czy skok na nową pozycję zakończył się sukcesem możemy dokonać tego na dwa sposoby:

- Sprawdzić aktualną pozycję pliku i porównać z tą, którą chcieliśmy otrzymać;
- Wywołać funkcję **fail()**, należącą do klasy **fstream**.

Pierwsza metoda jest logiczna, druga wymaga krótkiego omówienia. Definicja funkcji **fail()** wygląda następująco:

```
bool fail();
```

Jeśli wystąpi błąd podczas wykonywania skoku (i nie tylko skoku) funkcja ta zwróci wartość **true** informując nas, że ostatnia operacja na pliku nie powiodła się. Przykładowo:

```
std::fstream plik("plik.txt",std::ios::in);//zakładamy, że plik udało się otworzyć
plik.seekg(+2,std::ios_base::end);//skok do przodu o 2 względem końca pliku
if(plik.fail()) std::cout<<"Error! Nie udało sie przesunac wewnetrznego wskaźnika pliku"<<std::endl;
```

16.8. Pozostałe funkcje, wykorzystywane podczas pracy z plikami

Ostatnią funkcją, jaką chciałbym omówić jest **eof()**. Funkcja ta służy do sprawdzania, czy wskaźnik pliku znajduje się na końcu pliku. Definicja funkcji:

```
bool eof();
```

Funkcja zwróci wartość **true** wtedy, gdy nie będzie już w pliku więcej danych do odczytu. Dzięki tej funkcji możemy w bardzo łatwy sposób odczytać zawartość całego pliku. Poniższy przykład wyświetli zawartość całego pliku na ekran konsoli.

```
#include <iostream>
#include <fstream>
#include <conio.h>
using namespace std;
int main()
{
    fstream plik;
    plik.open("dane.txt",ios::in);
    if(plik.good())
    {
        string napis;
        cout<<"Zawartosc pliku:"<<endl;
        while(!plik.eof())
        {
            getline(plik,napis);
            cout<<napis<<endl;
        }
        plik.close();
    }else cout<<"Error! Nie udalo otworzyc sie pliku!"<<endl;
    getch();
    return(0);
}
```

16.8.1. Dokumentacja

Dobrym źródłem dostępnych funkcji w klasie **fstream** jest dokumentacja. Dokumentację dla klasy **fstream** znajdziesz pod adresem: <http://www.cplusplus.com/reference/iostream/fstream/>