Laboratorium Informatyki II
Ćwiczenie 14

Obsługa plików

Obsługa plików w języku `C++` realizowana jest w sposób analogiczny jak w języku `C`. Proces odbywa się w układzie zamkniętego cyklu dla utworzonego obiektu klasy `FILE`:

1. otwarcie dostępu do pliku w wybranym trybie
2. zapis lub odczyt danych
3. zamknięcie dostępu do pliku

Sam dostęp i związany z nim format zapisu danych może być **tekstowy** lub **binarny**. Obsługę strumienia wejściowego i wyjściowego do i z pliku realizowana jest za pomocą klasy i narzędzi dostarczonych przez bibliotekę `<fstream>`. Obiekt klasy `<fstream>` umożliwia nam zdefiniowanie nowego strumienia wejścia/wyjścia do pliku utworzonego na dysku. W klasie zdefiniowano kilka konstruktorów, a podstawowe pokazano poniżej:

- konstruktor domyślny tworzący obiekt bez dowiązania do pliku. Dowiązanie tworzy się za pomocą metody .open() i odpowiednich trybów dostępu

```
std::fstream plik_1;
plik_1.open("file.txt", std::ios_base::in | std::ios_base::app);
```

- konstruktor z dowiązaniem i określeniem trybu dostępu (określanym na podstawie wybranego trybu dostępu)

```
std::fstream plik_2("file.txt", std::ios_base::in | std::ios_base::app);
```

- za pośrednictwem deskryptora pliku określającego sposób otwarcia dostępu do pliku (tryb określany na podstawie dostępnych flag)

```
int deskryptor = open("file.txt", O_RDWR);
std::fstream plik_3(deskryptor);
```

Dostęp do pliku

Sposób dostępu do pliku może być określony przez:

1. tryb określony w metodzie `open()`
2. flagę zdefiniowaną w obiekcie klasy `filebuf`

Listę trybów dostępu zestawiono poniżej. Mogą one być łączone za pomocą operatora bitowego `OR` - | w celu zdefiniowania wielo-trybowego dowiązania.

- `std::ios::in`: Otwiera plik do odczytu. Pozwala na operacje odczytu danych z pliku.

- `std::ios::out`: Otwiera plik do zapisu. Pozwala na operacje zapisu danych do pliku. Jeśli plik już istnieje, zostanie nadpisany. Jeśli plik nie istnieje, zostanie utworzony. W pewnych sytuacjach plik może nie zostać utworzony, w takim przypadku należy dodać tryb `std:ios::trunc`.

- `std::ios::app`: Otwiera plik w trybie dodawania (`append`). Pozwala na zapisywanie danych na końcu pliku, bez nadpisywania już istniejącej zawartości.

- `std::ios::binary`: Otwiera plik w trybie binarnym. Pozwala na odczyt i zapis danych w formacie binarnym, z uwzględnieniem specyfikacji binarnej (np. bez konwersji znaków nowej linii).

- `std::ios::ate`: Otwiera plik i ustawia pozycję odczytu/zapisu na koniec pliku. Dzięki temu można natychmiastowo odczytywać/zapisywać dane na końcu pliku bez potrzeby przesuwania pozycji ręcznie.

- `std::ios::trunc`: Jeśli plik już istnieje, usuwa jego zawartość przy otwarciu w trybie zapisu (`std::ios::out`). Jeśli plik nie istnieje, zostanie utworzony.

Ustawienie trybu dostępu za pomocą flag obiektu `filebuf` realizowane jest na analogicznych zasadach. Dostępne flagi zestawiono w tabeli poniżej.

- `O_RDONLY`: Otwiera plik tylko do odczytu (**read-only**).

- `O_WRONLY`: Otwiera plik tylko do zapisu (**write-only**).

- `O_RDWR`: Otwiera plik do odczytu i zapisu (**read-write**).

- `O_CREAT`: Tworzy plik, jeśli nie istnieje.

- `O_TRUNC`: Wytnij (*obetnij*) plik, jeśli już istnieje.

- `O_APPEND`: Przesuń pozycję w pliku na koniec przed każdym zapisem.

- `O_EXCL`: Błąd, jeśli plik już istnieje w trybie tworzenia (`O_CREAT`).

- `O_NONBLOCK`: Otwiera plik w trybie nieblokującym (*asynchronicznym*).

- `O_SYNC`: Zapisywanie do pliku jest dokonywane synchronicznie (*blokująco*). Ponadto, istnieją również flagi rozszerzone, które są zależne od systemu operacyjnego i implementacji. Przykłady to:

- `O_BINARY`: Otwiera plik w trybie binarnym.

- `O_TEXT`: Otwiera plik w trybie tekstowym.

Kontrola stanu strumienia

Kontrola stanu strumienia w klasie `<fstream>` może być realizowana przez kilka metod na podobnych zasadach jak w przypadku standardowych wejścia i wyjścia (klawiatura i ekran). Wybrane metody zestawiono poniżej:

- metoda otwarcia dowiązania do pliku `open()` w przypadku poprawnego dowiązania do wskazanego pliku zwraca wartość `true`
- metoda `.is_open()` zwraca wartość `true` jeżeli dostęp do pliku jest otwarty i zapis lub odczyt może być realizowany
- metody testowania stanu strumienia `.good()` i `.fail()` pozwalają na wykrycie błędów wywołanych przez metody i funkcje zapisu lub odczytu danych z pliku
- wykrycie końca pliku w trakcie odczytu metodą `.eof()` zwracającą wartość `true` gdy osiągnięty zostanie koniec pliku.
- czyszczenie bitu błędu podobnie jak w przypadku strumieni standardowych realizuje metoda `.clear()`. Dojście z odczytem danych do końca pliku także wywołuje błąd strumienia i konieczność wyczyszczenia bitu błędu metodą `.clear()`.

Nawigowanie po pliku

Określanie pozycji kursora i jego przesuwanie jest istotnym zagadnieniem szczególnie w sytuacji równocześnie realizowanego dostępu do pliku w trybie zapisu i odczytu. Metody zestawiono poniżej:

- Określanie pozycji kursor w pliku:
  - `.tellp()` - zwraca pozycję w trybie zapisu od początku pliku
  - `.tellg()` - zwraca pozycję w trybie odczytu, dla pozycji końcowej pliku zwracana jest wartość `-1`
- Przemieszczanie kursora w pliku:
  - w trybie zapisu `std::ios_base::out` określanie pozycji za pomocą `.seekp(number,odniesienie);`, gdzie `number` określa przemieszczenie względem punktu odniesienia (+ w prawo, a - w lewo) określanym za pomocą jednej z trzech flag:
    - `std::ios_base::beg` - względem początku pliku (domyślny)
    - `std::ios_base::end` - względem końca pliku
    - `std::ios_base::cur` - względem bieżącej pozycji
  - w trybie odczytu `.seekg(number, odniesienie);` - analogicznie jak w trybie zapisu

Tryb tekstowy

W trybie tekstowym dane każdego typu konwertowane są do typu znakowego. Przetwarzanie danych może być realizowane na dwa sposoby:

- przekierowanie danych za pomocą operatorów przekierowania << i >> :

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

int main()
{
    std::fstream plik;
    std::string dane;
    plik.open("test.txt",std::ios_base::out | std::ios_base::trunc
                        | std::ios_base::in);
    plik << "Testowy zestaw znaków" << std::endl;
    plik << "x=" << 20 << std::endl;
    plik.seekp(0,std::ios_base::beg);
    while(!plik.eof()){
        plik >> dane;
        std::cout << dane;
    }
    plik.close();
}
```

Należy pamiętać że przy odczycie danych tracone są białe znaki (spacje, tabulacje, przejście do nowego wiersza).

- za pomocą metod i funkcji:
  - `.put()` //zapis danych typu **char**
  - `.get()` //odczyt danych typu **char**
  - `.getline()` //odczyt danych z pliku do tablicy **char**[]
  - `getline()` //odczyt danych z pliku do obiektu typu **string**
  - `.write()` //zapis danych typu **char** lub **char**[]
  - `.read()` //odczyt danych z pliku do zmiennej typu **char** lub **char**[]

Metoda i funkcja `getline()` nie odczytuje znaku przejścia do nowego wiersza.

```
#include <iostream>
#include <fstream>
int main()
{
    std::fstream file("plik.txt", std::ios_base::in | std::ios_base::out
                                | std::ios_base::trunc);
    std::string tekst = "Testowy ciąg tekstowy.";
    //TEST ZAPISU DANYCH
    for(int i=90;i<120;i++)
        file.put(i);
    file << std::endl;
    file.write(tekst.c_str(),tekst.size());
    file << std::endl;
    //TEST ODCZYTU DANYCH
    file.seekg(0,std::ios_base::beg);
    std::cout << "Test .put() i .get(): ";
    while(!file.eof()){
```

```
        char c = file.get();
        std::cout << c;
    }
    std::cout << std::endl;
    file.clear();
    file.seekg(0,std::ios_base::beg);
    std::cout << "Test .getline(): ";
    while(!file.eof()){
        char dane[256] = {'\0'};
        file.getline(dane,255);
        std::cout << dane;
        }
    std::cout << std::endl;
    file.clear();
    file.seekg(0,std::ios_base::beg);
    std::cout << "Test getline(): ";
    while(!file.eof()){
        std::getline(file,tekst);
        std::cout << tekst;
        }
    std::cout << std::endl;
    file.clear();
    file.seekg(0,std::ios_base::beg);
    std::cout << "Test .read(): ";
    while(!file.eof()){
        char dane[256] = {'\0'};
        file.read(dane,255);
        std::cout << dane;
    }
    std::cout << std::endl;
    file.clear();
    file.close();
    return 0;
}
```

Przy odczycie danych liczbowych w trybie tekstowym przydatne mogą być funkcje konwertujące liczby zapisane w tablicy znakowej do wybranego typu liczbowego `int` lub `float`:

- `atoi(char []);` - konwersja danych zapisanych w tablicy char do typu `int`
- `atof(char []);` - konwersja danych zapisanych w tablicy char do typu `float`

Tryb binarny

Tryb binarny dedykowany jest do zapisu danych liczbowych poprzez ich konwersję do postaci bitowej. Główną zaletą tego trybu jest szybkość realizacji zapisu i odczytu, brak konieczności konwersji z i do typu znakowego (jak to jest w trybie tekstowym), oraz możliwość zapisu tablic liczbowych. Do odczytu danych w trybie binarnym konieczna jest znajomość algorytmu ich zapisu.

Zapis danych w trybie binarnym możliwy jest po otwarciu dostępu do pliku w trybie binarnym. Sama procedura realizowana jest za pomocą dwóch metod:

- `.write()` - zapis danych
- `.read()` - odczyt danych

Obydwie funkcje do prawidłowego zapisania i odczytania danych z pliku binarnego wymagają zastosowania identycznie skonfigurowanych atrybutów. Pierwszy to rzutowana na typ `char` tablica danych, natomiast drugim jest rozmiar w bajtach zapisywanych danych.

```
#include <iostream>
#include <fstream>
#include <random>
void zapis(int,int);
void wczytanie();
void drukuj(int,int,float**);
int main()
{
    int row,col;
    std::cout << "Podaj rozmiar tablicy:" << std::endl;
    std::cout << "Liczba wierszy: "; std::cin >> row;
    std::cout << "Liczba kolumn: "; std::cin >> col;
    zapis(row,col);
    wczytanie();
}
void zapis(int row,int col)
{
    float **dane;
    std::random_device genrator;
    std::default_random_engine silnik(genrator());
    std::uniform_real_distribution<float> rozklad(0,100);
    dane = new float*[row];
    for(int i=0;i<row;i++){
        dane[i] = new float[col];
        for(int j=0;j<col;j++)
            dane[i][j] = rozklad(silnik);
    }
    std::fstream plik;
    plik.open("dane.txt",std::ios_base::out | std::ios_base::binary);
    if(plik.is_open())
    {
        plik.write(reinterpret_cast<char*>(&row),sizeof(row));
        plik.write(reinterpret_cast<char*>(&col),sizeof(col));
        for(int i=0;i<row;i++)
            plik.write(reinterpret_cast<char*>(dane[i]),col*sizeof(float));
        plik.close();
    }
    drukuj(row,col,dane);
}
void wczytanie()
{
    float **dane;
    int row=0, col=0;
    std::fstream plik;
```

```
    plik.open("dane.txt",std::ios_base::in | std::ios_base::binary);
    if(plik.is_open())
    {
        plik.read((char*)&row,sizeof(row));
        plik.read((char*)&col,sizeof(col));
        dane = new float*[row];
        for(int i=0;i<row;i++){
            dane[i] = new float[col];
            plik.read((char*)dane[i],col*sizeof(float));
        }
        plik.close();
    }
    drukuj(row,col,dane);
}
void drukuj(int row,int col,float **dane)
{
    std::cout << "TABLICA:" << std::endl;
    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            std::cout.width(3);
            std::cout.precision(3);
            std::cout << dane[i][j];
        }
        std::cout << std::endl;
    }
    std::cout << std::endl;
    for(int i=0;i<row;i++)
        delete [] dane[i];
    delete [] dane;
}
```

W powyższym przykładzie w funkcji `zapis()` w pierwszej kolejności zapisywane są informacje o rozmiarze tablicy poprzez zapisanie pojedynczej wartości typu `int`. Podobnie w funkcji `wczytywanie()` najpierw odczytywane są te wartości. W funkcjach pokazano dwa sposoby rzutowania zmiennej liczbowej na typ `char`, w funkcji `zapis()` realizowane jest to zgodnie z zalecanym standardem, ale obie formy są poprawne. W dalszym ciągu funkcji zapisywane są dane tablicy 2D, realizowane jest to wiersz po wierszu. W przypadku tablicy dynamicznej rozmiar bloku pamięci do zapisania określa się za pomocą iloczynu liczby komórek i rozmiaru pojedynczej komórki tablicy, w przypadku zapisu danych z tablicy statycznej można wykorzystać funkcję `sizeof()` do określenia rozmiaru w bajtach całej tablicy `sizeof(tab)`.

Zadania

W oparciu o informacje wykładowe i treść niniejszej instrukcji napisać program będący rozwinięciem opracowanej na wcześniejszych zajęciach aplikacji przetwarzającej zestawy wygenerowanych losowo liczb. Dodać do zaprojektowanych klas metody realizujące zapis i odczyt danych w trybie tekstowym i binarnym.