Temat: Operacje na plikach -cd.

1. Odczytywanie danych z pliku- Pobieranie danych wierszami

Kolejną metodą na odczytanie danych, to użycie funkcji **getline()**. **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 sie 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.

2. Zapisywanie danych do pliku

Po otwarciu pliku do zapisu możemy korzystać z kilku technik umożliwiających zapisywanie danych. 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.

a) Zapisywanie danych za pomocą strumienia

Zapisywanie danych za pomocą strumienia jest analogicznym działaniem do **std::cout**<<. Jedyną 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.

b) Zapisywanie danych blokami

Gdy zapisywanie danych w postaci tekstu jest dla nas niewystarczające kolejną funkcją klasy **fstream** jest **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 przesuwa 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
```

c) 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 przyśpieszenie 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.

d) 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 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 );
}
```

Takie zapisywanie danych jak tu zostało zaprezentowane nie jest wydajne. Funkcja **flush()**znajduje ona swoje praktyczne zastosowanie chociażby w serwerach profesjonalnych baz danych.

Ćwiczenie 1. Napisz program, który pobierze z pliku ***promienie.txt*** promienie dwóch kół. Twoim zadaniem jest stworzenie tego pliku ręcznie (z poziomu systemu operacyjnego), następnie otwarcie tego pliku za pomocą C++ oraz wyznaczenie pól kół i zapisanie wyników posortowanych rosnąco (najpierw mniejsze pole, następnie większe) do pliku ***wynik.txt.***

Ćwiczenie 2. Napisz program, który pobierze z pliku ***dane.txt*** trzy liczby całkowite(podstawa1, podstawa2, wysokość) (plik dane.txt tworzymy ręcznie z poziomu systemu operacyjnego) wyliczy pole trapezu i wynik zapisze w pliku ***pole.txt.***

Ćwiczenie 3. Stwórz plik ***dane.txt*** z poziomu systemu operacyjnego i wpisz do niego 5 liczb całkowitych. Otwórz ten plik z poziomu C++, pobierz wszystkie liczby i do pliku ***wynik.txt*** zapisz tylko te, których cyfra jedności kończy się na **0, 3, 8** lub **9.**

Ćwiczenie 4. W pliku dane.txt znajduje się ciąg liczb całkowitych (plik dane.txt tworzymy ręcznie z poziomu systemu operacyjnego. Napisz program, który wyznaczy sumę cyfr każdej z liczb i zapisze do pliku ***wynik.txt***.