



Programowanie obiektowe

Paweł Mikołajczak, 2019

8. Operacje plikowe

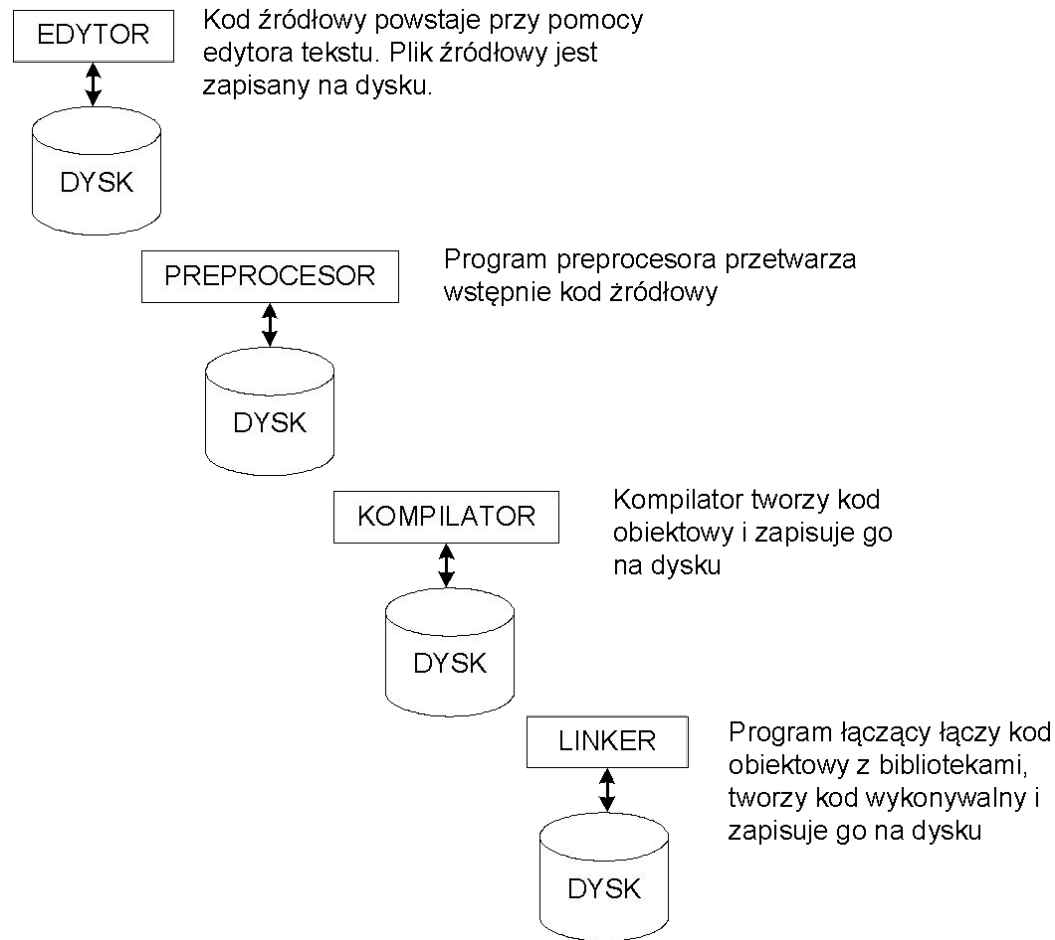
PWSZ Chelm

Operacje plikowe

Wykład 8

Wiele języków programowania ma wbudowaną obsługę wejścia – wyjścia. Języki C i C++ nie mają takich mechanizmów. Na początku rozwoju języka C obsługę wejścia – wyjścia pozostawiono twórcom i producentom tych kompilatorów. Później ustalił się nieformalny standard, który był przestrzegany przez wielu producentów. Producenci zazwyczaj realizowali biblioteki wejścia – wyjścia biorąc za wzór biblioteki opracowane dla systemu UNIX. W języku C++ dalej mamy implementowaną bibliotekę wejścia-wyjścia języka C (stdio.h) ale opracowano także nową bibliotekę w oparciu o nowe mechanizmy wejścia – wyjścia (iostream oraz fstream). W końcu komitet standaryzacji języka C++ organizacji ANSI/ISO podjął decyzję o opracowaniu standardu. Producenci kompilatorów języka C++ muszą przestrzegać tych standardów (co nie przeszkadza im nadmiernie umieszczać w tych bibliotekach swoich, dość specyficznych „wynalazków”).

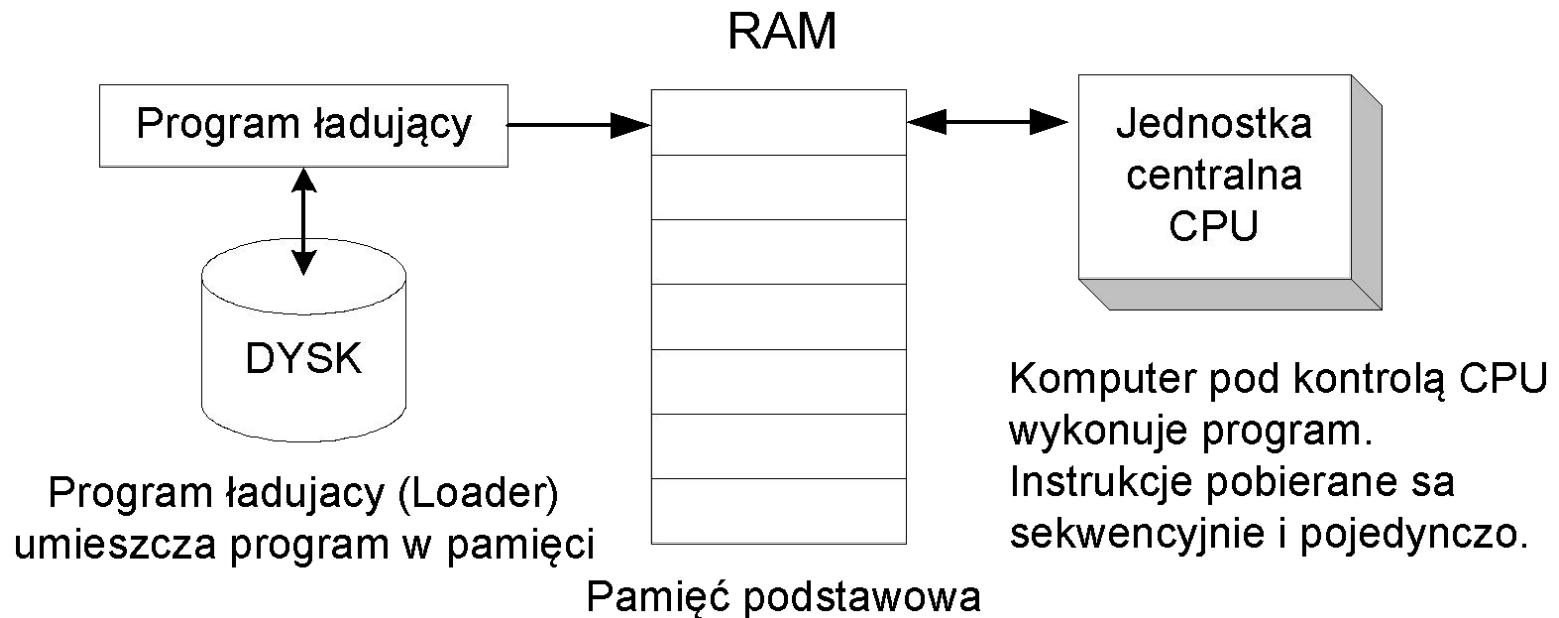
Typowe środowisko C/C++



Wykład 8

Zanim na ekranie monitora otrzymany zostanie wynik działania programu komputerowego, program w C/C++ standardowo przejdzie przez sześć faz:

- edycja
- przetwarzanie wstępne
- kompilacja
- łączenie
- załadowanie i wykonanie



Wykład 8

- Z punktu widzenia programisty ważne jest jak język C/C++ obsługuje pliki.
- Najprostsza definicja mówi, że plik (ang. file) jest ciągiem bajtów, z których każdy może być oddzielnie odczytany.
- Z punktu widzenia architektury komputera, plik jest wydzielonym obszarem pamięci posiadającym nazwę.
- Dla systemu operacyjnego plik jest dość skomplikowaną strukturą – np. plik może być przechowywany w kilku oddzielnych fragmentach.
- Plik jest dostępny za pośrednictwem wskaźnika do struktury, który jest zdefiniowany w standardowym pliku nagłówkowym **<stdio.h>** jako FILE. Struktura zawiera składowe, które opisują aktualny stan pliku. Abstrakcyjnie możemy traktować plik jak ciąg znaków, które są przetwarzane sekwencyjnie.
- System otwiera trzy standardowe pliki (definiuje trzy wskaźniki plikowe):

Nazwa pliku	Wskaźnik pliku podłączenie	
Standardowe wejście	stdin	klawiatura
Standardowe wyjście	stdout	ekran
Standardowe wyjście do błędów	stderr	ekran

Wykład 8

Większość programów czyta i zapisuje dane w pamięci dyskowej. Dyskowe operacje wejścia/wyjścia są wykonywane na plikach. Program w języku C/C++ może odczytywać i zapisywać pliki na różne sposoby.

Rozróżniamy standardowe wejście/wyjście (zwane także wysokiego poziomu lub strumieniowym) oraz systemowe wejście/wyjście (zwane także niskopoziomym).

Standardowe wejście/wyjście wysokiego poziomu (ang. standard high-level I/O) wykorzystuje funkcje biblioteczne i definicje znajdujące się w pliku **<stdio.h>**.

Systemowe wejście/wyjście (ang. low-level I/O) korzysta z podstawowych usług udostępnianych przez system operacyjny.

W języku C++ operujemy pojęciem strumienia bajtów. Bajty analogicznie jak woda w strumieniu przepływają z jednego miejsca w inne. Podczas operacji wejścia program pobiera bajty ze strumienia wejściowego, a podczas operacji wyjścia – program wstawia bajty do strumienia wyjściowego. Technicznie bajty są zbiorem bitów, mówimy zatem o przepływie bitów.

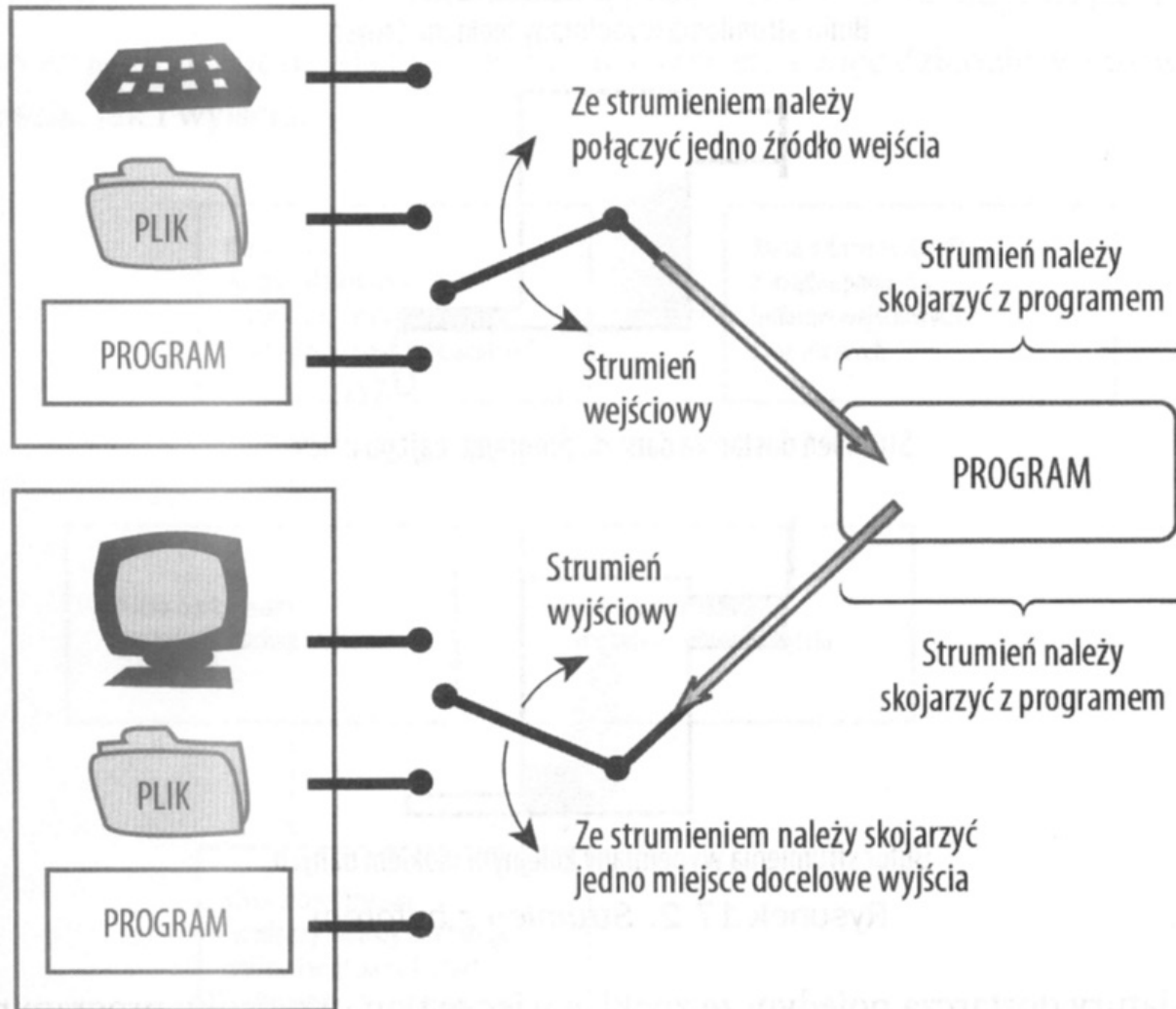
Obsługa wejścia składa się z dwóch etapów:

- skojarzenie strumienia z wejściem do programu
- połączenia strumienia z plikiem

Technicznie program analizuje strumień bajtów, nie musi wiedzieć skąd one pochodzą (klawiatura jest tak samo traktowana jak dysk twardy)

Wykład 8

C++
We-Wy



Dane w strumieniu mogą być umieszczane jako pojedyncze bajty albo grupy bajtów. Przesyłanie paczki bajtów jest zazwyczaj wydajniejszym procesem. W takim przypadku wykorzystujemy bufor.

Bufor jest blokiem pamięci, w którym umieszczamy tymczasowo paczkę bajtów. Podczas operacji wejścia-wyjścia bufor kolejno jest napełniany i opróżniany.

Napędy dyskowe wysyłają dane najczęściej w grupach po 512 bajtów.

Program może wysyłać dane na dysk jako pojedyncze bajty (np. znak po znaku) ale jest to proces bardzo wolny. Wykorzystanie bufora pozwala na znaczne przyspieszenie takiej operacji. Pamiętajmy także, że odczyt bajtów z bufora (pamięć operacyjna) jest znacznie szybszy niż odczyt danych z dysku.

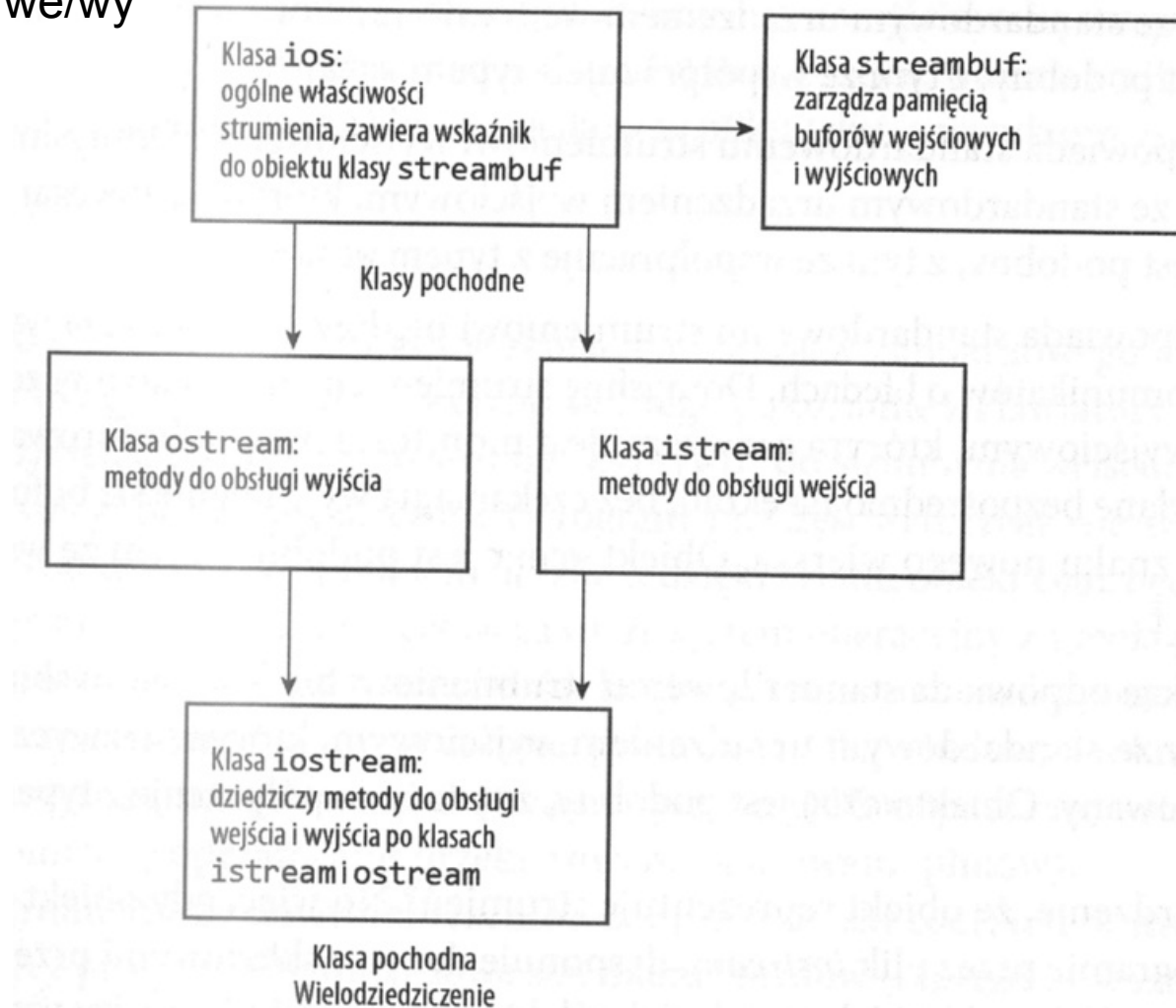
W języku C++ bufor wejściowy zwykle jest opróżniany po naciśnięciu klawisza ENTER.

Przy wyświetlaniu danych wyjściowych program w języku C++ bufor wyjściowy jest opróżniany po wysłaniu znaku nowego wiersza.

Praca ze strumieniami danych i buforami jest skomplikowana, na szczęście opracowana biblioteka standardowa dysponuje doskonałymi metodami (klasy), które w znacznym stopniu wyręczają programistę.

Wykład 8

Wybrane klasy we/wy



Mamy trzy rodzaje strumieni:

- wejściowy (obiekt klasy ifstream)
- wyjściowy (obiekt klasy ofstream)
- wejściowo-wyjściowy (obiekt klasy fstream)

Aby nawiązać komunikację z plikiem musimy podać nazwę pliku oraz tryb pracy.

W klasie ios mamy zdefiniowane następujące tryby (mody):

- | | |
|-------------|--|
| ios::app | zapis danych realizowany na końcu pliku |
| ios::ate | szukany jest koniec pliku |
| ios::binary | plik otwarty w trybie binarnym |
| ios::in | plik traktowany jest jako wejście |
| ios::out | plik traktowany jest jako wyjście |
| ios::trunc | niszczy istniejący plik, długość skracana jest do zera |

Zawsze należy sprawdzić czy nawiązanie komunikacji się udało.

Zapisywanie danych w pliku

Pracę z plikiem rozpoczyna się od jego otwarcia. Gdy plik jest otwarty, dalsze czynności wykonuje się tak samo, jak przy użyciu instrukcji `cin` i `cout`, tzn. za pomocą operatorów `<<` i `>>`. Kiedy jest mowa o komunikacji programu z obiektami zewnętrznymi, używa się pojęcia strumienia. W tym miejscu interesować nas będą strumienie plikowe, czyli strumienie umożliwiające zapis i odczyt plików.

Nagłówek **`fstream`**.

Wszystkie narzędzia niezbędne do pracy z plikami znajdują się w nagłówku `fstream`, który dołączamy na początku programu za pomocą dyrektywy `#include <fstream>`.

Pamiętamy, że `iostream` zawiera narzędzia pozwalające wysyłać i pobierać dane z konsoli. Nazwa **`iostream`** jest skrótem od angielskich słów **`input/output stream`** oznaczających strumień wejścia i wyjścia. Natomiast nazwa **`fstream`** pochodzi od angielskich słów **`file stream`** oznaczających strumień plikowy.

Otwieranie pliku do zapisu.

Faktycznie strumienie są obiektami. Przypomnę, że język C++ jest językiem obiektowym i strumienie są po prostu jednymi z jego wielu obiektów.

Strumienie plikowe definiuje się dokładnie tak samo, jak zwykłe zmienne. Po prostu definiujemy zmienną typu **ofstream** o wartości będącej **ścieżką do pliku**, który chcemy otworzyć.

Na początku zapiszemy w pliku informacje o ilości owoców w naszym sklepie.

Użyjemy operatora << do wprowadzenia danych do strumienia.

Oczywiście gdy chcemy zapisać dane do pliku, musimy podać nazwę tego pliku.

Gdy plik istnieje, dane są nadpisane.

Gdy pliku nie ma we wskazanym miejscu, zostanie on utworzony.

W moim przykładzie katalog i nazwa pliku mają postać:

```
ofstream wy("C:/testy/t_1.txt");
```

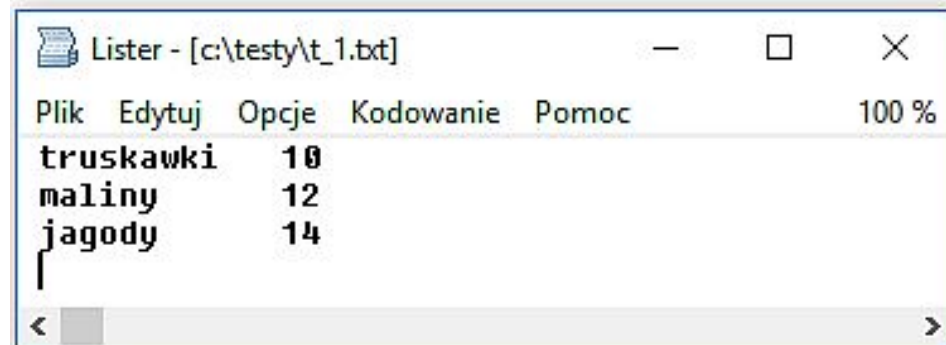
Wykład 8

```
//pliki
#include <bits/stdc++.h>
using namespace std;

int main()
{ ofstream wy("C:/testy/t_1.txt");

  if(!wy.is_open()) {
    cout <<"plik nie zostal otwarty \n";
    return 1;
  }
  wy <<"truskawki  " << 10<<endl;
  wy <<"maliny     " << 12<<endl;
  wy <<"jagody      " << 14<<endl;
  wy.close();    //zamkniecie pliku
}
```

Wynik zapisany w pliku,
Odczyt informacji.



Otwieranie pliku do odczytu

Zasada działania jest dokładnie taka sama, jak przy zapisywaniu plików, tylko zamiast strumienia **ofstream** będziemy używać strumienia **ifstream**.

Plik można odczytać na trzy sposoby:

- Linijka po linijce przy użyciu funkcji `getline()`.
- Słowo po słowie przy użyciu operatora `>>`.
- Znak po znaku przy użyciu funkcji `get()`.

Wykład 8

```
//pliki
#include <bits/stdc++.h>
using namespace std;

int main()
{ ifstream we("C:/testy/t_1.txt");
  string owoce;
  int kg;
  if(!we) {
    cout <<"plik nie zostal otwarty \n";
    return 1;
  }
  we >> owoce >>kg;
  cout <<owoce<<" "<<kg<<endl;
  we >> owoce >>kg;
  cout <<owoce<<" "<<kg<<endl;
  we >> owoce >>kg;
  cout <<owoce<<" "<<kg<<endl;
  we.close();
}
```

Wynik:

truskawki 10
maliny 12
jagody 14

Program przeczytał dane z pliku
i wydrukował je na ekranie monitora

W pokazanych prostych przykładach nazwy plików były wprowadzane bezpośrednio do funkcji `open()` jako jej argument:

```
ifstream we("C:/testy/t_1.txt");
```

Nie jest to przesadnie wygodny sposób obsługi plików.

Jasne jest, że wygodnie jest mieć możliwość wczytywania z klawiatury nazwy pliku.

Dzięki takiej technice, program będzie bardziej uniwersalny – mamy możliwość obsługi wielu różnych plików.

Wiemy, że nazwa pliku jest łańcuchem znakowym, dzięki temu jest dość łatwa do wprowadzenia z klawiatury:

```
string nazwa_pliku;  
ofstream wy;  
cout << "podaj nazwe pliku :";  
cin >> nazwa_pliku;  
wy.open(nazwa_pliku);
```

Wykład 8

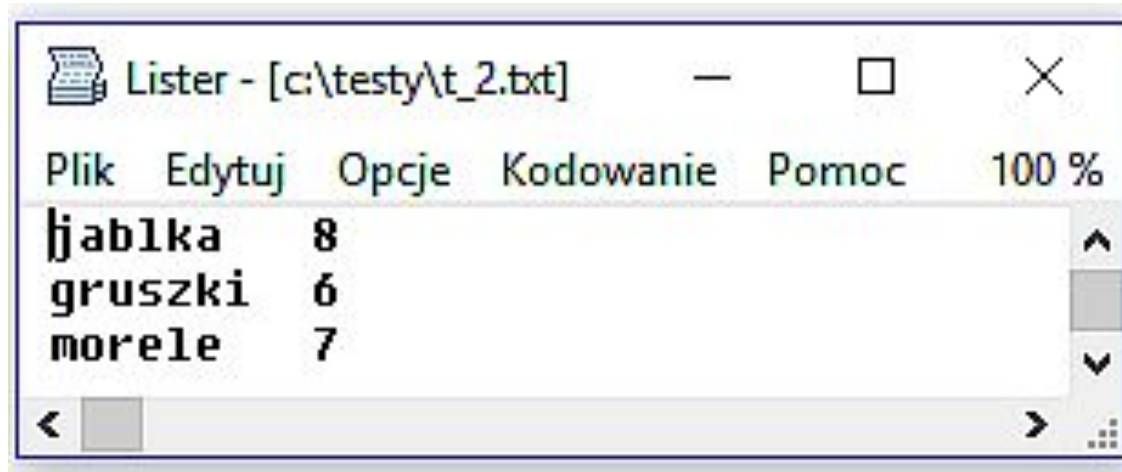
```
//pliki, nazwa: "C:/testy/t_2.txt"
#include <bits/stdc++.h>
using namespace std;

int main()
{ string nazwa_pliku;
  ofstream wy;
  cout <<"podaj nazwe pliku :";
  cin >> nazwa_pliku;
  wy.open(nazwa_pliku);
  if(!wy) {
    cout <<"plik nie zostal otwarty \n";
    return 1;
  }
  wy <<"jablka  " << 8<<endl;
  wy <<"gruszki  " << 6<<endl;
  wy <<"morele   " << 7<<endl;
  wy.close();    //zamkniecie pliku
  wy.close();
}
```

Przebieg programu:

podaj nazwe pliku :c:/testy/t_2.txt

W wyniku mamy plik z danymi:



```
Lister - [c:\testy\t_2.txt]
Plik  Edytuj  Opcje  Kodowanie  Pomoc  100 %
jablka  8
gruszki  6
morele   7
```

Odczytywanie pliku linijka po linijce

Ta metoda polega na odczytaniu jednej linijki tekstu i zapisaniu jej jako łańcucha znaków.

```
string linia;  
while(we) {  
    getline(we, linia); // Odczytanie jednej linii  
    cout <<linia<<endl;  
};
```

Efekt zastosowania tej techniki jest identyczny, jak użycia instrukcji cin.

Wykład 8

//pliki, wczytywanie linia po linii

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
int main()
```

```
{ ifstream we("C:/testy/t_1.txt");
```

```
  string linia;
```

```
  while(we) {
```

```
    getline(we, linia); // Odczytanie jednej linii
```

```
    cout <<linia<<endl;
```

```
  };
```

```
  we.close();
```

```
}
```

Wynik:

truskawki	10
maliny	12
jagody	14

Odczytywanie pliku znak po znaku

Ta metoda polega na odczytywaniu z pliku po jednym znaku i jako jedyna z wszystkich opisanych jest dla nas całkiem nowa. Ale oczywiście tak jak poprzednie, również łatwo ją opanować.

```
char a;  
we.get(a);
```

Powyższy kod odczyta jeden znak i zapisze go w zmiennej **a**.

Ta metoda odczytuje wszystkie rodzaje znaków, a więc także spacje, znaki nowego wiersza, znaki tabulacji itd.

Wykład 8

//pliki, wczytywanie znak po znaku

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
int main()
```

```
{ ifstream we("C:/testy/t_1.txt");
```

```
    char a;
```

```
    while(we) {
```

```
        we.get(a);
```

```
        cout << a;
```

```
    };
```

```
    we.close();
```

```
}
```

Wynik:

truskawki 10

maliny 12

jagody 14

Odczytywanie całego pliku

Często trzeba odczytać całą zawartość pliku.

Aby dowiedzieć się czy można kontynuować odczytywanie danych, należy użyć wartości zwrotnej funkcji `getline()`. Funkcja ta nie tylko wczytuje linie zawartości plików, ale dodatkowo zwraca wartość logiczną informującą, czy można kontynuować odczyt. Zatem jeśli funkcja ta zwróci wartość `true`, to wiadomo, że można kontynuować odczytywanie. Jeśli natomiast zwróci `false`, jest to znak, że został osiągnięty **koniec pliku** albo wystąpił **błąd**. W obu tych przypadkach należy zakończyć operację odczytu.

W programie wykorzystamy pętlę, aby odczytać zawartość pliku do samego końca. Najlepiej nadaje się do tego pętla `while`.

Wykład 8

```
//pliki, wczytywanie całego pliku
#include <bits/stdc++.h>
using namespace std;
int main()
{ ifstream we("C:/testy/t_1.txt");
  if(we)
  { // Udało się otworzyć plik, a więc można rozpocząć odczytywanie
    string linia; // Zmienna do odczytanych wierszy tekstu
    while(getline(we, linia)) // Jeśli jeszcze nie nastąpił koniec pliku,
      //czytamy dalej
    {
      cout << linia << endl; // Wyświetlamy odczytany tekst w konsoli
      // Można też zrobić z nim coś innego
    }
  }
  we.close();
}
```

Wynik:

truskawki	10
maliny	12
jagody	14

Sprawdzanie długości pliku

W C++ plik obsługiwany jest przez dwa wskaźniki: **wskaźnik wejścia** określa w którym miejscu pliku nastąpi kolejna operacja wejścia, drugi, **wskaźnik wyjścia** określa w którym miejscu pliku nastąpi operacja wyjścia. Gdy mamy do czynienia z operacją we/wy odpowiedni wskaźnik jest przesuwany sekwencyjnie. Dzięki odpowiednim funkcjom mamy możliwość swobodnego (niesekwencyjnego) dostępu do pliku.

Funkcja **seekg()** przesuwa wskaźnik wejścia o **offset** znaków względem położenia wskazanego przez **origin**, które musi przyjąć jedną z trzech wartości:

`ios::beg`

`ios::cur`

`ios::end`

Funkcja **seekp()** przesuwa wskaźnik wyjścia o **offset** znaków względem położenia wskazanego przez **origin**, które musi przyjąć jedną z trzech pokazanych wartości.

Aby sprawdzić długość pliku, należy przenieść wskaźnik(kursor) na koniec pliku, a następnie „spytać” strumień, gdzie jest kursor.

Wykład 8

```
//pliki, dlugosc pliku  
#include <bits/stdc++.h>  
using namespace std;
```

Funkcja `tellg()` pobiera pozycję aktualnego znaku w pliku wejściowym

```
int main()  
{ int rozmiar;  
  ifstream we("C:/testy/t_1.txt");  
  we.seekg(0, ios::end); // Przejście na koniec pliku  
  rozmiar = we.tellg(); // pozycję, która odpowiada długości pliku!  
  cout << "rozmiar pliku w bajtach wynosi: " << rozmiar << "." << endl;  
  we.close();  
}
```

Wynik:

rozmiar pliku w bajtach wynosi: 48.

Wykład 8

Dołączanie danych do pliku

Plik należy otworzyć w trybie `ios_base::app` :

```
ofstream wy(plik, ios_base::out | ios_base::app);
```

Tabela trybów języka C++

tryb	znaczenie
<code>ios_base :: in</code>	Otwiera do czytania
<code>:: out</code>	Otwiera do zapisu
<code>::out :: trunc</code>	Zapis , gdy plik istnieje jest redukowany
<code>::out :: app</code>	Zapis wyłącznie na końcu pliku
<code>::in :: out</code>	Odczyt, zapis, zapis w dowolnym miejscu
<code>::in :: out :: trunc</code>	Odczyt, zapis, redukcja pliku jeśli istnieje
<code>ios_base :: binary</code>	Otwarcie w modzie binarnym
<code>ios_base :: ate</code>	Otwiera i przechodzi na koniec pliku

Wykład 8

```
//pliki,dopisywanie danych
#include <bits/stdc++.h>
using namespace std;
const char * plik = "C:/testy/t_3.txt";
int main()
{ char zn;
  string dd;
  // pokaz dane
  ifstream we;
  we.open(plik);
  if(we.is_open()) {
    cout <<"aktualne dane"<<plik<<":\n";
    while(we.get(zn))  cout << zn;
    we.close();
  }
```



Wykład 8



```
//dodaj dane
ofstream wy(plik, ios_base::out | ios_base::app);
cout <<"Podaj nowe dane, pusty wiersz konczy\n";
while(getline(cin, dd) && dd.size() > 0) {
    wy << dd << endl;
}
wy.close();
//pokaz zmieniony plik
we.open(plik);
if(we.is_open()) {
    cout <<"nowe dane"<<plik<<":\n";
    while(we.get(zn)) cout << zn;
    we.close();
}
}
```

Wynik po pierwszym uruchomieniu:

Mercedes
Audi
Opel

Wykład 8

Przebieg programu:

aktualne daneC:/testy/t_3.txt:

Mercedes

Audi

Opel

Podaj nowe dane, pusty wiersz konczy

Skoda

Fiat

Honda

nowe daneC:/testy/t_3.txt:

Mercedes

Audi

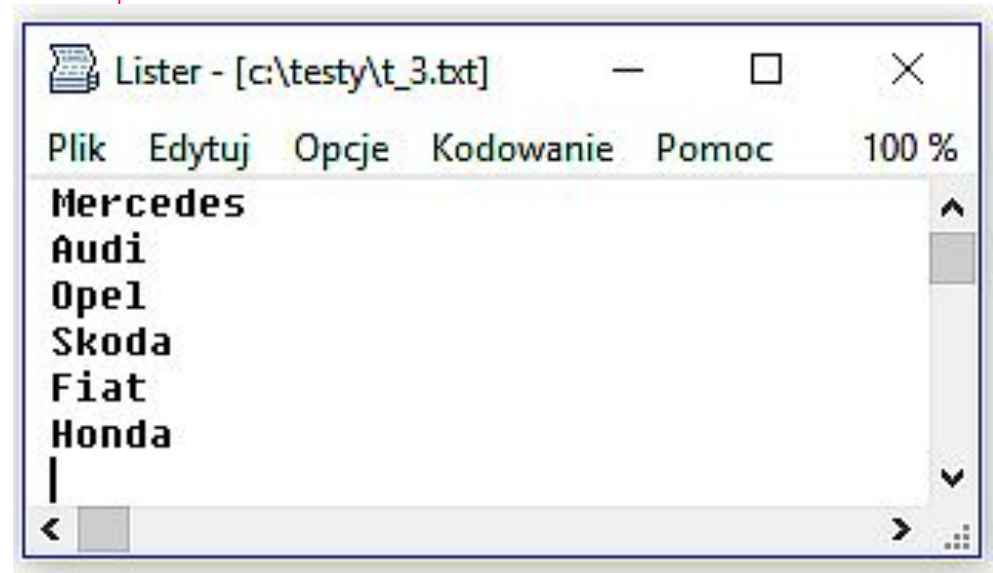
Opel

Skoda

Fiat

Honda

Zawartość pliku:



```
Lister - [c:\testy\t_3.txt]
Plik  Edytuj  Opcje  Kodowanie  Pomoc  100 %
Mercedes
Audi
Opel
Skoda
Fiat
Honda
|
```

Formatowanie wewnętrzne (incore formatting)

Klasy z pliku **fstream** realizują operacje we/wy pomiędzy plikiem i programem.

W bibliotece mamy także plik **sstream** który pomaga realizować takie operacje pomiędzy programem a obiektami typu **string**.

Proces odczytu danych formatowanych z obiektu klasy **string** lub zapisu danych formatowanych do obiektu klasy **string** nazywany jest formatowaniem wewnętrznym.

W pliku **sstream** znajdują się między innymi klasy **ostringstream** i **istringstream**.

Wszystkie klasy:

istringstream Input string stream (class)

ostringstream Output string stream (class)

stringstream Input/output string stream (class)

stringbuf String stream buffer (class)

Wykład 8

Dzięki tym klasom możemy korzystać z klasy ostream, takich jak np. getline() i innych.

Obiekty klas sstream wykorzystują bufor do obsługi typu string (string buffer), który zawiera sekwencje znaków. Taka sekwencja znaków może być dostępna wprost jako obiekt typu string przy pomocy metody **str()**.

Pokażemy wykorzystanie formatowania wewnętrznego przy pomocy dwóch programów.

W programie z obiektem istringstream korzystamy z metody instr().

Wykład 8

```
//ostream
#include <iostream>
#include <sstream>
#include <string>
using namespace std;
int main()
{ ostream xstr; //strumien string
  string nazwisko;
  int pesel;
  cout << "podaj nazwisko :";
  getline(cin, nazwisko);
  cout <<"podaj pesel :";
  cin >>pesel;
  xstr <<"Obywatel " << nazwisko << " ma pesel :"
    <<pesel<<endl;
  //metoda str() zwraca zawartosc bufor
  string wynik = xstr.str();
  cout << wynik;
  return 0;
}
```

Wynik:

podaj nazwisko :Jan Kowalski
podaj pesel :110466777
Obywatel Jan Kowalski ma pesel :110466777

Wykład 8

```
//iostream
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

int main()
{ string tekst;
  cout << "podaj tekst :";
  getline(cin, tekst);

  istringstream instr(tekst);
  string word;
  while(instr>>word)
    cout <<word<<endl;
  return 0;
}
```

Metoda klasy istream odczytuje string.
Instrukcja:
 istringstream instr(tekst);
Inicjalizuje strumień obiektem tekst.
Program wypisuje tekst słowo po słowie.

Wynik:

```
podaj tekst :siala baba mak, nie wiedziala jak!
siala
baba
mak,
nie
wiedziala
jak!
```

Wykład 8

```
// liczenie wyrazow w tekście
// stringstream.
#include <bits/stdc++.h>
using namespace std;
void stat(string st)
{map<string, int> m;
  stringstream ss(st);    // wydzieli wyraz
  string Word;             // przechowuje wyrazy
  while (ss >> Word)
      m[Word]++;           //zlicza stystyke
  map<string, int>::iterator it;
  for (it = m.begin(); it != m.end(); it++)
      cout << it->first << " : "
           << it->second << "\n";
}
int main()
{string s = "ala nie miala psa, oraz nie miala kota";
  stat(s); //wydrukuj statystyke
  return 0;
}
```

Wynik:

```
ala : 1
kota : 1
miala : 2
nie : 2
oraz : 1
psa, : 1
```

Wykład 8

Czytamy dane (typ int) z pliku a następnie dane umieszczamy w wektorze.

Rezerwujemy pamięć na cztery liczby, gdy chcemy umieścić w wektorze więcej danych zwiększamy rozmiar wektora o następne 4 miejsca.

Dane w pliku t_4.txt mają postać:

1 2 3 4 5

6 7 8

9 10 11 12 13

Wykład 8

```
#include <bits/stdc++.h>
using namespace std;
const char * plik = "C:/testy/t_4.txt";
int main()
{vector<int> v(4);
 ifstream pp(plik, ios::in); //komunikacja z plikiem
 if(pp.is_open()) cout <<"dobra";
 else
     cout <<"niedobra";
 cout <<endl;
 int i = 0;
 while(!pp.eof())
 { pp >> v[i++];
   if(i % 3 == 0) v.resize(v.size() + 4);
 }
 for(int x: v) {cout <<x<<" ";}
 cout << endl;

 return 0;
}
```

Wynik:

dobra

1 2 3 4 5 6 7 8 9 10 11 12 13 0 0 0 0 0 0

Wykład 8

Czytamy nazwiska z pliku i zapisujemy je w wektorze

Kolejny przykład jest podobny do poprzedniego – czyta dane z pliku (typ string) i zapisuje je w wektorze.

The following example is almost the same as the previous example: it reads in string data from a file and fills in vector.

Uwaga.

`std::istream::ctype_base::space` is the default delimiter which makes it stop reading further character from the source when it sees whitespace or newline.

Dane w naszym pliku:

```
const char * plik = "C:/testy/t_5.txt";
```

Mają postać:

Jan Rura Ewa Kwiatek Lola Zimna Edgar Silny

Gdy czytane są dane, imię i nazwisko są przechowywane w wektorze. Ale my chcemy traktować imię i nazwisko jako parę. Dlatego pary umieszczamy w map.

Wykład 8

```
#include <bits/stdc++.h>
using namespace std;
const char * plik = "C:/testy/t_5.txt";
int read_words(vector<string>& words, ifstream& in)
{ int i = 0;
  while(!in.eof())
    in >> words[i++];
  return i-1;
}
int main()
{ ifstream ifp(plik);
  vector<string> w(500);
  int number_of_words = read_words(w, ifp);
  w.resize(number_of_words);
  map<string, string> wMap;
  for(int i = 0; i < number_of_words;) {
    wMap.insert(pair<string, string>(w[i], w[i+1]));
    i += 2;
  }
  cout << "liczba obywateli = " << wMap.size() << endl;
  for(auto it = wMap.begin(); it != wMap.end(); it++)
    cout << it->first << " " << it->second << endl;
}
```

Wynik:

liczba obywateli = 4
Edgar Silny
Ewa Kwiatek
Jan Rura
Lola Zimna

Russian Peasant Multiplication

The algorithm in fact may have Egyptian roots, as a similar procedure has been routinely used in the famous Rhind Papyrus [Midonick, pp. 706-732, Fauvel, pp. 14-16]. It is sometimes referred to as the Ethiopian (Peasant) Multiplication; the linkage could be explained by the proximity of the two nations and intermixing of their cultures. It is curious to note in passing that the great-grandfather of the illustrious Russian poet Alexander Serge'evich Pushkin was a blackamoor of Ethiopian origin. However, the spurious idea that Ibrahim Petrovitch Gannibal, a page to Peter the Great, may be a historic conduit for the algorithm from North Africa to Russia clashes with the Peasant part of the designation. A pity.

The algorithm instructs us to create a column beneath each of the multiplicands. We start by dividing the first number by 2 (and dropping the remainder if any) and recording the result in the first column. Then we divide by 2 the recorded number and write the result below. The process of division by two of the successive results continues until 1 is reached. In the second column we write the numbers obtained by successive **multiplication** by 2 that starts with the second multiplicand. Let's see how it works for the product 85×18 :

Wykład 8

Dividing 85 by 2 (and dropping the remainder) we get successively: 85, 42, 21, 10, 5, 2, 1. Now, multiply by 2 the second number, 18, as many times: 18, 36, 72, 144, 288, 576, 1152. Let's write these in two columns:

$$\begin{array}{rcl} 85 & \times & 18 \\ \hline 85 & & 18 \\ 42 & & 36 \\ 21 & & 72 \\ 10 & & 144 \\ 5 & & 288 \\ 2 & & 576 \\ 1 & & 1152 \\ & & \hline & & 1530 \end{array}$$

Zaznaczamy nieparzyste liczby w 1 kolumnie i sumujemy odpowiadające im liczby z 2 kolumny. Mamy wynik!

Wykład 8

```
#include <bits/stdc++.h>
using namespace std;
int RussianPeasant(int a, int b)
{int x = a, y = b;
 int val = 0;
 cout << left<<setw(5)<<x << left<<setw(5)
      <<y<< left<<setw(5)<<val<<endl;
 while (x > 0) {
   if (x % 2 == 1) val = val + y;
   y = y << 1; // double
   x = x >> 1; // half
   cout
 << left<<setw(5)<<x<< left<<setw(5)
      <<y<< left<<setw(5)<<val<<endl;
 }
 return val;
}
```

```
int main() {
  int a,b;
  cout <<"podaj a i b : ";
  cin>>a>>b;
  RussianPeasant(a, b);
  cout <<"komputer : "<<a*b;
  return 0;
}
```

Wynik:

```
podaj a i b : 85 13
85  13  0
42  26  13
21  52  13
10  104  65
5   208  65
2   416  273
1   832  273
0   1664 1105
komputer : 1105
```

Copy Files in C++

To copy the content of one file to the other file in C++ programming, you have to first ask to the user to enter the source file to open the source file and then ask to enter the target file to open the target file and start reading the source file's content character by character and place or write the content of the source file to the target file character by character at the time of reading as shown here in the following program.

C++ Programming Code to Copy Files

Following C++ program ask to the user to enter the two file name. First file name is source file (the content of this file is going to copy to the target file) and the second name is target file (the content of the source file is copied to this target file).

After entering the two file name, the below program start copying the content of the source file to the target file.

filebuf -- buffer for file I/O

filebufs specialize streambufs to use a file as a source or sink of characters. Characters are consumed by doing writes to the file, and are produced by doing reads. When the file is seekable, a filebuf allows seeks. At least 4 characters of putback are guaranteed. When the file permits reading and writing, the filebuf permits both storing and fetching. No special action is required between gets and puts (unlike stdio). A filebuf that is connected to a file descriptor is said to be open. Files are opened by default with a protection mode of openprot, which is 0644.

public member function

std::ios::rdbuf (Josuttis, str 869)

get (1) streambuf* rdbuf() const;

set (2) streambuf* rdbuf (streambuf* sb);

Get/set stream buffer

The first form (1) **returns a pointer** to the stream buffer object currently associated with the stream.

The second form (2) also sets the object pointed by sb as the stream buffer associated with the stream and clears the error state flags.

Wykład 8

Program kopiuje dane z jednego pliku do drugiego

```
//#include <bits/stdc++.h>
#include <fstream>
#include <cstdio>
const char *splik = "C:/testy/t_5.txt"; //zrodlo
const char *dplik = "C:/testy/t_6.txt"; //odbiorca
int main () {
    // std::fstream src,dest;
    std::ifstream src;
    std::ofstream dest;
    src.open (splik);
    dest.open (dplik);
    std::filebuf* inbuf = src.rdbuf();
    std::filebuf* outbuf = dest.rdbuf();

    char c = inbuf->sbumpc();
    while (c != EOF) {
        outbuf->sputc (c);
        c = inbuf->sbumpc();
    }
    dest.close();
    src.close();
    return 0;
}
```

Z1.

Napisz program, który otwiera dwa pliki tekstowe z nazwiskami.

Do otwartego trzeciego pliku kopiuje nazwiska z pierwszego i drugiego pliku. Tak otrzymane nazwiska w trzecim pliku należy posortować alfabetycznie i pokazać na ekranie.

Z2.

Zbuduj klasę Pracownik z następującymi składowymi:

- publiczne pole **nazwisko** i **miejsce_urodzenia** pracownika typu char
- publiczne pole **pensja** pracownika typu float
- publiczna funkcja wyświetlająca nazwisko pracownika
- publiczna funkcja wyświetlająca miejsce urodzenia pracownika
- publiczna funkcja wyświetlająca pensję pracownika
- Napisać funkcje testującą main(). Wprowadź dane trzech pracowników. Umieść te dane w pliku tekstowym.



Wykład 8

KONIEC