C++: I/O C/C++

Zajęcia laboratoryjne nr 2 - Metodyka i Techniki Programowania II

Sem. zimowy, r. akadem. 2021/2022

Autor instrukcji: dr inż. Andrzej Staniszewski

Wstęp

Operacje wejścia-wyjścia nie są częścią definicji ani języka C ani też języka C++. W obu językach realizowane są przez funkcje/strumienie zdefiniowane w stosownych bibliotekach: **stdio.h** dla języka C i **iostream** dla języka C++.
Operacje wejścia-wyjścia można realizować na wiele sposobów i na kilku poziomach w obu językach: od dostępu do pojedynczych bajtów/znaków, aż po funkcje wysokiego poziomu, stosujące buforowanie i struktury/klasy pośrednie w realizacji procesów wyjściowych i wejściowych.
W niniejszym laboratorium poruszone zostaną dwa zagadnienia realizacji operacji we/wy dla języka C i C++:
- użycia funkcji/strumieni wysokiego poziomu z możliwością formatowania danych,
- operacje na plikach.

Język C

C: Funkcje printf() i scanf()

Prototyp funkcji **printf()** jest następujący:

int printf(char *format, arg1, arg2, ….)

Funkcja **printf()** w argumencie **format** ma zapisany sposób przekształcania, formatowania i wypisywania argumentów **arg1, arg2, ….** do standardowego wyjścia. Argument **format** steruje procesem wypisywania danych i składa się z dwóch typów elementów: zwykłych znaków, które kopiowane są wprost do strumienia wyjściowego i znaków specyfikacji konwersji, które powodują odpowiednie przekształcenie i wypisanie kolejnego argumentu (**arg1, arg2, …**) funkcji **printf()**.
Każda specyfikacja konwersji zaczyna się od znaku **%** a kończy znakiem określającym typ danych. Pomiędzy znakiem **%** a znakiem typu danych mogą znajdować się modyfikatory.
Przykład użycia funkcji **printf()**:

printf(”L. wierszy %d\n L. slow %d\n L. znakow\n”, nl, nw, nc);
Powyższe można zrealizować poprzez następujące wywołanie funkcji printf(): **printf(”%s %d\n%s %d\n%s %d\n”, ”L. wierszy”, nl, ”L. slow”, nw, ”L. znakow”, nc);**

Uwaga: jeżeli podałeś(-aś) argumenty złego typu lub nie podałeś(-aś) wystarczającej liczby argumentów to otrzymasz błędne wyniki!

Ćwiczenie 1:
Zapoznaj się z modyfikatorami oraz typami danych występującymi w specyfikacji konwersji funkcji **printf()**.

Ćwiczenie 2:
Wykorzystując wiadomości zdobyte w zadaniu 1 wypisz tekst ”Ahoj, przygodo!” w następujący sposób; dwukropki określają szerokość pola:

:Ahoj, przygodo!:
:Ahoj, przy:
:Ahoj, przygodo! :
: Ahoj, przygodo!:
: Ahoj, przy:
:Ahoj, przy :

Ćwiczenie 3:
Dlaczego format **printf(”%*s”, max, s)** powoduje wypisanie co najwyżej **max** znaków z ciągu **s**?

Ćwiczenie 4:
Zapoznaj się z formatami i modyfikatorami funkcji **scanf()**.

Ćwiczenie 5:
Zapoznaj się z pozostałymi funkcjami z „rodziny” funkcji **printf(**) i **scanf()**.

C: Plik

W języku C do obsługi pliku używany jest wskaźnik na strukturę **FILE** (potocznie określany jako „wskaźnik do pliku”). Programista nie musi znać samej struktury **FILE**.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   FILE *f;
   return 0;
}
```

Otwieranie pliku
Do otwarcia pliku służy metoda **fopen()**, zwracająca wskaźnik na strukturę **FILE**, albo **null**, jeśli otwarcie pliku jest niemożliwe. Argumenty funkcji **fopen()** to ścieżka oraz tryb otwarcia pliku. Po zakończeniu operacji na pliku, plik należy zamknąć, aby nie blokować innym programom dostępu do niego.

```
if ((f=fopen("file.txt", "w"))==NULL) {
    printf("Cant write.\n");
    exit(1);
    }
fclose(f);
```

Wyjaśnij powyższy kod.

Zapis do pliku
W języku C istnieje kilka funkcji służących do zapisu do pliku - **fprintf**, **fputc**, **fputs** lub **fwrite**. Funkcja **fprintf()** może być używana analogicznie do funkcji **printf()**, przy czym pierwszym argumentem dla **fprintf()** jest wskaźnik na strukturę **FILE**.

```
int year = 2013;
char *month = "January";
int day = 4;
fprintf(f, "Today is %s %i, %i.", month, day, year);
```

Przećwicz zapis do pliku za pomocą innych funkcji.

Odczyt pliku

Plik do odczytu jest otwierany analogicznie, jak do zapisu, przy czym drugi argument fopen przy otwarciu pliku do odczytu to ”r”. Jakie są inne tryby otwierania pliku? Do czego służą?

```
if ((f=fopen("file.txt", "r"))==NULL) {
    printf("Cant read.\n");
    exit(2);
    }
fclose(f);
```

Do operacji czytania pliku służą polecenia **fscanf()**, **fgets()**, **fgetc()**, **fread()**. Są one odpowiednikami wcześniej wymienionych funkcji, służących do zapisu do pliku.

```
int y;
char m[8];
int d;

fscanf(f, "Today is %s %i, %i.", m, &d, &y);
fclose(f);
printf("From a file: %s %i, %i.", m, d, y);
```

Przećwicz wczytywanie zmiennych z pliku za pomocą innych funkcji.

Poruszanie się po pliku

Do poruszania się po pliku służą funkcje fsetpos(), fgetpos() i fseek(). Z wszystkimi operacjami na pliku związane jest pojęcie kursora, który przesuwa się po pliku w czasie realizacji operacji czytania pliku lub zapisu do pliku. Kursor ten można przestawiać za pomocą wymienionych funkcji. Przeanalizuj i wyjaśnij poniższy przykład.

```
if ((f=fopen("file.txt", "r"))==NULL) {
      printf("Cant read.\n");
      exit(2);
   }
int size = 0;

    fseek (f, 0, SEEK_END);
    fgetpos (f, &size);
    fclose(f);
    printf("File size is: %i", size);
```

TEKST KONTYNUOWANY NA STRONIE NASTĘPNEJ!

Język C++

C++: Strumienie cout i cin

W języku C++ mamy następujące strumienie, zdefiniowane standardowo:

cout cin cerr clog
wcout wcin wcerr wclog

W pierwszej linijce wymienione zostały strumienie działające na zwykłych znakach typu **char**.
W drugiej linijce wymienione zostały strumienie działające na znakach szerokich typu w**char_t**.

Ćwiczenie 6.
Dowiedz się więcej o znakach szerokich i typie **wchar_t**.

Aby korzystać z powyższych strumieni należy umieścić w programie deklarację:
#include <iostream>

cout – powiązany jest ze standardowym urządzeniem wyjścia – zwykle ekran,
cin – powiązany jest ze standardowym urządzeniem wyjścia – zwykle klawiatura,
cerr – powiązany jest z urządzeniem, na które chcemy wypisywać komunikaty o błędach – zwykle ekran,
clog – tak samo jak **cerr**, ale strumień ten jest buforowany.

Ponieważ już „nieformalnie" stosowaliśmy strumienie cout i cin oraz związane z nimi przeciążone (dlaczego?!) operatory **<<** i **>>**, to pozostaje odnieść się do sterowania formatem tekstu wyjściowego i/lub wejściowego.

Ćwiczenie 7.
Zapoznaj się w literaturze z flagami stanu formatowania i manipulatorami oraz sposobami ich użycia w strumieniu wyjściowym i wejściowym.

Ćwiczenie 8.
Użyj zdobytych wiadomości do wypisania tekstu "Ahoj, przygodo!" w sposób opisany w ćwiczeniu

C++: Wejście i wyjście plikowe

Do jednoczesnych operacji wejścia z pliku i wyjścia na plik służy klasa **fstream**. Klasa **ifstream** służy do obsługi wejścia plikowego, a klasa **ofstream** do obsługi wyjścia plikowego. Definicje tych klas znajdują się w pliku nagłówkowym **fstream**, a wywodzą się z klas zdefiniowanych w pliku nagłówkowym **iostream**.
W języku C++ do obsługi pliku używana jest zmienna (obiekt) klasy **fstream** (tu: obiekt o nazwie **plk**).
Programista nie musi znać samej klasy **fstream**.

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   fstream plk;    //obiekt klasy fstream o nazwie plk
   return 0;
}
```

Otwieranie pliku

Plik w programie możemy otworzyć tylko do pisania, tylko do czytania lub jednoczesnego pisania i czytania. Służą do tego obiekty klas **ofstream**, **ifstream** lub **fstream**. Do otwarcia pliku służy metoda **open()**. Do zamknięcia pliku służy metoda **close()**. Otwarcia pliku wyjściowego/wejściowego możemy dokonać w następujący sposób:

Sposób 1 (użycie konstruktora obiektu):

Pisanie do pliku:

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   ofstream plk (“dane_wy.txt”);

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

Czytanie z pliku:

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   ifstream plk (“dane_we.txt”);

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

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   fstream plk (“dane_wy.txt”, ios::out);

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

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   fstream plk (“dane_we.txt”, ios::in);

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

Sposób 2:

Pisanie do pliku (użycie metody obiektu):

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   ofstream plk;
   plk.open (“dane_wy.txt”);

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

Czytanie z pliku: …

Ćwiczenie 9:

Napisz programy realizujące kod pisania/czytania do/z pliku z wywołaniem metody **open()** dla obiektów (zmiennych) klasy **ifstream** i **fstream**. Sprawdź poprawność działania programu.

W powyższym kodzie pojawiły się statyczne stałe **ios::out** i **ios::in** określające tryb otwarcia pliku (tu: odpowiednio do **pisania** i **czytania**). Innymi stałymi określającymi tryb otwarcia pliku są np.: **ios::app** (ang. append) – dopisywanie danych do pliku, czy **ios::ate** (ang. at end) – po otwarciu ustaw wskaźnik pliku na jego koniec. Stałe określające tryb otwarcia pliku można łączyć operatorem logicznym **or**, znak: ‘**|**’:

fstream strumien (“plik.dat”, ios::in | ios::out);

co otwiera plik do czytania i zapisu.
Statyczne stałe **ios::in**, **ios::out** itd. pochodzą (dziedziczone są) z klasy **ios_base**.

Ćwiczenie 10:
Sprawdź jak działa otwarcie pliku „dane_wy.txt” do pisania (**ofstream**, **fstream)** gdy plik:

a. nie istnieje
b. istnieje
c. istnieje i nie jest pusty (w tym celu utwórz niepusty plik „dane_wy.txt” pod dowolnym edytorem(sic!) tekstu)

Takie same badanie przeprowadź przy otwarciu pliku do czytania (**ifstream**, **fstream**). Czy kod programu diagnozował wszystkie zachowania obiektu klasy **ifstream**?
Podsumuj wyniki ćwiczenia.

Ćwiczenie 11:
Poszerz kody z ćwiczenia 2 o diagnostykę otwarcia pliku używając metody **good()** lub **is_open()** zwracające wartości: **true**, gdy plik został otwarty lub **false**, gdy otwarcie pliku nie powiodło się.
Fragment uzupełniającego kodu może być następujący:

```
if (plk.good() == false) {
   cout << “Plik nie istnieje!” << endl;
   return -1;
}
```

Przykładowy kod programu to:

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   ifstream plk ("dane_we.txt");
   if(plk.is_open() == false) {
            cout << "Plik nie istnieje!" << endl;
            return -1;
   }

//operacje na pliku

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

Ćwiczenie 12:
Napisz programy wykorzystujące obiekty innych klas i inne metody. Sprawdź ich działanie.

Zapis do pliku
Zapisu do pliku dokonujemy deklarując obiekt (strumień) klasy **ofstream** lub **fstream** i wiążąc go z plikiem wyjściowym, a następnie pisząc do pliku (używając strumienia) podobnie jak przy użyciu obiektu **cout**. Różnica zachodzi tylko w ujściu strumienia — przy strumieniu **cout** jest to ekran monitora, a teraz będzie to plik. Po zakończeniu pracy z plikiem strumień należy zamknąć; metoda **close()**.

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
fstream plik;
plik.open("dane.dat", ios::out);

plik << "Imie: Jan" << endl;
plik << "Nazwisko: Kowalski" << endl;
plik << "Nr tel.: 12 3456" << endl;
plik.close();
}
```

```
#include <iostream>
#include <fstream>
using namespace std;

string imie, nazwisko;
int nrtel;

int main(){

fstream plik;

cout << "Podaj imie: "; cin >> imie;
cout << "Podaj nazwisko: "; cin >>nazwisko;
cout << "Podaj nr tel.: "; cin >> nrtel;

plik.open("wizytowka.txt", ios::out);
plik << imie << endl;
plik << nazwisko << endl;
plik << nrtel << endl;
plik.close();
}
```

Ćwiczenie 13:
Zapisz, skompiluj i wykonaj kody powyższych programów. Jakie są ich wyniki? Przeanalizuj i porównaj kody programów. Wyjaśnij zmienną typu **string**.
Odczyt (z) pliku (tu: wierszami)
Plik do odczytu jest otwierany analogicznie jak do zapisu. Drugi argument metody **open()** to **ios::in**. Jakie są inne tryby otwierania pliku? Do czego służą?

```
#include <iostream>
#include <fstream>
using namespace std;

string imie, nazwisko, nrtel;

int main(){
fstream plik;

plik.open("wizytowka.txt", ios::in);
if(plik.good() == false) {
		cout << "Plik nie istnieje!" << endl;
		return -1;
		}

getline (plik, imie);
getline(plik, nazwisko);
getline (plik, nrtel);

plik.close();

cout << "Imie: " << imie << endl;
cout << "Nazwisko: " << nazwisko << endl;
cout << "Nr tel.: " << nrtel << endl;

return 0;
}
```

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

string imie, nazwisko;
int nrtel;

int main(){
fstream plik;
string linia;
int nrl = 1;

plik.open("wizytowka.txt", ios::in);
if(plik.good() == false) {
		cout << "Plik nie istnieje!" << endl;
		return -1;
		}

while (getline(plik, linia)) {
		switch (nrl) {
				case 1: imie = linia; break;
				case 2: nazwisko = linia; break;
				case 3: nrtel = atoi(linia.c_str()); break;
				}
				nrl++;
}
plik.close();

cout << "Imie: " << imie << endl;
cout << "Nazwisko: " << nazwisko << endl;
cout << "Nr tel.: " << nrtel << endl;

return 0;
}
```

Ćwiczenie 14:
Zapisz, skompiluj i wykonaj kody powyższych programów. Jakie są ich wyniki? Przeanalizuj i porównaj kody programów. Usuń z folderu plik "**wizytowka.txt**" – jaki otrzymasz wynik programu?

Z którą funkcją związana jest biblioteka **cstdlib**? Przećwicz wczytywanie zmiennych z pliku za pomocą innych funkcji.

Istnieją inne metody wczytywania danych z pliku: znakami, wierszami (vide: przykłady powyżej), blokami oraz znakowo i binarnie. Służą ku temu odpowiednie funkcje i metody zdefiniowane w bibliotekach.

Poruszanie się po pliku
Do poruszania się po pliku (strumieniu) służą metody:

streampos tellg( ) — zwraca aktualną pozycję lokalizatora do odczytu (litera 'g' pochodzi od *get*). Tryb otwarcia pliku musi zawierać **ios::in**.
streampos tellp( ) — zwraca aktualną pozycję lokalizatora do zapisu (litera 'p' pochodzi od *put*). Tryb otwarcia pliku musi zawierać **ios::out**.
Typem zwracanym przez powyższe metody jest **streampos**. Zwykle jest to typ **long**.

Z poruszaniem się po pliku związane są **lokalizatory** zawierające numer bajtu (licząc od zera) w pliku (strumieniu), na który nastąpi następny zapis/odczyt. Lokalizatory są typu **streampos**. Po otwarciu pliku domyślnie lokalizatory ustawiane są na samym jego początku, a więc na bajcie zerowym (numer zero), chyba że plik został otwarty w trybie **ios::ate** lub **ios::app** (ang. append – dołączyć), gdy plik pozycjonowany jest na bajcie pierwszym za ostatnim (numeryczna wartość lokalizatora jest wtedy równa długości pliku w bajtach). Po każdej operacji czytania/pisania lokalizator przesuwany jest tak, aby wskazywał na pierwszy bajt jeszcze nie wczytany/zapisany.

Poniższe metody ustawiają lokalizatory na żądanej pozycji w pliku:

ostream& seekg(streampos poz) — przesuwa lokalizator do odczytu na pozycję **poz**. Zwraca referencję do strumienia, na rzecz którego została wywołana. Tryb otwarcia pliku musi zawierać **ios::in**.
ostream& seekp(streampos poz) — przesuwa lokalizator do zapisu na pozycję **poz**. Zwraca referencję do strumienia, na rzecz którego została wywołana. Tryb otwarcia pliku musi zawierać **ios::out**.; poza tym analogiczna do metody seekg(streampos).
ostream& seekg(streamoff offset, ios::seek_dir poz) — przesuwa lokalizator do odczytu na pozycję **offset** bajtów licząc od pozycji **poz**. Argument **offset** może być ujemny. Typy **streamoff** i **ios::seek_dir** są aliasami typów całkowitoliczbowych. Argument **poz** musi być równy jednej ze stałych statycznych z klasy **ios**:
ios::beg — licz od początku pliku,
ios::cur — licz od aktualnej pozycji w pliku,
ios::end — licz od końca pliku.
Zwraca referencję do strumienia, na rzecz którego została wywołana.
ostream& seekp(streamoff offset, ios::seek_dir poz) — przesuwa lokalizator do zapisu na pozycję **offset** bajtów licząc od pozycji **poz**. Metoda ta jest analogiczna do metody seekg(streamoff offset, ios::seek_dir poz).

Przy używaniu metod **seekg()** (get) tryb otwarcia pliku musi zawierać **ios::in**.
Przy używaniu metod **seekp()** (put) tryb otwarcia pliku musi zawierać **ios::out**.
Próba sięgnięcia przed początek bądź za koniec pliku powoduje przejście strumienia do stanu **bad**, co można sprawdzić za pomocą warunku **if (strumien) ...**.

Pozycjonowanie odczytu i zapisu z/do pliku ilustruje poniższy program:

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {

   int k;

fstream file("file.dat",ios::in | ios::out | ios::binary);
if (! file ) {
   cout << "Nie mozna otworzyc file" << endl;
   return -1;
   }
   file.seekg(0, ios::end);
   streamsize len = file.tellg();
   cout << "Plik ma dlugosc (1): " << len << " bajtow\n";
   file.seekg(0);

   cout << "Kolejne bajty zawieraja:" << endl;
   while ( (k = file.get()) != EOF )
            cout << k << " ";
   cout << endl;

   file.clear();
   file.seekg(4);
   file.read((char*) &k, 4);
   cout << "Integer od pozycji 4: " << k << endl;

   file.seekp(12);
   file.write((char*)&k, 4);

   file.seekg(0);
   cout << "Kolejne bajty pliku teraz zawieraja:" << endl;
   while ( (k = file.get()) != EOF )
            cout << k << " ";
   cout << endl;

   file.clear();

   file.seekg(0, ios::end);
   len = file.tellg();
   cout << "Plik ma dlugosc (2): " << len << " bajtow\n";
   file.close();
}
```

Ćwiczenie 15:
Przeanalizuj dokładnie kod programu. Jakie nowe metody zauważyłeś? Poszukaj informacji o ich użyciu i działaniu.
Zapisz, skompiluj kod powyższego programu. Przed wykonaniem przygotuj pod edytorem tekstu plik "**file.dat**" (może to być jeden z poprzednich plików tekstowych zapisanych przez wcześniej wykonywane programy, np. plik "dane.dat" czy "wizytowka.txt"; zmień nazwę pliku). Jaki jest wynik programu?
Przeanalizuj go i spróbuj wyciągnąć właściwe wnioski patrząc na wydruk na ekranie. Czy zawartość pliku "file.dat" została zmieniona po wykonaniu programu?