

Wykład 14

- Biblioteki standardowe do obsługi we/wy w C++
- Strumienie
- Operatory `>>` i `<<`
- Domniemania w pracy strumieni predefiniowanych
- Priorytet operatorów `>>` i `<<`
- Sterowanie formatem
 - flagi stanu formatowania
 - zmiana trybu formatowania – funkcje `setf` i `unsetf`
 - inne funkcje składowe do sterowania formatem

Biblioteki standardowe do obsługi we/wy w C++

- 1. Biblioteka stdio (standard input/output) – zapewnia kompatybilność z językiem C**
 - 2. Biblioteka stream – zbiór klas zawierających funkcje we/wy – stara wersja**
 - 3. Biblioteka iostream – zalecana do stosowania w programowaniu w C++**
-
-

Biblioteka iostream (input output stream)

Jej elementy (klasy, funkcje składowe i operatory) umożliwiają:

- Wyprowadzanie i wprowadzanie informacji do i ze standardowych urządzeń we/wy (ekran monitora, klawiatura)**
- Operacje na plikach danych**
- Wyprowadzanie i wprowadzanie informacji do/z określonego obszaru pamięci (np. tablicy znakowej)**

Aby skorzystać z biblioteki `iostream` należy dyrektywą `include` włączyć do programu odpowiednie pliki nagłówkowe:

`iostream.h` jakiegokolwiek korzystanie z biblioteki

`fstream.h` operacje na plikach zewnętrznych

**`strstream.h` operacje `we /wy` na tablicach (formatowanie
wewnętrzne)**

Strumienie

Wprowadzanie i wyprowadzanie informacji C++ traktuje jako strumień bajtów płynący od źródła do ujścia.

W języku C++ wykonywanie operacji we/wy zaimplementowano w sposób obiektowy. Operacje na strumieniach dostępne są poprzez obiekty odpowiednich **klas**.

W C++ dostępna jest oczywiście również tzw. wersja „proceduralna” obsługi wejścia/wyjścia, która bazuje na bibliotece `stdio`. Ten sposób obsługi już poznaliśmy.

Wszystkie klasy obsługujące we/wy (klasy `istream`) bazują na koncepcji strumienia, w której przedstawiany jest on jako „rozszerzony plik”. Taki rozszerzony plik może być zarówno źródłem bajtów, jak też miejscem ich zapisu. W systemie operacyjnym urządzenia we/wy (jak klawiatura, monitor, drukarka oraz porty komunikacyjne) widziana są przez ich sterowniki właśnie jako rozszerzone pliki.

Obiekty klas odpowiedzialnych za wykonywanie operacji wejścia/wyjścia współpracują z tymi plikami.

Cechy strumienia, z punktu widzenia implementacji klasy, zależą od jej możliwości: metod, parametrów i zaimplementowanych operatorów wstawiania i pobierania danych.

Podjęcie obiektowe pozwoliło uniknąć wielokrotnego definiowania tych samych operacji dla klas działających na różnych strumieniach (odpowiadających różnym urządzeniom). Na przykład składnia operacji czytania z i pisania do strumieni jest taka sama dla strumieni związanych z pamięcią jak i dla strumieni związanych z plikami dyskowymi.

Działanie strumieni można rozpatrywać na dwóch poziomach:

- niski: od źródła do ujścia przesyłane są określone bajty informacji, ich znaczenie nie jest istotne; przez strumień odbywa się tylko przepływ bajtów; klasa `stringstream`

- wysoki: przesyłanie informacji przez strumień wraz z jej interpretowaniem (formatowaniem); klasy `istream` (strumień wejściowy), `ostream` (strumień wyjściowy), `iostream` (potomna powyższych klas – umożliwia definiowanie własnych strumieni wejściowych i wyjściowych).

Aby można było posłużyć się strumieniem należy:

1. Zdefiniować w pamięci "ośrodek" sterowania strumieniem
2. Wskazać mu urządzenie zewnętrzne, którym ma się zajmować
3. Zlikwidować strumień

Pozornie jest to skomplikowane, jednak wczytanie pojedynczej liczby x z klawiatury załatwia instrukcja:

```
cin >> x;
```

Strumienie predefiniowane

-Obiekty zdefiniowane w bibliotece `iostream` – czynności 1-3 są wykonywane automatycznie.

Są to obiekty o nazwach: `cout`, `cin`, `cerr`, `clog`

Aby można było z nich skorzystać należy w programie umieścić dyrektywę

```
#include <iostream.h>
```

`cout` – jest związany ze standardowym urządzeniem wyjścia (ekran monitora)

`cin` - związany ze standardowym urządzeniem wejścia (klawiatura)

`cerr` - związany ze standardowym urządzeniem, na które chcemy wyprowadzać komunikaty o błędach (ekran monitora); strumień ten jest niebuforowany

`clog` – jak powyższy, ale buforowany.

Co to znaczy buforowany i niebuforowany ?

Operatory >> i <<

```
cout << x;
```

powoduje wstawienie liczby x (wartości zmiennej x) do strumienia. Jego ujście to ekran monitora. Operator ten często nosi nazwę operatora *wstawiania*.

```
cin >> y;
```

wczytuje informację (np. wartość do zmiennej y). Operator >> nazywa się często operatorem *ekstrakcji* lub pobrania ze strumienia.

```
#include <vcl.h>
#include <iostream.h>
int main()
{
    int myInt;
    long myLong;
    double myDouble;
    float myFloat;
    unsigned myUnsigned;
    cout << "int = ";
    cin >> myInt;
    cout << "long = ";
    cin >> myLong;
    cout << "double = ";
    cin >> myDouble;
    cout << "float = ";
    cin >> myFloat;
    cout << "unsigned = ";
    cin >> myUnsigned;
    cout << "Wprowadzono dane: " << endl;
    cout << "int = " << myInt << endl
        << "long = " << myLong << endl
        << "double = " << myDouble << endl
        << "float = " << myFloat << endl
        << "unsigned = " << myUnsigned << endl;
    system("PAUSE");
    return 0;
}
```

E:\Zajecia_zima_2005\Metody_programowania_C_2\Wyklad_3\Przyklad_1\Pojekt3_1.exe

```
int = 345
long = 1800091
double = 234.9012e15
float = 345.00119
unsigned = 5671
Wprowadzono dane:
int = 345
long = 1800091
double = 2.34901e+17
float = 345.001
unsigned = 5671
Aby kontynuować, naciśnij dowolny klawisz . . .
```

```
system ("PAUSE");
```

E:\Zajecia_zima_2005\Metody_programowania_C_2\Wyklad_3\Przyklad_1\Projekt3_1.exe

```
int = 901
long = 9000011
double = -890.11e-12
float = 567.199999
unsigned = -345
Wprowadzono dane:
int = 901
long = 9000011
double = -8.9011e-10
float = 567.2
unsigned = 4294966951
Aby kontynuować, naciśnij dowolny klawisz . . .
```

```
cout << "wprowadzono dane." << endl;
cout << "int = " << myInt << endl;
```

```
#include <vcl.h>
#include <iostream.h>
int main()
{
    char tekst[80];
    cout << "Wpisz swoje nazwisko: ";
    cin >> tekst;
    cout << "Nazywasz sie: " << tekst << endl;
    cout << "Wpisz swoje imie i nazwisko: ";
    cin >> tekst;
    cout << "Twoje imie i nazwisko to: " << tekst << endl;
    system("PAUSE");
    return 0;
}
```

```
E:\Zajecia_zima_2005\Metody_programowania_C_2\Wyklad_3\Przyklad_2\Przyklad_2.exe
Wpisz swoje nazwisko: Kowalski
Nazywasz sie: Kowalski
Wpisz swoje imie i nazwisko: Jan Kowalski
Twoje imie i nazwisko to: Jan
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Domniemanie w pracy strumieni predefiniowanych

Strumień cout:

- typy wbudowane, które służą do przechowywania liczb całkowitych są wypisywane w systemie dziesiętnym,
- typy `char` i `unsigned char` są wypisywane jako pojedyncze znaki ASCII
- liczby typu `float` i `double` są wypisywane z dokładnością do 6 miejsc po kropce dziesiętnej
- wskaźniki (z wyjątkiem `char*` i `unsigned char*`) są wypisywane w postaci szesnastkowej
- żądanie wypisania wskaźników `char*` i `unsigned char*` jest rozumiane jako żądanie wypisania łańcucha, który pokazuje ten wskaźnik.

Przykładowo:

```
char tekst[] = "Napis";  
char *wsk = tekst;  
cout << wsk;
```

spowoduje wyświetlenie na ekranie napisu Napis a nie adresu miejsca, gdzie jest on przechowywany.

W celu wyświetlenia adresu należy użyć instrukcji

```
cout << (void*) wsk;
```

```
#include <vcl.h>
#include <iostream.h>
int main()
{
    char tekst[] = "Napis";
    char *wsk = tekst;
    cout << "Wypisuje tekst: " << wsk << endl;
    cout << " i jego adres: " << (void*) wsk << endl;
    system("PAUSE");
    return 0;
}
```

E:\Zajecia_zima_2005\Metody_programowania_C_2\Wyklad_3\Przyklad_3\Przyklad_3.exe

```
Wypisuje tekst: Napis  
i jego adres: 1245060  
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Strumień cin:

- dane wszystkich typów mogą poprzedzać "białe znaki" (spacje, tabulatory)
- do typów liczbowych obowiązuje interpretacja znaków wg konwencji zapisu stałych dosłownych w C++, np. 12 zostanie zinterpretowane jako liczba dziesiętna 12, 012 zostanie zinterpretowane jako liczba ósemkowa (dziesiętnie 10), 0x12 zostanie zinterpretowane jako liczba szesnastkowa (dziesiętnie 18)
- jeżeli wprowadzaną liczbę chcemy poprzedzić znakiem + lub -, to pomiędzy nim a liczbą nie powinno być żadnej spacji
- wczytywanie liczby kończy się po napotkaniu znaku nie będącego cyfrą
- w liczbie w notacji półlogarytmicznej może wystąpić powtórnie znak + lub -, np. 2.3e-15 -1.44e2 -1.44e-2
- wczytywanie łańcucha rozpoczyna się po zignorowaniu spacji poprzedzających tekst i kończy się po napotkaniu pierwszego białego znaku

Priorytet operatorów >> i <<

Niski, ale nie najniższy - np. niższy od operatorów arytmetycznych, ale wyższy od operatorów logicznych.

Np..

```
cout << a + b << endl;
```

Niepotrzebne nawiasy `cout << (a + b) << endl;`

Ale tutaj `cout << a && b << cout;`

Pojawi się błąd bowiem operator << ma wyższy priorytet niż operatory logiczne, poprawna postać to

```
cout << (a && b) << endl;
```

Sterowanie formatem

Jeżeli omówione wcześniej "domniemania" przy operacjach na strumieniach nie odpowiadają potrzebom, można je zmienić. Od momentu zmiany strumień będzie czytać lub pisać dane według nowych zasad (nowego formatu).

Sposoby sterowania formatem:

- 1. Flagi stanu formatowania**
 - 2. Funkcje składowe klas strumieni istream oraz ostream**
 - 3. Manipulatory**
-
-

Flagi stanu formatowania

Zasady formatowania są zapisane za pomocą poszczególnych bitów słowa typu long int.

Przykład – liczby całkowite są przez domniemanie wypisywane w systemie dziesiętnym. Jeżeli chcemy wypisać wyniki obliczeń w postaci liczb w innym systemie, np. szesnastkowym, to musimy zmienić odpowiednią flagę stanu formatowania.

Flagi formatowania umieszczono w klasie wspólnej dla klas istream i ostream. Klasą tzw. "bazową" ("przodkiem") jest klasa ios (mechanizm dziedziczenia zostanie omówiony nieco później).

Definicja typu wyliczeniowego ułatwiającego pracę z flagami ma postać:

```
//...
public:
enum {
    skipws      = 0x0001,    // ignoruj białe znaki
    left        = 0x0002,    // justowanie lewe
    right       = 0x0004,    // justowanie prawe
    internal    = 0x0008,    // justowanie wewnętrzne
    dec         = 0x0010,    // konwersja dziesiętna
    oct         = 0x0020,    // konwersja ósemkowa
    hex         = 0x0040,    // konwersja szesnastkowa
    showbase    = 0x0080,    // pokaż podstawę konwersji
    showpoint   = 0x0100,    // pokaż kropkę dziesiętną
    uppercase   = 0x0200,    // wielkie litery w liczbach
    showpos     = 0x0400,    // znak + w liczbach
    scientific  = 0x0800,    // notacja "naukowa"
    fixed       = 0x1000,    // notacja stałoprzecinkowa
    unitbuf     = 0x2000,    // buforowanie
    stdio       = 0x4000     // współpraca z stdio
};
```

Użycie flag stanu formatowania – przykłady:

ios::left

ios::scientific

Ponieważ zostały zdefiniowane w klasie ios jako publiczne, to można je używać poza tą klasą poprzedzając kwalifikatorem zakresu.

Znaczenie flag:

skipws – w procesie wyjmowania ze strumienia ignorowane będą tzw. "białe znaki" (spacje, tabulatory, znaki nowej linii) poprzedzające właściwe znaki danych

left

right

internal – pole justowania **ios::adjustment**

przykład działania – liczba ujemna -73 zostanie wypisana za pomocą 10 znaków w następujący sposób:

-73_____	left
_____ -73	right
-_____73	internal

dec

oct

hex - pole podstawy konwersji ios::basefield decyduje w jakim systemie będą wczytywane lub wypisywane liczby całkowite

showbase – żądanie wypisywania liczb z możliwością stwierdzenia systemu zapisu

Przykład:

	flaga ustawiona	ios::showbase nie ustawiona
hex	0xc7f	c7f
oct	06177	6177
dec	1297	1297

showpos – przy wypisywaniu dziesiętnych liczb dodatnich zostaną one poprzedzone znakiem plus

uppercase – litery x lub e oznaczające podstawę konwersji będą wypisywane jako wielkie

showpoint – wypisywane będą nawet nieznaczące zera i kropka dziesiętna;
przykłady (obowiązuje dokładność do 6 miejsc)

flaga ios::showpoint	
nie ustawiona	ustawiona
3.141	3.141000
-8	-8.000000

scientific

fixed - pole odpowiedzialne za notację liczb zmiennoprzecinkowych
ios::floatfield ustawienie flagi **scientific** powoduje,
że liczby te będą wypisywane w notacji wykładniczej,
przykład:

scientific	fixed
-3.38e-2	-0.0338

unitbuf - ustawienie tej flagi oznacza rezygnację z buforowania strumienia

stdio - do współpracy ze starą biblioteką **stdio**

Zmiana trybu formatowania – funkcje setf i unsetf

Fragment definicji klasy ios bazowej dla klas m. in. ostream i istream:

```
class ios {
    long   flagi_stanu_formatowania;
public:
    // funkcje skladowe
    long setf(long, long);
    long setf(long);
    // lub wygodniejsze w uzyciu
    int width(int);
    int precision(int);
    // i jeszcze wiele innych
}
```

Zmiany trybu formatowania można dokonać na trzy różne sposoby:

- 1. Za pomocą "elementarnych" funkcji składowych klasy ios. Są to funkcje setf, unsetf. Stosując tą metodę należy pamiętać nazwy wszystkich flag.**
- 2. Za pomocą funkcji składowych klasy ios, których nazwy odpowiadają czynności, np. width, precision.**
- 3. Za pomocą manipulatorów. Zamiast wywoływać funkcję składową dla danego strumienia wystarczy wpuścić do niego specjalne kody (reprezentowane symbolicznymi nazwami), które strumień zinterpretuje jako żądanie zmiany sposobu formatowania**

Funkcje składowe klasy ios tak jak każdą funkcję składową klasy wywołuje się w następujący sposób:

obiekt.funkcja()

W przypadku gdy obiektem jest predefiniowany strumień wyjściowy wywołanie to wygląda tak:

cout.funkcja()

a w przypadku strumienia wejściowego:

cin.funkcja()

Instrukcja np. `cin.setf(ios::skipws);` ustawi flagę odpowiedzialną za ignorowanie białych znaków w strumieniu wejściowym (flaga ta jest ustawiona domyślnie).

Instrukcja `cin.unsetf(ios::skipws);` kasuje ustawienie tej flagi.

Ustawianie flag tworzących pole.

Pewne flagi występują w grupach tworzących pola, np. flagi ios::dec, ios::hex, ios::oct tworzą pole odpowiedzialne za konwersję liczb całkowitych o nazwie ios::basefield.

Domyślnie strumień wyjściowy wypisuje liczby całkowite w systemie dziesiętnym. Jeżeli chcemy wypisywać liczby np. w systemie szesnastkowym, to musimy ustawić flagę ios::hex i skasować flagę ios::dec

```
cout.setf(ios::hex);  
cout.unsetf(ios::dec);
```

Wygodniejsze jest użycie przeciążonej funkcji

```
long setf(long flaga, long nazwa_pola);
```

W naszym przykładzie wygląda to tak:

```
cout.setf(ios::hex, ios::basefield);
```

Funkcja ta nie tylko ustawia nową flagę w danym polu, ale również zeruje wszystkie pozostałe w danym polu.

```

#include <iostream.h>
int main()
{
    float x = -3854;
    long stare_flagi;
    cout << "1)  " << x << endl;
    cout << "Zapamietanie aktualnych ustawien flag\n";
    stare_flagi = cout.flags();
    cout.setf(ios::showpoint);
    cout << "2)  " << x << endl;
    cout.setf(ios::scientific, ios::floatfield);
    cout << "3)  " << x << endl;
    cout.setf(ios::uppercase);
    cout << "4)  " << x << endl;
    cout.unsetf(ios::showpoint);
    cout << "5)  " << x << endl;
    cout << "Przywrocenie poczatkowego sposobu formatowania\n";
    cout.flags(stare_flagi);
    cout << "6)  " << x << endl;
    system("PAUSE");
    return 0;
}

```

1) -3854

Zapamiętanie aktualnych ustawień flag

2) -3854.00

3) -3.854000e+03

4) -3.854000E+03

5) -3.854000E+03

Przywrócenie początkowego sposobu formatowania

6) -3854

Aby kontynuować, naciśnij dowolny klawisz . . .

Inne funkcje składowe do sterowania formatem

Oprócz omówionych funkcji do sterowania flagami stanu formatowania `setf`, `unsetf`, `flags` istnieją inne funkcje składowe:

```
int width(int);  
int width();
```

```
int precision(int);  
int precision();
```

```
int fill(int);  
int fill();
```

Funkcja width - za pomocą ilu (co najmniej) znaków należy wypisać daną liczbę. Domyślnie jest to zero, co oznacza, że należy użyć tylu znaków aby wypisać ją w całości. Funkcja zwraca dotychczasową wartość szerokości pola.

Przykład:

```
#include <iostream.h>
#include <iomanip.h>
int main()
{
    int x = 398;
    int stara_szerokosc;
    cout << "Wartosc liczby = " << x << endl;
    cout << "a potem = ";
    stara_szerokosc = cout.width(9);
    cout << x << endl;
    cout << "i jeszcze raz = ";
    cout.width(12);
    cout << x << endl;
    cout << "i tak samo jak na poczatku = ";
    cout.width(stara_szerokosc);
    cout << x << endl;
    system("PAUSE");
    return 0;
}
```

Wartosc liczby = 398

a potem = 398

i jeszcze raz = 398

i tak samo jak na poczatku = 398

Aby kontynuować, naciśnij dowolny klawisz . . .

Funkcja fill - wypełnia „puste” miejsca określonymi znakami (domyślnie jest to spacja).

```
#include <iostream.h>
#include <iomanip.h>
int main()
{
    int x = 398;
    int stara_szerokosc;
    cout << "Wartosc liczby = " << x << endl;
    cout << "a potem = ";
    stara_szerokosc = cout.width(9);
    cout.fill('*');
    cout << x << endl;
    cout << "i jeszcze raz = ";
    cout.width(12);
    cout.fill('$');
    cout << x << endl;
    cout << "i tak samo jak na poczatku = ";
    cout.width(stara_szerokosc);
    cout << x << endl;
    system("PAUSE");
    return 0;
}
```

Wartosc liczby = 398

a potem = ***398**

i jeszcze raz = \$\$\$\$\$\$\$\$\$398

i tak samo jak na poczatku = 398

Aby kontynuować, naciśnij dowolny klawisz . . .

Funkcja precision – określa dokładność wypisywania liczb zmiennoprzecinkowych. Domyślnie jest to dokładność do 6 znaczących. Deklaracje tych funkcji w klasie ios mają postać:

```
int precision(int);  
int precision();
```

Przykład użycia:

```
#include <vcl.h>  
#include <iostream.h>  
int main()  
{  
    float x = 3.1234567891011, y = 199.12, z = 2;  
    cout << "Aktualna dokladnosc = " << (cout.precision()) << endl;  
    cout << "x = " << x << "   y = " << y << "   z = " << z << endl;  
    cout.precision(8);  
    cout << "A teraz dokladnosc = " << (cout.precision()) << endl;  
    cout << "x = " << x << "   y = " << y << "   z = " << z << endl;  
    system("PAUSE");  
    return 0;  
}
```

Aktualna dokladnosc = 6

x = 3.12346 y = 199.12 z = 2

A teraz dokladnosc = 8

x = 3.1234567 y = 199.12 z = 2

Aby kontynuować, naciśnij dowolny klawisz . . .

Ustawienie dokładności ma charakter trwały. Nieznaczące zera ani kropka dziesiętna nie są wypisywane. Taki sposób można zmienić modyfikując przykładowy program do postaci:

```
#include <iostream.h>
int main()
{
    float x = 3.1234567891011, y = 199.12, z = 2;
    cout << "Aktualna dokladnosc = " << (cout.precision()) << endl;
    cout << "x = " << x << "  y = " << y << "  z = " << z << endl;
    cout.setf(ios::showpoint);
    cout.precision(8);
    cout << "A teraz dokladnosc = " << (cout.precision()) << endl;
    cout << "x = " << x << "  y = " << y << "  z = " << z << endl;
    system("PAUSE");
    return 0;
}
```

Aktualna dokladnosc = 6

x = 3.12346 y = 199.12 z = 2

A teraz dokladnosc = 8

x = 3.1234567 y = 199.12000 z = 2.0000000

Aby kontynuować, naciśnij dowolny klawisz . . .