

Wykład 6: Obsługa wejścia – wyjścia.

dr inż. Andrzej Stafiniak

Wrocław 2023



HR EXCELLENCE IN RESEARCH



Politechnika Wroclawska

Urządzenia wejścia – wyjścia w języku C

- W ogólnym uproszczeniu programy komputerowe wykonują obliczenia. Obliczenia te wykonywane są na danych, które należy wprowadzić do komputera, natomiast wyniki obliczeń muszą zostać w jakiś sposób przekazane użytkownikowi.
- Dane te, zarówno wprowadzane do komputera jak i wychodzące z niego, obsługiwane są przez **urządzenia wejścia – wyjścia**.
- Standardowym **urządzeniem wyjściowym** może być np.: **ekran/terminal**, drukarka, ale też **plik**.
- Standardowym **urządzeniem wejściowym** może być np.: **klawiatura**, ale też **plik**.

Urządzenia wejścia – wyjścia w języku C

- Deklaracje większości funkcji obsługi **urządzeń wejścia/wyjścia** umieszczone są w pliku nagłówkowym **biblioteki standardowej** języka C - **stdio.h** (ang. *standard input/output*). Część funkcji (np. `getch()`, `putch()`) zadeklarowane są pliku **conio.h**, który nie jest częścią biblioteki standardowej.
- Jednym z dostępnych najczęściej stosowanych procedur obsługi operacji wejścia-wyjścia są **procedury strumieniowe** (czyli wysokiego poziomu, istnieją również procedury niskiego poziomu oraz obsługi portów we/wy).
- W **procedurach strumieniowych** pliki traktowane są jako ciąg, strumień bajtów, a obsługa operacji wejścia-wyjścia wymaga stosowania bufora. Buforowanie danych przy operacjach odczytu i zapisu przyspiesza działanie ponieważ zmniejsza ilość operacji na dysku.

Urządzenia wejścia – wyjścia w języku C

- Podczas uruchomienia programu otwieraną są predefiniowane strumienie/kanały (pliki) między innymi:
 - **stdin** - standardowy strumień wejścia, domyślnie standardowe urządzenie wejściowe wczytuje znaki z klawiatury, ale strumień **stdin** może ulec przekierowaniu np. na plik czyli umożliwia zacytanie z pliku lub innego urządzenia,
 - **stdout** - standardowy strumień wyjścia, domyślnie standardowe urządzenie wyjściowe wyświetla dane na ekran, ale strumień **stdout** może ulec przekierowaniu np. na plik czyli umożliwia zapisanie do pliku,
 - **stderr** - standardowy strumień wyjściowy do obsługi błędów, domyślnie wysyła informacje na ekran, możliwe przekierowanie na plik.

Urządzenia wejścia – wyjścia w języku C

- Podczas uruchomienia programu otwieraną są predefiniowane strumienie/kanały (pliki) między innymi:
 - **stdin** - standardowy strumień wejścia, domyślnie standardowe urządzenie wejściowe wczytuje znaki z klawiatury, ale strumień **stdin** może ulec przekierowaniu np. na plik czyli umożliwia zacytanie z pliku lub innego urządzenia,
 - **stdout** - standardowy strumień wyjścia, domyślnie standardowe urządzenie wyjściowe wyświetla dane na ekran, ale strumień **stdout** może ulec przekierowaniu np. na plik czyli umożliwia zapisanie do pliku,
 - **stderr** - standardowy strumień wyjściowy do obsługi błędów, domyślnie wysyła informacje na ekran, możliwe przekierowanie na plik.

wskazniki plikowe

Wykorzystując przekierowanie można wskazać inne pliki jako **pliki/strumienie standardowego wejścia/wyjścia**.

Funkcje obsługi standardowego wejścia-wyjścia

Funkcje obsługi plików standardowego wejścia-wyjścia w języku C możemy podzielić na dwie grupy:

➤ domyślne realizujące obsługę strumieni **stdin** (klawiatura) oraz **stdout** (ekran):

- we/wy znakowe (**getchar()** , **putchar()**)
- we/wy łańcuchowe (~~**gets()**~~ , **puts()**)
- we/wy sformatowane (**printf()** , **scanf()**)

➤ realizujące obsługę plików (dyskowe wejście-wyjście):

- we/wy łańcuchowe (**fgets()** , **fputs()**)
- we/wy sformatowane (**fprintf()** , **fscanf()**)
- we/wy rekordowe (**fread()** , **fwrite()**)
- we/wy znakowe (**getc()** , **putc()**)

Funkcje obsługi standardowego wejścia-wyjścia

Funkcje obsługi plików standardowego wejścia-wyjścia w języku C możemy podzielić na dwie grupy:

➤ domyślne realizujące obsługę strumieni `stdin` (klawiatura) oraz `stdout` (ekran):

- we/wy znakowe (`getchar()`, `putchar()`)
- we/wy łańcuchowe (~~`gets()`~~, `puts()`)
- we/wy sformatowane (`printf()`, `scanf()`)

Usunięte od C11

Ze względu że funkcja przyjmuje tylko jeden argument, adres bufora, istnieje możliwość przepełnienia bufora, lepiej skorzystać z funkcji `fgets()`.

➤ realizujące obsługę plików (dyskowe wejście-wyjście):

- we/wy łańcuchowe (`fgets()`, `fputs()`)
- we/wy sformatowane (`fprintf()`, `fscanf()`)
- we/wy rekordowe (`fread()`, `fwrite()`)
- we/wy znakowe (`getc()`, `putc()`)

np. `sizeof` bufor

```
char * fgets ( char * bufor, int size, FILE * stream );
```

np. `stdin`

Tablica char'ów realizująca bufor

Funkcja `printf()` – `stdio.h`

Nagłówek funkcji `int printf(const char *format, ...)`

- Funkcja wysyła sformatowany tekst na standardowym strumieniu wyjścia **`stdout`** czyli domyślnie ekran konsoli.
- Sformatowany tekst (tekst sterujący) jest to ciąg znaków (łańcuch) który może być uzupełniony **tagami**, zwanymi **specyfikatorami formatu**, umożliwiającymi wyświetlanie wartości (w sformatowany sposób) zmiennych lub stałych.
- Funkcja `printf()` to funkcja o nieokreślonej liczbie argumentów, których wartość może być wyświetlona zgodnie ze **specyfikatorem formatu**.

Funkcja printf() – stdio.h

Nagłówek funkcji `int printf(const char *format, ...)`

- Funkcja `printf()` jest typu `int` i zwraca liczbę wyświetlonych znaków w przypadku poprawnego jej wykonania oraz wartość ujemną w przypadku błędnego działania

```
# include <stdio.h>

int main () {

    int z=0;
    z=printf ("Ile to znakow? - ");
    printf("%d", z);
    return 0;
}
```

Ile to znakow? - 17

Funkcja printf() – stdio.h

Nagłówek funkcji `int printf(const char *format, ...)`

```
char znak = '!';
```

```
printf("Stop war %c%c%c", '!', 33, znak);
```

Stop war !!!



Specyfikator formatu – kod formujący, tag formujący.

Funkcja printf() – stdio.h

Nagłówek funkcji `int printf(const char *format, ...)`

```
char znak = '!';  
printf("Stop war %c%c%c", '!', 33, znak);
```

Stop war !!!

Tekst sterujący - jest to stała łańcuchowa zapisana między znakiem cudzysłowu, zawierająca znaki oraz specyfikatory formatu wyświetlające wartości kolejnych argumentów w zdefiniowany sposób.

Znaki specjalne:

`\n` – nowa linia LF

`\b` – backspace BS

`\t` – tabulacja pozioma HT

`\v` – tabulacja pionowa VT

`\r` – początek linii

`\f` – nowa strona

`\"`

`\'`

`\\`

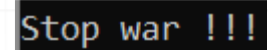
`\0` – znak zerowy

Funkcja printf() – stdio.h

Nagłówek funkcji `int printf(const char *format, ...)`

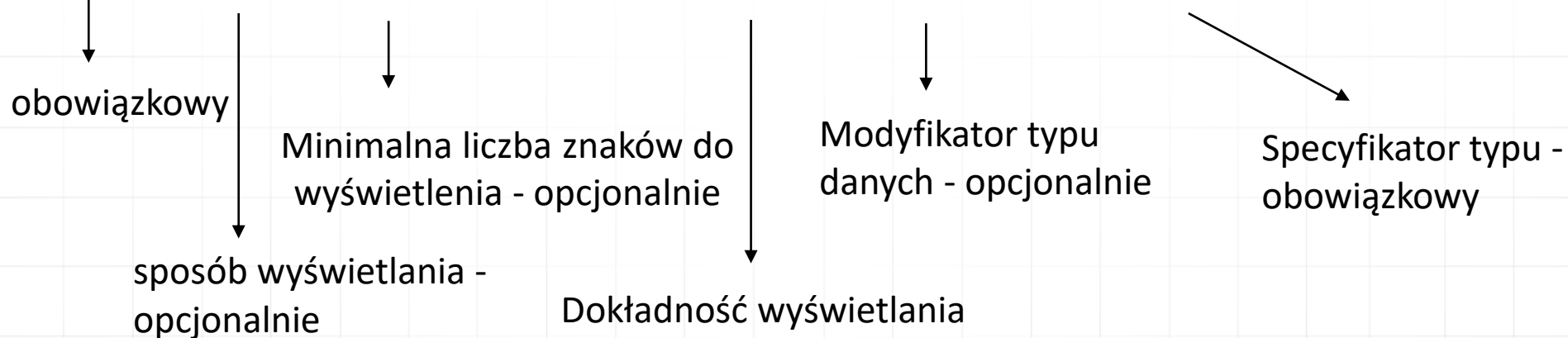
```
char znak = '!';
```

```
printf("Stop war %c%c%c", '!', 33, znak);
```



Specyfikator formatu – kod formujący, tag formujący. Konstrukcja:

`%[flags][width][.precision][length]specifier`



Funkcja printf() – stdio.h

Specyfikator formatu – kod formujący, tag formujący. Konstrukcja:

`%[flags][width][.precision][length]specifier`

Specyfikator typu -
obowiązkowy

%Specyfikator typu :

%c pojedynczy znak

%s łańcuch znaków

%d liczba dziesiętna ze znakiem

%f liczba zmiennoprzecinkowa (notacja dziesiętna)

%e liczba zmiennoprzecinkowa (notacja wykładnicza)

%g liczba zmiennoprzecinkowa (krótszy z formatów %f%e)

%u liczba dziesiętna bez znaku

%x liczba w kodzie szesnastkowym (bez znaku)

%o liczba w kodzie ósemkowym (bez znaku)

Funkcja printf() – stdio.h

Specyfikator formatu – kod formujący, tag formujący. Konstrukcja:

`% [flags] [width] [.precision] [length] specifier`

Flags

- Wyrównuje do lewej w ramach szerokości pola (do prawej jest domyślne)
- + Wymusza poprzedzanie wyświetlanej wartości znakiem + lub -, nawet dla liczb dodatnich. Domyślnie tylko liczby ujemne są poprzedzone znakiem.
- spacja Jeśli nie zostanie wyświetlony żaden znak, przed wartością wstawiana będzie spacja.
- # Użyta razem ze specyfikatorem o, x, X wyświetlana wartość poprzedzana jest odpowiednio 0, 0x, 0X dla wartości różnych od zera. Użyta razem z a, A, e, E, f, F, g lub G wymusza wyświetlenie kropki dziesiętnej, nawet jeśli nie ma więcej cyfr. Domyślnie, jeśli nie następują żadne cyfry, nie jest wyświetlana kropka dziesiętna.
- 0 Lewe dopełnianie liczby zerami (0) zamiast spacji, dla określonej szerokości pola.

Funkcja printf() – stdio.h

Specyfikator formatu – kod formujący, tag formujący. Konstrukcja:

`%[flags] [width] [.precision] [length]specifier`

width

(liczba)

Minimalna liczba znaków do wyświetlenia. Jeśli wartość do wyświetlenia jest krótsza niż ta **liczba**, wynik jest uzupełniany spacjami. Wartość nie jest obcinana, nawet jeśli jest większa od liczby.

Szerokość nie jest określona kodem formatu, ale jako dodatkowy argument wartości całkowitej poprzedzający argument, który ma być sformatowany.

Funkcja printf() – stdio.h

Specyfikator formatu – kod formujący, tag formujący. Konstrukcja:

`%[flags][width][.precision][length]specifier`

.precision

.liczba Dla specyfikatorów a, A, e, E, f, F liczba cyfr po kropce dziesiętnej do wyświetlenia; dla specyfikatorów d, i, o, u, x, X: liczba cyfr do wyświetlenia (dopełnij wartość do zadanej precyzji poprzedzającymi spacjami lub zerami w zależności od flagi)

.*

dokładność nie jest określona kodem formatu, ale jako dodatkowy argument wartości całkowitej poprzedzający argument, który ma być sformatowany.

Funkcja printf() – stdio.h

Specyfikator formatu – kod formujący, tag formujący. Konstrukcja:

`%[flags][width][.precision][length]specifier`

length - modyfikuje długość typu danych.

length	d i	u o x X	f F e E g G a A	c	s	p	n
(none)	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*
l	long int	unsigned long int		<u>wint_t</u>	wchar_t*		long int*
ll	long long int	unsigned long long int					long long int*
j	<u>intmax_t</u>	<u>uintmax_t</u>					<u>intmax_t</u> *
z	<u>size_t</u>	<u>size_t</u>					<u>size_t</u> *
t	<u>ptrdiff_t</u>	<u>ptrdiff_t</u>					<u>ptrdiff_t</u> *
L			long double				

Funkcja `scanf()` – `stdio.h`

Nagłówek funkcji `int scanf (const char *format, ...)`

- Funkcja `scanf()` pobiera dane z standardowego strumienia wejścia **`stdin`** i przekazuje je zgodnie z kodem formującym do lokalizacji wskazanych w liście argumentów.
- Jako argument, funkcja pobiera adres zmiennej do której chcemy przekazać dane. Adres może być przekazany za pomocą zmiennej **wskaźnikowej** lub za pomocą operatora poboru adresu **`&`**.
- Funkcja `scanf`, podobnie jak `printf()`, wykorzystuje **łańcuch sterujący**, w którym po zastosowaniu odpowiednich tagów – **specyfikatorów formatu** – odpowiednio formatuje wprowadzane dane do zmiennych.

Funkcja scanf() – stdio.h

Nagłówek funkcji `int` scanf(`const char` *format, ...)

```
int intValue = 0;
int *pointer = &intValue;
char word[10];
scanf("%d", &intValue);
scanf("%d", pointer);
scanf("%s", word);
```

Funkcja scanf() – stdio.h

Składnia specyfikatora formatu `%[*][width][length]specifier`

Specyfikator typu:

- `%c` pojedynczy znak, odczyta znaki białe
- `%s` łańcuch/ciąg znaków, dla znaków drukowanych oznacza to, że pobiera tylko słowa a nie całe zdania,
- `%d` liczba całkowita dziesiętna ze znakiem
- `%i` liczba całkowita dziesiętna ze znakiem, ale poprzedzona znakiem x0 interpretowana jako liczba w systemie szesnastkowym, a znakiem 0 jako w systemie ósemkowym,
- `%f` liczba zmiennoprzecinkowa (notacja dziesiętna)
- `%e, E` liczba zmiennoprzecinkowa (notacja wykładnicza z zapisem z e lub E)
- `%g` liczba zmiennoprzecinkowa (krótszy z formatów `%f` `%e`)
- `%u` liczba całkowita dodatnia dziesiętna bez znaku
- `%x, X` liczba całkowita w systemie szesnastkowym (bez znaku)
- `%o` liczba całkowita w systemie ósemkowym (bez znaku)

Funkcja scanf() – stdio.h

Składnia specyfikatora formatu `%[*][width][length]specifier`

* - zaczytana wartość ma zostać pominięta, nie będzie zapisana do kolejnego adresu

```
# include <stdio.h>

int main () {
    printf ("%#x\n", 16894);
    printf ("%+09.3f\n", 1.25);
    printf ("%s\n", " text tekst ");

    long long number=0;
    printf ("wprowadz liczbe: ");
    scanf ("%8lld", &number);
    printf ("8 cyfr tej liczby to: ");
    printf ("%d\n", number);
    return 0;
}
```

```
0x41fe
+0001.250
 text tekst
wprowadz liczbe: 1234567890
8 cyfr tej liczby to: 12345678
```

Obiektowe wejście wyjście – C++

- Język C++ posiada wsteczną kompatybilność za równo do poprzednich standardów jak i do języka C, dlatego w języku C++ możliwe jest korzystanie z funkcji `printf()` oraz `scanf()` na tych samych zasadach co w C.
- Do obsługi wejścia-wyjścia w języku C++ mamy dodatkowo do dyspozycji obiekty `std::cin` -obiekt klasy `istream` oraz `std::cout` obiekt klasy `ostream`. Klasy te są częścią zorientowanej obiektowo biblioteki `<iostream>` obsługi strumieni wejścia-wyjścia.
- Przekazywanie argumentów do strumienia wejściowego `cin` i wyjściowego `cout`, odbywa się z wykorzystaniem przeciążanych operatorów przesunięcia bitowego `<<` oraz `>>`.
- Za pomocą obiektu `std::endl` możemy przejść do nowej linii.

Obiektowe wejście wyjście – C++

- Przeciążanie operatora przesunięcia bitowego umożliwia zastosowanie wygodnego **wywołania łańcuchowego**.

```
#include <iostream>

int main () {
    int liczba ;
    std::cout <<"Wprowadz liczbe calkowita: ";
    std::cin >> liczba;
    std::cout << std::endl << "Podana liczba to: " << liczba << ", a jej kwadrat to:" << liczba*liczba
    << std::endl;
    return 0;
}
```

```
Wprowadz liczbe calkowita: 12
```

```
Podana liczba to: 12, a jej kwadrat to:144
```

Trochę więcej o C++

- Język C++ prowadzi pojęcie **przestrzeni nazw**. Udogodnienie to rozwiązuje problem konfliktu nazw w sytuacji dużych projektów.
- **Przestrzeń nazw** określa pewien obszar/zakres dla stałych, zmiennych oraz funkcji. Obiekty zadeklarowane w danej **przestrzeni nazw** nie kolidują z obiektami zadeklarowanymi w innej przestrzeni, mimo np. tego samego identyfikatora, nazwy.
- Przestrzeń nazw deklarujemy z wykorzystaniem słowa kluczowego **namespace**.
- Operator :: rozdzielczości zakresu – odwołanie do elementów przestrzeni nazw.

Trochę więcej o C++

➤ Przestrzeń nazw

```
#include <iostream>

namespace myHome{    // przestrzeń myRoom
    int sofa = 5;
}

namespace yourHome{
    namespace yourRoom{ // zagnieżdżanie przestrzeni yourRoom
        int sofa = 7;    // w przestrzeni yourHome
    }
}

int sofa = 1;        // globalna przestrzeń

int main() {
    int sofa = 2;    // brak kolizji
    std::cout << "Local sofa: " << sofa << std::endl;
    std::cout << "Global sofa: " << ::sofa << std::endl;
    std::cout << "My sofa: " << myHome::sofa << std::endl;
    std::cout << "Your sofa: " << yourHome::yourRoom::sofa << std::endl;
    return 0;
}
```

```
Local sofa: 2
Global sofa: 1
My sofa: 5
Your sofa: 7
```

Trochę więcej o C++

- Wraz z rozwojem **języka C** oraz ogólnie programowania pojawia się problem nazewnictwa funkcji, funkcje muszą mieć unikalny, niepowtarzalny identyfikator.
- W **języku C++** obiekty, jak funkcje, klasy czy szablony z biblioteki standardowej zostały zadeklarowane w przestrzeń **std**.
- Ze względu na kompatybilność wsteczną **C++**, elementy klasycznego **C** nie wymagają stosowania konkretnej przestrzeni.

```
int main() {  
    int sofa = 2;    // brak kolizji  
    std::cout << "Local sofa: " << sofa << std::endl;  
    std::cout << "Global sofa: " << ::sofa << std::endl;  
    std::cout << "My sofa: " << myHome::sofa << std::endl;  
    std::cout << "Your sofa: " << yourHome::yourRoom::sofa << std::endl;  
    return 0;  
}
```

Trochę więcej o C++

- Alternatywą do odwołania się do obiektów z pomocą nazwy **przestrzeni** oraz operatora `::` (np. `std::`) jest wykorzystanie **deklaracji (dyrektywy) `using`**.
- **Deklaracja `using`** (np. `using std::cout;`) - importuje pojedynczą nazwę/obiekt z zadanej **przestrzeni nazw** do danego **obszaru deklaracyjnego** (bloku `{}`).

```
int main() {  
    using std::cout;  
    int sofa = 2;    // brak kolizji  
    cout << "Local sofa: " << sofa << std::endl;  
    cout << "Global sofa: " << ::sofa << std::endl;  
    cout << "My sofa: " << myHome::sofa << std::endl;  
    cout << "Your sofa: " << yourHome::yourRoom::sofa << std::endl;  
    return 0;  
}
```

Trochę więcej o C++

- Alternatywą do odwołania się do obiektów z pomocą nazwy **przestrzeni** oraz operatora `::` (np. `std::`) jest wykorzystanie **deklaracji (dyrektywy) `using`**.
- **Dyrektywa `using`** (np. `using namespace std;`) - importuje wszystkie nazwy z zadanej **przestrzeni nazw** do danego **obszaru deklaracyjnego** (bloku `{}`).

```
int main() {  
    using namespace std;  
    int sofa = 2;    // brak kolizji  
    cout << "Local sofa: " << sofa << endl;  
    cout << "Global sofa: " << ::sofa << endl;  
    cout << "My sofa: " << myHome::sofa << endl;  
    cout << "Your sofa: " << yourHome::yourRoom::sofa << endl;  
    return 0;  
}
```

Trochę więcej o C++

- Alternatywą do odwołania się do obiektów z pomocą nazwy **przestrzeni** oraz operatora `::` (np. `std::`) jest wykorzystanie **deklaracji (dyrektywy) `using`**.
- **Dyrektywa `using`** (np. `using namespace std;`) - importuje wszystkie nazwy z zadanej **przestrzeni nazw** do danego **obszaru deklaracyjnego** (bloku `{}`).

```
int main() {  
    using namespace std;  
    int sofa = 2;    // brak kolizji  
    cout << "Local sofa: " << sofa << endl;  
    cout << "Global sofa: " << ::sofa << endl;  
    cout << "My sofa: " << myHome::sofa << endl;  
    cout << "Your sofa: " << yourHome::yourRoom::sofa << endl;  
    return 0;  
}
```

- **Uwaga!** Importowanie nazw, zwłaszcza całych przestrzeni, może prowadzić do **kolizji nazw**. Dyrektywy `using` nie stosujemy w plikach nagłówkowych.

Obiektowe wejście wyjście – C++

Obsługa strumienia `cout` - sterowanie formatem

- Sterowanie formatem odbywa się za pomocą zmian statusu **flag stanu formatowania**. Flagi te zawarto w **klasie ios**.
- Włączenie lub wyłączenie flag: `cout.setf(ios::flaga)`
`cout.unsetf(ios::flaga)`
- Wywołanie **metody** (funkcji z klasy ios) na obiekcie: `cout.metoda()`

Obiektowe wejście wyjście – C++

Obsługa strumienia cout - sterowanie formatem

➤ Ustawianie szerokości pola oraz justowanie

```
#include <iostream>

int main(){
    int liczba = - 123;
    std::cout << liczba << std::endl;

    std::cout.width( 12 );
    std::cout << std::left << liczba << std::endl;

    std::cout.width( 12 );
    std::cout << std::right << liczba << std::endl;

    std::cout.width( 12 );
    std::cout << std::internal << liczba << std::endl;

    return 0;
}
```

```
-123
-123
      -123
      123
```

Obiektowe wejście wyjście – C++

Obsługa strumienia cout - sterowanie formatem

➤ Ustawianie systemu liczbowego

```
//ustawienie systemu liczbowego
```

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

```
std::cout << liczba << std::endl;
```

```
std::cout.setf(std::ios::uppercase);
```

```
std::cout << liczba << std::endl;
```

```
std::cout.setf(std::ios::showbase);
```

```
std::cout << liczba << std::endl;
```

```
std::cout.setf(std::ios::oct, std::ios::basefield);
```

```
std::cout << liczba << std::endl;
```

```
std::cout.setf(std::ios::dec, std::ios::basefield);
```

```
std::cout << liczba << std::endl << std::endl;
```

```
7b  
7B  
0X7B  
0173  
123
```


Obiektowe wejście wyjście – C++

Obsługa strumienia cout - sterowanie formatem

➤ Ustawianie systemu liczbowego

```
float liczba1 = 1234;  
float liczba2 = -0.00789;  
std::cout.setf(std::ios::showpos);  
std::cout << liczba1 << std::endl;  
  
std::cout.setf(std::ios::showpoint);  
std::cout << liczba1 << std::endl;  
  
std::cout.setf(std::ios::fixed, std::ios::floatfield);  
std::cout << liczba1 << std::endl << std::endl;  
  
std::cout.setf(std::ios::scientific, std::ios::floatfield);  
std::cout << liczba1 << std::endl;  
std::cout << liczba2 << std::endl;  
  
std::cout.precision(4);  
std::cout << liczba1 << std::endl;  
std::cout << liczba2 << std::endl << std::endl;  
  
std::cout.setf(std::ios::fixed, std::ios::floatfield);  
std::cout << liczba1 << std::endl;  
std::cout << liczba2 << std::endl;
```

```
+1234  
+1234.00  
+1234.000000  
  
+1.234000E+003  
-7.890000E-003  
+1.2340E+003  
-7.8900E-003  
  
+1234.0000  
-0.0079
```

Obiektowe wejście wyjście – C++

Obsługa strumienia cout - sterowanie formatem

➤ Wypełnienie i szerokość pola

```
std::cout.width(12);  
std::cout.fill('*');  
std::cout << liczba1 << std::endl;  
std::cout.width(15);  
std::cout.fill(' ');  
std::cout << liczba2 << std::endl;
```

```
+**1234.0000  
-_____0.0079
```

Obiektowe wejście wyjście – C++

Obsługa strumienia cout – manipulatory bezargumentowe

```
#include <iostream>

int main(){
    long liczba = 123;
    float liczba1 = 1234;
    float liczba2 = -0.00789;
    std::cout << liczba1 << std::endl;
    std::cout << liczba2 << std::endl << std::endl;

    // left, right, internal, - wyrównanie danych
    std::cout.width(12);
    std::cout << std::right << liczba1 << std::endl << std::endl;

    // fixed, scientific – formatowanie liczb zmiennopozycyjnych,
    std::cout << std::scientific;
    std::cout << liczba1 << std::endl;
    std::cout << liczba2 << std::endl;
    std::cout << std::fixed;
    std::cout << liczba1 << std::endl;
    std::cout << liczba2 << std::endl << std::endl;

    //hex, oct, dec, showbase, noshowbase, showpoint, noshowpoint – pokaż baze
    std::cout << std::hex << liczba << std::endl;
    std::cout << std::showbase << liczba << std::endl;

    std::cout << std::oct << liczba << std::endl;
    std::cout << std::noshowbase << liczba << std::endl;

    // endl, ends -> \0, flush

    return 0;
}
```

```
1234
-0.00789

      1234

1.234000e+003
-7.890000e-003
1234.000000
-0.007890

7b
0x7b
0173
173
```

Obiektowe wejście wyjście – C++

Obsługa strumienia cout – manipulatory parametryczne

```
//#include <iomanip>
std::cout << std::setw( 6 ) << liczba << std::endl << std::setw( 9 ) << std::setfill( '*' )
    << liczba << std::endl << std::setw( 12 ) << std::setfill( 'x' ) << liczba << std::endl;

std::cout << std::setprecision( 2 ) << liczba2 << " " << std::setprecision( 4 ) << liczba2
    << " " << std::setprecision( 8 ) << liczba2 << std::endl;
```

```
173
*****173
xxxxxxxxxx173
-0.01  -0.0079  -0.00789000
```