

Wykład 1

- **Biblioteki standardowe do obsługi we/wy w C++**
- **Strumienie**
- **Operatory `>>` i `<<`**
- **Domniemania w pracy strumieni predefiniowanych**
- **Priorytet operatorów `>>` i `<<`**

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 `iostream`) 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 (odpowiadającym różnym urządzeniom). Na przykład składnia operacji czytania z i pisanie 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 << endl;`

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

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