Wejście-wyjście w C++
Podstawy operacji

- **C++ wspiera dwie kompletne systemy We/Wy:**
  - **We\Wy języka C**
  - **We\Wy obiektowy – C++**

- **Główną zaletą We\Wy C++ : jest możliwe przeciążenie dla tworzonych klas. Nadaje to możliwość wbudowywać w systemy We\Wy tworzone przez programista typy danych.**

- **Strumień – to urządzenie logiczne, produkujące lub pobierajjące informacje.**

- **Wszystkie strumieni We/Wy zachowują się tak samo – system We/Wy daje ten samy interfejs dla różnych urządzeń fizycznych. To oznacza, na przykład, że ta sama funkcja będzie wykonywała wyprowadzenie na monitor, w plik, na drukarkę, … .**

- **Przy starcie programu w języku C pozostają otworzone potoki stdin, stdout, stderr.**

- **W języku C++:**

Strumień	Znaczenie	Urządzenie domyślne
cin	Standardowe wejście	Klawiatura
cout	Standardowe wyjście	Ekran
cerr	Standardowe wyjście dla błędów	Ekran
clog	Buforowana wersja cerr	Ekran

- **Standard C++ tworzy również potoki dodatkowe wcin, wcout, wcerr, wclog – dla wide text format (wchar_t) (16-bitiwych) symbolów.**

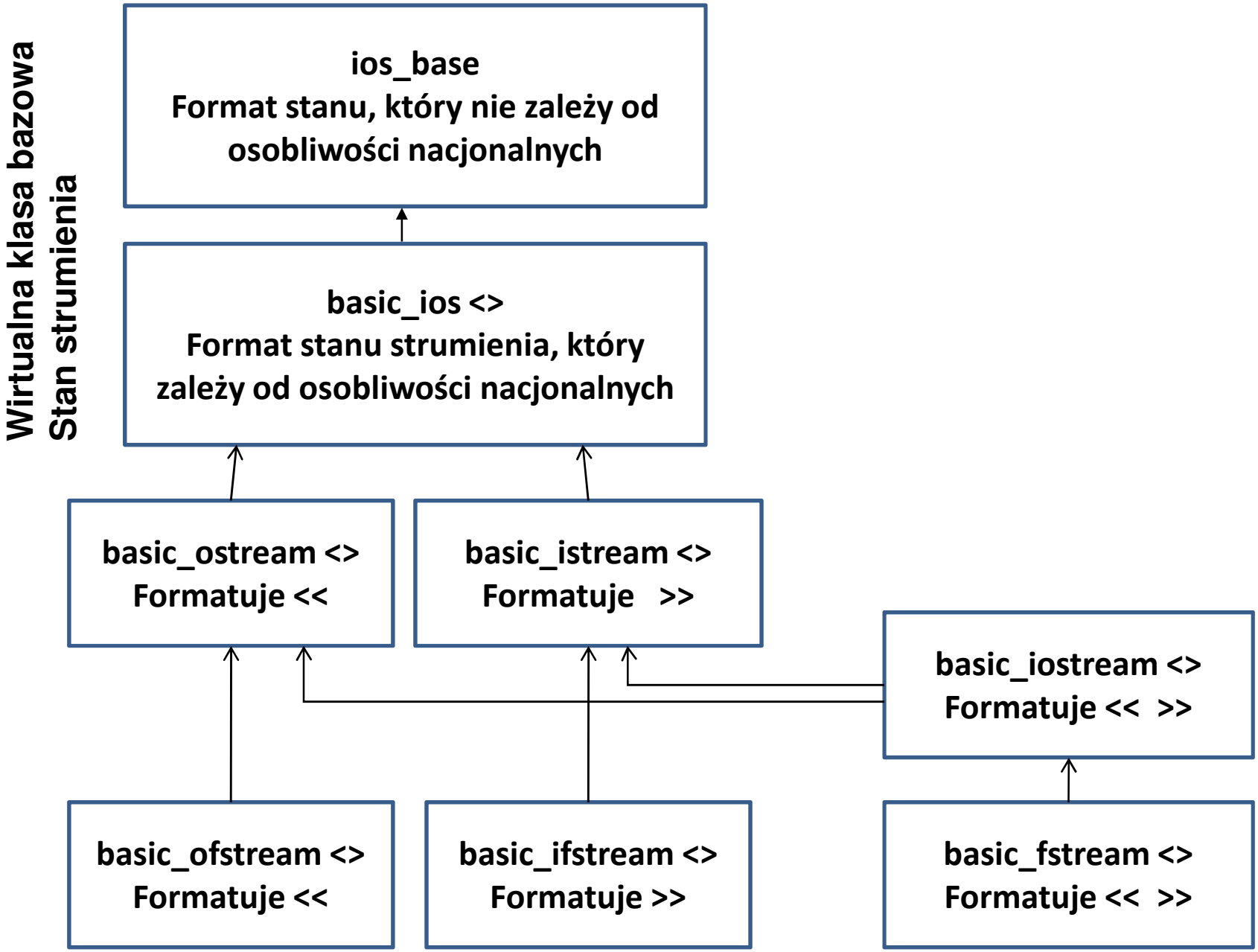
- **System We\Wy bazuje się na dwóch różnych hierarchiach klas-szablonów.**

- **Pierwsza hierarchia klas jest pochodną od klasy dolnego poziomu *basic_streambuf*. Ten klas jest bazowym dla operacji We\Wy dolnego poziomu i wspiera cały system We\Wu.**

- **Druga hierarchia klas jest pochodną od klasy *basic_ios*. To jest klasa bazowa dla operacji We\Wy górnego poziomu, który zabezpiecza formatowanie, kontrole błędów i informacje o stanie strumienia.**

- **Biblioteka We\Wy tworzy dwie rozdzielne wersje hierarchii klas – jedną – 8-bitiwą, drugą – 16-bitową.**

Klasa - szablon	Klasa dla wersji 8-bitowej
basic_streambuf	streambuf
basic_ios	ios
basic_istream	istream
basic_ostream	ostream
basic_iostream	iostream
basic_fstream	fstream
basic_ifstream	ifstream
basic_ofstream	ofstream


Wirtualna klasa bazowa
Stan strumienia
ios_base
Format stanu, który nie zależy od osobliwości nacjonalnych
basic_ios <>
Format stanu strumienia, który zależy od osobliwości nacjonalnych
basic_ostream <>
Formatuje <<
basic_istream <>
Formatuje >>
basic_iostream <>
Formatuje << >>
basic_ofstream <>
Formatuje <<
basic_ifstream <>
Formatuje >>
basic_fstream <>
Formatuje << >>

- **Dla dostępu do poważnej klasy ios trzeba dołączyć nagłówek <iostream>**

Formatowanie operacji Wejścia \ Wyjścia

- **Formatowanie We\Wy można dokonywać:**
  - **przy użyci składowych klasy ios**
  - **przy wykorzystani manipulatorów – specjalnych funkcji, które można umieszczać w wyrażeniach dotyczącycz We\Wy**

Formatowanie przy użyci składowych ios

- **Z każdym strumieniem związany jest zbiór flag formatowania, kontrolujących sposób formatowania informacji. To są maski bitowe.**

- **W klasie ios zadekłarowana wyliczeniowa maska bitowa *fmtflags*: (class *ios_base*, members – patrz MSDN)**

Enumeration *fmtflags*:

adjustfield	floatfield	right	skipws
basefild	hex	scientific	unitbuf
boolalpha	internal	showbase	uppercase
dec	left	showpoint	
fixed	oct	showpos	

flaga	ustawiona	wyzerowana
skipws	Przy wyprowadzeniu do strumienia wiodące znaki puste (spacje, tabulatory, nowej linii) są ignorowane	nie są ignorowane
left	Wyrównanie po lewej krawędzi	Jeśli wszystkie te flagi są ignorowane – ustawienia domyślne
right	Wyrównanie po prawej krawędzi	
internal	Wartości numeryczne są dopełniane tak, by zajęły całe pole – wstawiane spacje pomiędzy cyframi i znakiem liczby	

flaga	ustawiona	wyzerowana
oct	wyświetlanie wartości w systemie ósemkowym	domyślnie - syst. dziesiętnym
hex	w systemie szesnastkowym	
dec	przywrócenie do systemu dziesiętnego	----------
showbase	pokazać podstawą systemu liczenia	nie pokazywać
uppercase	w notacji naukowej wyświetlamy ‘E’, syst. Szesnastkowym – ‘X’	domyślnie: ‘e’, ‘x’
showpos	znak ‘+’ jest wyświetlany	nie wyświetlany
showpoint	wyświetla wszędzie przecinek i zero końcowe	nie wyświetla
scientific	wartości zmienno-przecinkowe są wyświetlane w notacji naukowej 3.141592e+00	kompilator sam wybiera odpowiednie notacje
fixed	wartości zmienno-przecinkowe są wyświetlane w notacji zwykłej: 3.141592	
unitbuf	bufor jest opróżniany (flush) po każdej operacji insertion (wyprowadzenie w strumień)	nie
boolalpha	wartości logiczne mogą być wprowadzone i wyprowadzone jako *false, true*	liczbami 0, 1
basefield	dec | hex | oct	nie

flaga	ustawiona	wyzerowana
adjustfield	internal | left | right	nie
floatfield	fixed | scientific	nie

Funkcji-składowe klasy ios_base:

fmtflags setf(fmtflags flagi)	Zwraca poprzednie ustawienie flag i ustawia wymienione na liście argumentów. stream.setf(ios::showpos | ios::scientific); stream – strumień, do którego należy ustawienie flag, jest obiektem klasy pochodnej od klasy ios_base. W C++ nie istnieje formatowania globalnego, każde formatowanie odbywa się dla poszczególnych strumieni. showpos, scientific – to są wartość wyliczeniowa w klasie ios, dla tego operator zakresu :: jest konieczny, inaczej kompilator „nie widzi" tych wartości.
void unsetf(fmtflags flagi)	Flagi, wymienione na liście *flagi*, są wyzerowane. Wszystkie pozostałe flagi - bez zmian.

fmtflags flags();	**Zwraca bieżący stan flag, nic nie zmienia w ustawieni flag** **ios::fmtflags f = flags();**
fmtflags flags(fmtflags *f*);	**Ustawia wszystkie flagi, zdefiniowane w szablonie fmtflags *f,* zwraca poprzednie ustawienie flag** **ios::fmtflags f = ios::showpos | ios::showbase;** **cout.flags(f);**
streamsize width(streamsize w);	**domyślnie przy wyprowadzeni wartość zajmuje tyle pozycji, ile ma symbolów. Funkcje width() zadaje minimalną szerokość pola w, zwraca poprzednią szerokość pola. Po wykonaniu każdego wyprowadzenia szerokość pola pozostaje ustawiona jako wartość domyślna. Reszta pola (nie zajęta symbolami) będzie wypełniona symbolem zapełnienia (domyślnie – spacja). Jeśli ilość symbolów przekracza ustawioną szerokość pola, będzie wyprowadzone tyle symbolów, ile ma wartość.**
streamsize precision(streamsize p);	**Domyślnie liczby zmiennoprzecinkowe są wyprowadzone z dokładnością 6 cyfr. Dokładność można zmienić przy pomocy funkcji precision(). Zwraca poprzednią dokładność.**

Funkcja fill() – składowa klasy pochodnej basic_ios:
template <class Elem**, class** Traits**> class basic_ios : public ios_base**

char fill(char ch);	Domyślnie wypełnienie pustych pozycji odbywa się spacjami. Funkcje fill() nadaje możliwość wypełnić te pozycji symbolem ch

- **Przykład W11**

Manipulatory We \ Wy

- **Manipulatory – to funkcje specjalne, które można włączać do wyrażeń We \ Wy. Dla dostępu k manipulatorom z parametrami trzeba dodać nagłówek <iomanip>. Standardowe manipulatory:**

manipulator	przeznaczenie	We \ Wy
boolalpha	Włącza flage boolalpha	We \ Wy
dec	Włącza flage dec	We \ Wy
endl	Wyprowadza znak nowej linii i opróżnia strumień	Wy
ends	Wyprowadza znak null	Wy
fixed	Włącza flage fixed	Wy
flush	Opróżnia strumień	Wy

manipulator	przeznaczenie	We \ Wy
hex	Włącza flagę hex	We \ Wy
internal	Włącza flagęinternal	Wy
left	Włącza flagę left	Wy
noboolalpha	Wyłącza flagę boolalpha	Wy
noshowbase	Wyłącza flagę showbase	Wy
noshowpoint	Wyłącza flagę showpoint	Wy
noskipws	Wyłącza flagę skipws	Wy
nounitbuf	Wyłącza flagę unitbuf	Wy
nouppercase	Wyłącza flagę uppercase	Wy
oct	Włącza flagę oct	We \ Wy
resetiosflags(fmtflags f)	Wyłącza flagi wymienione w f	Wy
right	Włącza flagę right	Wy
scientific	Włącza flagę scientific	Wy
setbase(int base)	Przypisuje podstawie systemu liczenia wartość, wskazaną w base	We \ Wy
setfill(int ch)	Definiuje jako znak wypelnienia ch	Wy
setiosflags(fmtflags f)	Włącza flagi wymienione w f	We \ Wy

manipulator	przeznaczenie	We \ Wy
setprecision(int p)	Ustala wartość dokładności	Wy
setw(int w)	Definiuje szerokość pola jako w	Wy
showbase	Włącza flagę showbase	Wy
showpoint	Włącza flagę showpoint	Wy
showpos	Włącza flagę showpos	Wy
skipws	Włącza flagę skipws	Wy
unitbuf	Włącza flagę unitbuf	Wy
uppercase	Włącza flagę uppercase	Wy
ws	Opuszcza puste znaki wiodące	We

- **Przykład:**

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

int main()
{
    cout << hex << 100  << endl;
    cout << setfill('?') << setw(10) << 2343.0;
}
wydruk:  64
         ??????2343
```

- **Manipulatory występują w wyrażeniach złożonych i często okazują się bardziej przydatne dla zastosowania, niż składowe klasy ios.**

- **Jeśli manipulator nie pobiera argumenty (na przykład, endl, hex), nie występują nawiasy. Dzieje się tak dla tego, że jest to adres funkcji przekazywany do przeciążonego operatora << .**

- **Manipulatory We \ Wy wpływają tylko na strumień, w wyrażenie dla którego są włączone. Na inne strumienie, otwarte w programie, nie wpływają.**

- **Przykład W11_1**

Tworzenie własnego insertera – przeciążenie operatora <<

- **Operator wyprowadzenia wstawia dane w strumień – stąd i pochodzi nazwa inserter (insertion operator)**

```
ostream &operator << (ostream &stream, typ_klasy ob)
{
    //ciało funkcji
    return stream;
}
```

- **Operator << powinien być włączany w kolejką: stream << ob**

- **Operator << ma dwa operandy – strumień i obiekt klasy, przy czym lewostronni operand – to strumień, prawostronni – obiekt klasy lub referencje do obiektu klasy**

```
ostream &operator << (ostream &stream, const typ_klasy &ob)
```

- **Operator-funkcje nie może być składową klasy, dla którego jest tworzona, dla tego że włączanie w kolejkę stream << ob wymaga, że lewostronni operand powinien być strumieniem.**

- **Wartość zwracana – referencje do strumienia stream. To jest koniecznie dla tego, żeby można było włączać w wyrażenie stream << ob1 << ob2;**

- **Najczęściej przeciążona funkcje-operator << występuje jako funkcje zaprzyjaźniona do klasy, obiekt której ma wyprowadzić w strumień. To nadaje możliwość dostępu do ukrytych składowych klasy.**

Tworzenie własnego ekstraktora – przeciążenie operatora >>

```
istream &operator >> (istream &stream, typ_klasy &ob)
{
        //ciało extractora
        return stream;
}
```

- **Operator >> powinien być włączany w kolejką: stream >> ob dla tego zwraca referencje do strumienia stream.**

- **Operator >> ma dwa operandy – strumień i referencje do obiektu klasy, przy czym lewostronni operand – to strumień, prawostronni – referencje do obiektu klasy. Sprawa w tym, że przy wykonaniu wejścia wartości składowych klasy prawdopodobnie będą zmienione.**

- **Przykład:**

```
class mcoord
{
protected:
        double x;
        double y;
public:
        mcoord(double xx, double yy) { x = xx; y = yy; }
        mcoord() {x=0; y=0; }
};

class node : public mcoord
{
        int numb;
        char str[512];
public:
        node(int nb, char *st, double xx, double yy);
        ~node() {}
……..
        friend ostream & operator << (ostream &stream, const node &ob);
        friend istream & operator >> (istream &stream, node &ob);
};
```

```
istream & operator >> (istream &stream, node &ob)
/*=========================================================
Ekstraktor – przeciążenie operatora >> dla obiektów klasy node
=========================================================*/
{
        stream >> ob.x >> ob.y >> ob.numb >> ob.str;
        return stream;
}

ostream & operator << (ostream &stream, const node &ob)
/*=========================================================
Inserter – przeciążenie operatora << dla obiektów klasy node
=========================================================*/

{
        stream << ob.x << "  " << ob.y << "  " << ob.numb << "  " << ob.str << endl;
        return stream;
}
```

Tworzenie własnych manipulatorów

- **Pozwala konsolidować sekwencje kilku oddzielnych operacji wejścia-wyjścia w jeden manipulator**

- **Pozwala wielokrotnie powtarzalne operacji przy We\Wy „zapakować" w jeden manipulator, a dalej używać jego tyle razy, ile to jest potrzebne**

- **Istnieją dwa typy manipulatorów:**
  - **operujących na strumieniach wejściowych**
  - **operujących na strumieniach wyjściowych**

- **Każdą z tych grup można podzielić na manipulatory, pobierające argumenty, i manipulatory niepobierające argumenty. Tworzenie manipulatorów sparametryzowanych istotnie zależy od kompilatora. Dla tego rozważymy tworzenie manipulatorów niesparametryzowanych.**

- **Szkielet manipulatorów wyjścia:**

```
ostream  & nazwa_manipulatora(ostream  &stream)
{
        //kod
        return stream;
}
```

- **Zwraca referencje do strumienia typu ostream. Wykorzystuje się jako element większego wyrażenia wyjścia (stream << … << nazwa_manip << ..). Posiada jeden argument (referencje do strumienia, na którym operuje), przy wykorzystaniu nie jest używany żaden argument.**

```
ostream & manip_a(ostream &stream)
{
        stream.setf(ios::showbase);
        stream.setf(ios::hex | ios::basefield);
        return stream;
}

int main()
{
        cout << 256 << " " << manip_a << 256 << endl;

        system("pause");
        return 0;
}
```

```
256 0x100
Для продолжения нажмите любую клавишу . . .
```

➢ Szkielet manipulatorów wejścia:

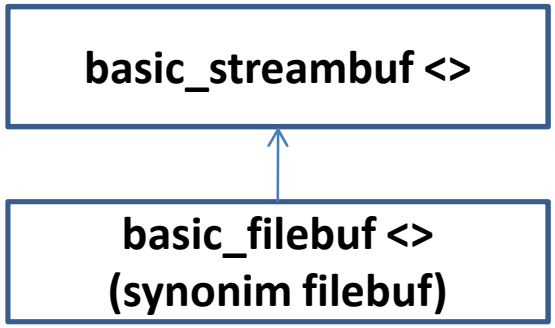
```
istream &nazwa_manipulatora(istream &stream)
{
        //kod
        return stream;
}
```

```
#include "stdafx.h"
#include "windows.h"
#include <iostream>
using namespace std;

istream & manip_b(istream &stream)
{
        MessageBeep(-1);                                    //dzwięk
        cout << "wprowadź  i wciśnij  ENTER\n";
        return stream;
}

int main()
{
        int i;
        cin >> manip_b >> i;

        system("pause");
        return 0;
}
```

Operacje Wejścia / Wyjścia, przeprowadzane na plikach

- **Pliki dyskowe mają wielu unikalnych cech, dla tego w układzie We\Wy C++ pozostało tworzone klasy specjalne dla pracy z plikami: ofstream, ifstream, fstream (stor. 4) – hierarchie klas. Nagłówek <fstream> .**

- **Dla zarządzania nizkopoziomowego strumieniami dla plików istnieje klasa szablonowa basic_filebuf**



- **Otwieranie i zamykanie pliku. Otwieranie pliku polega na połączeniu go ze strumieniem. Są trzy rodzaje strumieni:**
  - **wejściowe (obsługiwany przez klasę ifstream)**
  - **wyjściowy (ofstream)**
  - **wejście\wyjście (fstream)**
- **Otwieranie pliku powstaje przy wywołaniu konstruktora odpowiedniej klasy lub przy jawnym użyciu metody klasy open(…)**
- **Przykład W11_4 – otwieranie pliku przy wywołaniu konstruktora klasy**
- **Metody open:**

void basic_ifstream::open(const char *_Filename, ios_base::openmode _Mode = ios_base::in, int _Prot = (int)ios_base::_Openprot);

void basic_ofstream::open(const char *_Filename, ios_base::openmode _Mode = ios_base::out, int _Prot = (int)ios_base::_Openprot);

void basic_fstream::open(const char *_Filename, ios_base::openmode _Mode = ios_base::in | ios_base::out, int _Prot = (int)ios_base::_Openprot);

- **Tryb otwierania pliku zależy od wartości** ios_base::openmode _Mode.

typedef T3 openmode;
static const openmode app, ate, binary, in, out, trunc;

ios_base::app	**Przed każdym zapisem w plik wskaźnik pozycji pliku będzie przestawiony na koniec pliku (tryb dodawania w koniec pliku)**
ios_base::ate	**Przy otwierani pliku wskaźnik pozycji będzie przestawiony na koniec pliku, przecież zapis \ odczyt może być dokonywany w dowolnym miejscu pliku**
ios_base::binary	**Tryb binarny. Domyślnie pliki są otwarte w trybie tekstowym. Plik, otwarty w trybie tekstowym, „widzi" znaki formatowania i znaki specjalne. Plik, otwarty w trybie binarnym, „nie widzi" formatowania.**
ios_base::in	**Plik jest otwarty dla odczytu (extraction)**
ios_base::out	**Plik jest otwarty dla zapisu (insertion)**
ios_base::trunc	**Zawartość pliku istniejącego pozostaje zniszczona, plik w trakcie otwierania będzie ucięty do zerowej długości**

basic_filebuf::open

ios_base::in	becomes "r" (open existing file for reading).
ios_base::out ios_base::out | ios_base::trunc	becomes "w" (truncate existing file or create for writing).
ios_base::out | ios_base::app	becomes "a" (open existing file for appending all writes)
ios_base::in | ios_base::out	becomes "r+" (open existing file for reading and writing).
ios_base::in | ios_base::out | ios_base::trunc	becomes "w+" (truncate existing file or create for reading and writing).
ios_base::in | ios_base::out | ios_base::app	becomes "a+" (open existing file for reading and for appending all writes).

_Prot - the default file opening protection.

- **Sprawdzenie poprawności otwarcia pliku.**

  - **W przypadku niepowodzenia strumień jest ustawiony na false:**

```
fstream iof("myfile.dat", ios_base::in | ios_base::out);
if(! iof)
{
        //plik nie pozostał otwarty
}
//OK
```

  - **metoda** bool basic_filebuf::is_open**(): zwraca true, jeśli plik pozostał otwarty, false – niepowodzenie.**

```
fstream iof("myfile.dat", ios_base::in | ios_base::out);
if(!iof.is_open())
{
        //plik nie pozostał otwarty
}
//OK.
```

- **Zamykanie plików:** void basic_fstream :: close( );

Niesformatowane i binarne operacje wejścia-wyjścia

- **W funkcjach klasycznych We\Wy C\C++ pozostałe zrealizowane operacje na bajtach. Dla tego że sizeof(char) = 1 B, w prototypach tych funkcji jest używany typ char. Z wprowadzeniem Unicode (wchar_t) sytuacje się zmieniła. Ale w obrębie tego kursu będziemy i dalej posługiwali funkcjami klasycznymi.**

- **Pliki będziemy otwierać w trybie binarnym (ios_base::binary).**

- **Funkcje C++ Run-time library *put, get* dla plików binarnych powodują zapis i odczyt jednego bajta informacji:**

```
istream &get(char &ch);
ostream &put(char ch);
```

- **Przykład W11_5 !!!**

- **Funkcji C++ Run-time library read, write dla zapisu \ odczytu bloków**

```
basic_istream& read( char_type *_Str, streamsize _Count );
basic_ostream& write( const char_type *_Str, streamsize _Count );
```

- **Jeśli chodzi o plikach binarnych, to** _Str **jest wskaźnikiem do bufora pamięci rzutowany na typ char * lub const char * , a** _Count **podaje ilość bajtów, które trzeba zapisać \ przeczytać.**

- **Jeśli przy odczycie koniec pliku występuje wcześniej niż będzie przeczytane** _Count **bajtów, odczyt pozostaje przerwany, a buforze** _Str **znajduje się tyle bajtów, ile faktycznie pozostało przeczytane. Dla tego żeby wyjaśnić, ile bajtów pozostało przeczytane, istnieje funkcja**

```
streamsize basic_istream :: gcount( ) const;
```

- **Dla wyładowania buforu systemowego We\Wy w plik, używamy**

```
basic_ostream&  basic_ostream :: flush( );
```

Nie używamy fleszowanie pliku bez potrzeby – zwalnia to istotnie działanie programu. Przy zamykaniu pliku powstaje automatyczne wyładowanie danych z bufora systemowego i ich zapis do pliku.

- **Dostęp swobodny (bezpośredni). W C++ plik ma 2 wskaźnika pozycji – jeden określa pozycje pliku, gdzie nastąpi kolejna operacje wejścia, a drugi – wyjścia. Dla przesunięcia wskaźników pozycji plików mamy 2 układy funkcji: xxxg() (get - odczyt), xxxp() (put – zapis).**

- **Przesuwa wskaźnik pozycji pliku na wartość** _Off **odnośnie** _Way **:**
  - _Way == ios_base::beg **– odnośnie początku pliku**
  - _Way == ios_base::end **– odnośnie końca pliku**
  - _Way == ios_base::cur **– odnośnie bieżącej pozycji**

```
basic_istream& basic_istream::seekg( off_type _Off, ios_base::seekdir _Way );
basic_ostream& basic_ostream:: seekp( off_type _Off, ios_base::seekdir _Way );
```

- **Ustawia wskaźnik pozycji pliku w pozycje, podaną przez** _Pos:

```
basic_istream& basic_istream::seekg( pos_type _Pos );
basic_ostream& basic_ostream:: seekp( pos_type _Pos );
```

- **Informuje, w której pozycji znajduje się wskaźnik pozycji pliku:**

```
pos_type  basic_istream :: tellg( );
pos_type  basic_ostream :: tellp( );
```

- **Przykład:**

```
#include <iostream>
#include <fstream>

int main ( )
{
  using namespace std;
  ifstream file;
  char c, c1;

  file.open( "basic_istream_seekg.txt" );
  file.seekg(2);   // chars to skip
  file >> c;
  cout << c << endl;

  file.seekg( 0, ios_base::beg );
  file >> c;
  cout << c << endl;

  file.seekg( -1, ios_base::end );
  file >> c1;
  cout << c1 << endl;
}
```

Plik:
basic_istream_seekg.txt:

```
0123456789
```

Output:

```
2
0
9
```

- **Sprawdzenie stanu wejścia \ wyjścia. Bieżący stan systemu We \ Wy jest przechowywany w obiekcie typu** ios_base::iostate :

typedef T2 iostate;

static const iostate badbit, eofbit, failbit, goodbit;

ios_base::goodbit	**Żaden bit błędu nie był wykryty**
ios_base::eofbit	**1 – EOF, 0 – nie pozostał osiągnięty**
ios_base::failbit	**1 – wystąpił niekrytyczny błąd We \ Wy, 0 – nie wystąpił**
ios_base::badbit	**1 – wystąpił krytyczny błąd We \ Wy, 0 – nie wystąpił**

- **Funkcje** ios_base::iostate basic_ios:: rdstate( ) const; **zwraca obiekt typu iostate, zawierający te flagi bitowe.**

- **failbit = 1 – na przykaład, przy wprowadzeniu int okazał się symbol, który nie może być konwertowany poprawnie na typ int (Przykład IO_1)**

Przykład:

```
template <typename T>
void checkfilestate(T &stream)
/*=====================================================
T == ifstream || T == ofstream || T == fstream
=====================================================*/
{
        ios_base::iostate st = stream.rdstate();

        if(st & ios_base::eofbit)
                cout << "EOF achieved\n";
        else if(st & ios_base::failbit)
                cout << "nie fatalny blad I/O\n";
        else if(st & ios_base::badbit)
                cout << "fatalny blad I/O\n";
}
```

- **Inny sposób sprawdzenia:**

```
bool  basic_ios::bad( ) const;      //true, jeśli flaga badbit jest ustawiona
bool  basic_ios::eof( )  const;     //true, jeśli flaga eofbit jest ustawiona
bool  basic_ios::fail ( ) const;    // true, jeśli flaga failbit jest ustawiona
bool  basic_ios::good ( ) const;  // true, jeśli flaga goodbit jest ustawiona
```

- **wyzerowanie flag błędów:**

  void basic_ios::clear(iostate _State **= goodbit);**

  Jeśli jako argument jest podana flaga goodbit, wszystkie flagi pozostaną wyzerowane. Jeśli jako argument podać inną flagę, tylko ta flaga będzie wyzerowana.

- **Przykład W16 – jeden plik – dwa strumieni**
- **Przykład L7 – jeden plik – jeden strumień (otwieranie plików, odczyt-zapis, testowanie odczytu-zapisu, wielokrotne powtarzanie, eksportowanie pliku do Excel i uruchomienie procesu Excel).**