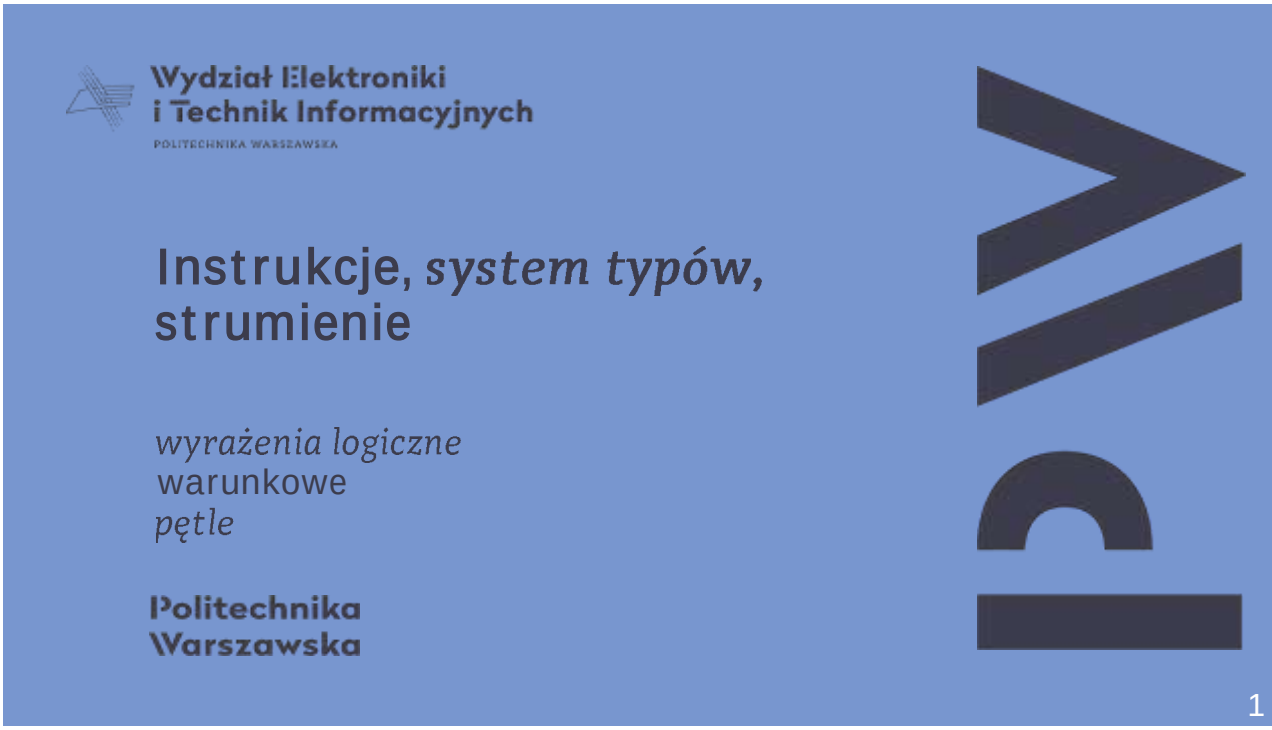


if, if else

```
int x = 6;
int y = 2;

 if (x > y)
    cout << "x is greater than y\n";
 else if (y > x)
    cout << "y is greater than x\n";
 else
    cout << "x and y are equal\n";
```

```
if(warunek)
  instrukcja_true
else
  instrukcja_false
```

```
if(warunek)
{
  instrukcja1
  instrukcja2
}
```

`warunek` - wyrażenie przyjmujące wartość logiczną `true` lub `false`

- w C++ typ `bool`
- W C wyrażenie !=0 traktowane jest jak `true`

operator **?:**

```
warunek ? wyrażenie1 : wyrażenie2
```

```
cout <<(x>y? "x is > y : "y <= x\n");
```

Politechnika Warszawska

switch - case


```
switch (x)
   {
      case 1: cout << "x is 1\n";
              break;
      case 2: int z = 0; // initialization
      case 3: cout << "x is 2 or 3";
              break;
    default:
      cout << "x is not 1, 2, or 3";
   }
```

typu całkowitego lub z konwersją do typu całkowitoliczbowego

wyrażenie stałe

w C++17 instrukcja switch może wprowadzać i inicjować zmienne

compilation error:
jump to default: would enter the scope of 'z' without initializing it

Pętle – while, do, for


```
int x = 0;
while (x < 10)
{
  cout << x << ' ';
  x = x + 1;
}
cout << endl;

x = 0;
do
{
  cout << x << ' ';
  x = x + 1;
} while (x < 10);
cout << endl;
```

```
x = 0;
do
{
   cout << x++ << ' ';
} while (x < 10);
cout << endl;

for (x = 0; x < 10; ++x)
  cout << x << ' ';
```

postinkrementacji

preinkrementacja

0 1 2 3 4 5 6 7 8 9

instrukcje break/continue w pętlach


```
x = 0;
do
{
   if (x == 3) continue;
   if (x == 6) break;
   if (x == 7) continue;
   cout << x << ' ';
} while (++x < 10);
```

0 1 2 4 5

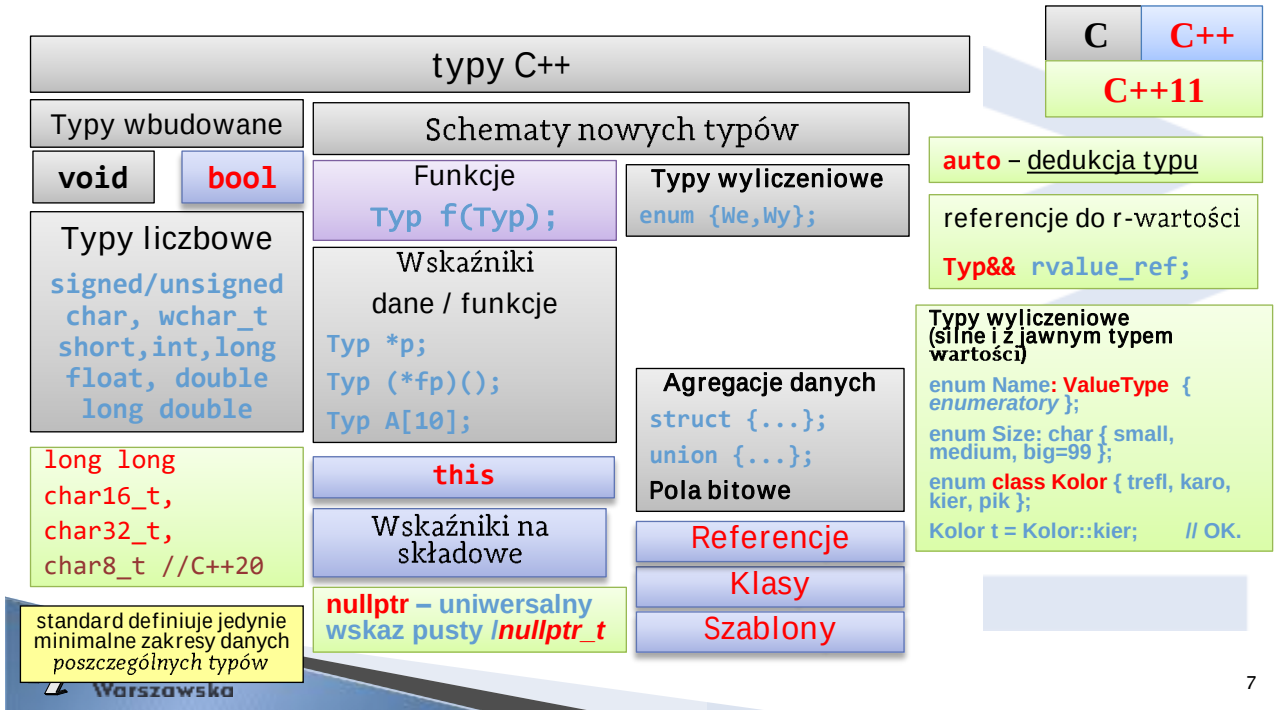
Przykład - quiz


```
#include <iostream>
using std::cout;

int main()
{
    int x = 0, y = 0;
    while ( x++ < 3 || y++ < 3 )
    {
        cout << x << ' ' << y << std::endl;
    }
    cout << "x="<< x << ", y=" << y << std::endl;
}
```

1 0
2 0
3 0
4 1
5 2
6 3
x=7, y=4

ile razy wykona się pętla?
x = ?, y = ?



Typ bool

Trzy słowa kluczowe: **bool**, **false**, **true**
Dla zachowania zgodności wstecz stosowane konwersje implikowane na **bool** w kontekstach warunkowania:

```
if(warunek) ...;// warunek sprowadzany do bool
while(warunek) ...;
do { .... } while(warunek);
for(...; warunek; ...) ...;
... (warunek)? exp1 : exp2 ...
```

Wszystkie operatory relacji i logiczne zwracają **bool**
Wartości **bool** w kontekstach numerycznych zamieniane na 0 (false) albo 1 (true)

Konwencja języka C (mało bezpieczna):
wartość numeryczna lub wskazanie != 0: TRUE
wartość numeryczna lub wskazanie == 0: FALSE

Politechnika Warszawska

```
int main()  // Będą ostrzeżenia
{ int i=5, j=3, k=1;
  bool b1=false, b2=true, b3;
  P(sizeof(bool));
  P(b1);
  P(b2);
  P(b2=5);
  P(b2+b2);          // kontekst numeryczny
  P(++b2);           // ++b2 = 1 !
  P(b1=b2+b2);
  P(b1 & !b2);
  b1++;
  //P(b1--);         //Błąd
  //P(--b1);         //Błąd
  P(i<j<k);          // coś nie tak
  P(i<(j<k));        // coś nie tak
  P(i<j && j<k);
  P((b2=cin));       // cin to obiekt!
  void*p;
  P((p=cin));        // (p=cin) = jakiś adres
}
```

```
#define P(x) cout << #x " = " << (x) << endl
// Tylko dla skrocenia zapisu !!!
```

```
sizeof(bool) = 1
b1 = 0
b2 = 1
b2=5 = 1
b2+b2 = 2
++b2 = 1
b1=b2+b2 = 1
b1 & !b2 = 0
i<j<k = 1
i<(j<k) = 0
i<j && j<k = 0
(b2=cin) = 1
(p=cin) = 0F41C288
```

Politechnika Warszawska

Typy wbudowane, liczbowe

char	Znaki kodu ASCII. char, signed char, unsigned char - różne typy; (minimum 8 bit) promocja do int zależy od kompilatora (ze znakiem lub bez)
short	signed short int lub short int lub short: zakres nie mniejszy od char, i nie większy od int
int	zakres nie mniejszy od short i nie większy od long
long **long long**	signed long int lub long int: zakres nie mniejszy od int signed long long int lub long long int: zakres nie mniejszy od long
float	najmniejszy zakres i precyzja zmiennopozycyjna
double	zakres i precyzja ≥ float i ≤ long double.
long **double**	zakres i precyzja nie mniejsza od double.
wchar_t	typ zdefiniowany przez typedef (unsigned short)

Politechnika Warszawska

char ≤ short ≤ int ≤ long ≤ long long

Typy całkowite C++11

int8_t, int16_t, uint32_t …	Typ o jawnych (dokładnych) rozmiarach
int_fast8_t int_fast16_t	Typy natywne o minimalnych rozmiarach
intmax_t	Maksymalny typ całkowity
uint_least32_t	O długości nie mniej niż
INT8_MIN INT16_MIN INT64_MAX	Makra dopuszczalnych zakresów danych
INT16_C UINT32_C	Makra konwersji do określonego typu

Dla C++11 <cstdint>
(http://en.cppreference.com/w/cpp/types/integer)

Zakresy typów wbudowanych <limits>


```
CHAR_BIT   8          // liczba bitów w char
SCHAR_MIN -128        // minimalna wartość  signed char
SCHAR_MAX  127        // maksymalna wartość signed char
UCHAR_MAX  0xff       // maksymalna wartość unsigned char
CHAR_MIN   SCHAR_MIN  // jeśli char signed
CHAR_MAX   SCHAR_MAX  // jeśli char signed
CHAR_MIN   0          // jeśli char unsigned
CHAR_MAX   UCHAR_MAX  // jeśli char unsigned
SHRT_MIN  -32768      // minimalna  wartość short
SHRT_MAX   32767      // maksymalna wartość short
USHRT_MAX  0xffff     // maksymalna wartość unsigned short
INT_MIN    -2147483648   // min. wartość int
INT_MAX     2147483647   // max. wartość int
UINT_MAX    0xffffffff   // max. wartość unsigned int
LONG_MIN   -2147483648L  // min. wartość long
LONG_MAX    2147483647L  // max. wartość long
ULONG_MAX   0xffffffffUL // max. wartość unsigned long
LLONG_MAX   9223372036854775807LL // max. long long
LLONG_MIN  -9223372036854775808LL // min. long long
ULLONG_MAX  0xffffffffffffffffULL //max. unsigned long long
```

`char` może być `signed` lub `unsigned`

`int` to zawsze `signed int`

Szablon std::numeric_limits<>


```
#include <iostream>
#include <limits>
int main()
{
  int mx = std::numeric_limits<int>::max();
  std::cout << "UCHAR_MAX :" << UCHAR_MAX <<"\n";
  std::cout << "max int   :" << mx << ": !\n";
  mx++;
  std::cout << "max int+1 :" << mx << ": !\n";
}
```

```
UCHAR_MAX :255
max int   :2147483647: !
max int+1 :-2147483648: !
```


Konwersje typów


```
#define P(x) cout<< #x << " (" << sizeof(x)<< ") : "<< x << endl;

int main()
{
  P(2 + 3);
  P(2. + 3);
  short si = 1, sj = 3;
  char c1 = '0';
  char c2 {58};  // C++11
  char c3 = 346; //256+90
  //char c4 { 346 };  // C++11 - błąd
  int i1 = c1, i3 = c3;
  int i2 = 2.7;
  P(c1); P(c2); P(c3); P(i1); P(i2); P(i3);
  P(c1 + c1);   P(si); P(si + sj);
  long double ld = 3.6;
  P(ld);  return(0);
}
```

operator sizeof(ident), sizeof(typ) podaje rozmiar obiektu lub typu

```
2 + 3 (4) : 5
2. + 3 (8) : 5
c1 (1) : 0
c2 (1) : :
c3 (1) : Z
```

```
i1 (4) : 48
i2 (4) : 2
i3 (4) : 90
c1 + c1 (4) : 96
si (2) : 1
si + sj (4) : 4
ld (8) : 3.6
```

- konwersje podczas przypisania
- konwersje w wyrażeniach
- konwersja wartości przekazywanych jako parametry funkcji

Tablice

liczba elementów

typ wartości elementów

nazwa

inicjalizacja

```
unsigned short tablica[4] = {1,2,3,4};

cout << sizeof(short) << " "
     << sizeof(tablica) << endl;
for (auto v : tablica)
   cout << v << " "; cout << endl;

cout << tablica[0]++  << " "
     << ++tablica[1] << endl;
for (auto v : tablica)
  cout << v << " "; cout << endl;

cout << tablica << " "
     << tablica+1 << endl;
```

```
2 8
1 2 3 4
1 3
2 3 3 4
0053F984 0053F986
```

	53F982	33	
	53F983	3	
tablica	53F984	0	[0]
	53F985	1	
	53F986	0	[1]
	53F987	2	
	53F988	0	[2]
	53F989	3	
	53F98A	0	[3]
	53F98B	4	
	53F98C	39	
	53F98D	27	
	53F98E	100	
	53F98F	65	
	53F990	46	
	...		

A co jeśli napiszemy?

```
cout << *tablica << " "
     << *(tablica+1) << endl;
```

Politechnika Warszawska

Łańcuchy znaków


```
#define P(x) cout<< #x << " (" << sizeof(x)<< ") : "<< x << endl;
int main() {  //deklaracja ciagów
  char proi[10] = { 'P', 'R', 'O', 'I', '\0' };
  char cpp[10] = { 'C', 'P', 'P'  };
  char zly[5] = { 'H', 'e', 'l', 'l', 'o' };
  char dobry[] = "Dobry tekst";
  std::string ss = "string";
  std::string sl = "**To jest troche dluzszy string**";
  P(proi);
  P(cpp);
  P(zly);
  P(dobry);
  P(ss);
  P(sl);
  P(sl.c_str());
```

Łańcuch musi być zakończony znakiem NULL (wartością 0)

pozostałe elementy inicjalizowane 0

klasa string

```
proi (10) : PROI
cpp (10) : CPP
zly (5) : Hello╠╠╠╠╠╠╠╠╠╠CPP
dobry (12) : Dobry tekst
ss (28) : string
sl (28) : **To jest troche dluzszy string**
sl.c_str() (4) : **To jest troche dluzszy string**
```

Politechnika Warszawska

Sekwencje specjalne

Sekwencja specjalna	Opis
\'	Apostrof
\"	Cudzysłów
\\	Backslash
\0	Znak zerowy
\a	Dzwonek
\b	Backspace
\f	Formfeed
\n	Nowa linia
\r	Powrót karetki
\t	Tabulacja pozioma
\v	Tabulacja pionowa
\nnn	Numer znaku (ósemkowo)
\xnn	Numer znaku (hex)

std::vector<>


```
vector<int> v1 = { 1, 2, 3, 4, 5 }; // initializer list
vector<int> v2{ 6, 7, 8, 9, 10 };   // uniform initialization
vector<int> v3(5, 10);
cout << "\nv1 = ";
for (const int& i : v1)
      cout << i << "  ";

cout << "\nv3 = ";
for (int i : v3)
    cout << i << "  ";

v1.push_back(6);
v1.push_back(7);
```

```
cout << "\nv1[3]=" << v1[3] ;
cout << "\nv1[9]=" << v1[9];  //!! wartośc przypadkowa
                               // błąd zakresu (Debug)
// cout << v1.at(8); //  exception

v1.at(4) = 14;
v1[3] = 13;

cout << "\nv1 = ";
for (const int& i : v1)
   cout << i << "  ";

}
```

```
v1 = 1  2  3  4  5
v3 = 10  10  10  10  10
v1[3]=4
v1[9]=0
v1 = 1  2  3  13  14  6  7
```

Literał	Podstawa	Typ
"hello"	ASCII	char *
L"hello"	ASCII	unsigned short * (wchar_t*)
'1'	ASCII	char
L'ab'	ASCII	unsigned short int
01, 1, 0x1, 'ABC'	octal, decimal, hex, ASCII	int
01U, 1U, 0x1U	octal, decimal, hex	unsigned int
01L, 1L, 0x1L	octal, decimal, hex	long int
01UL, 1UL, 0x1UL	octal, decimal, hex	unsigned long int
12.3F, 12E1F	decimal	float
12.3, 12e1	decimal	double
12.3L, 12E1L	decimal	long double

Niestety o **auto**

Lepiej unikać przy literałach

```
int main() {
  auto i = 1;    //int
  auto l = 1L;   //long
  auto u = 1u;   //unsigned int
  auto lu = 1LU; //unsigned long int
  auto f = 2.5f; //float
  auto d = 35.87;//double


  auto cc = "hello";    // const char*
  auto cwc = L"hello"; // const wchar_t*
  auto cc16 = u"hello"; // const char16_t*
  auto cc32 = U"hello"; // const char32_t*
  auto pl = "Dzięcioł"; // const char*
  auto utf8 = u8R"(Dzięcioł)";//const char* (C++20)

  auto c0 = 'A';   // char
  auto i0 = 'AB';   // int
  //auto b = 'CDEFG';  // error
  auto c1 = u8'A'; // char
  auto wc = L'A';  // wchar_t
  auto us = L'AB'; // wchar_t
  auto c16 = u'A'; // char16_t
  auto c32 = U'A'; // char32_t
}
```

… i wiele, wiele innych !

Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych
POLITECHNIKA WARSZAWSKA

Operacje We/Wy

Strumienie

Politechnika Warszawska

Typy wbudowane i strumienie

C: <stdio.h>

Strumienie standardowe (FILE*)

```
stdin  getchar(.)
       scanf(.)
stdout putchar(.)
       printf(.)
stderr  fprintf(.)
```

Inne usługi (~50 funkcji)

```
fopen(.),    fclose(.)
vfprintf(.), fflush(.)
sscanf(.) .........
```

Konwersje typów wbudowanych poprzez deskryptory w formatach:

```
printf("%d %s\n",v,"m/s");
```

C++: <iostream>

Strumienie standardowe // wide

```
istream std::cin; // wcin
ostream std::cout;// wcout
ostream std::cerr;// wcerr
ostream std::clog;// wclog
```

Usługi podstawowe

operator: << (wyjściowy)

operator: >> (wejściowy)

Inne usługi - kilkadziesiąt

Konwersje typów wbudowanych i nowych typów: przez przeciążenie operatorów

```
std::cout<<v<<" m/s"<< std::endl;
```

```
3.5 m/s
```

C++20: <format>

Usługi formatowania

```
std::format, std::format_to, ...

std::cout <<
    std::format("Hello {}!\n",
    "world");
constexpr double pi
    { std::numbers::pi };
std::cout <<
    std::format("Default: {},\n
    digits: {:.2}\n,
    fixed:  {:.2f}\n", pi, pi, pi);
```

C++ 23: <print>

```
std::print("{2} {1}{0}!\n", 23,
    "C++", "Hello");
```

```
Hello C++23!
```

Konwersje typów wbudowanych i nowych typów: definiowanie specjalizacji szablonow klas

```
Hello world!
Default: 3.141592653589793,
digits: 3.1,
fixed: 3.14
```

Politechnika Warszawska

Typy wbudowane i strumienie (cd1)

- strumienie standardowe są obiektami unikalnymi utworzonymi podczas preludium aktywacji programu i istnieją aż do zakończenia epilogu programu
- operatory << i >> można stosować do wszystkich typów wbudowanych oraz do manipulatorów (jak endl)
- operatory zwracają referencję na odpowiedni strumień – można zatem stosować je kaskadowo:

```
cout << "a+b = "; cout << a+b; cout << endl;
cout << "a+b = "<< a+b << endl;
cin >> x >> y;  cin >> x; cin >> y;
```

- strumienie (z wyjątkiem cerr) są buforowane; manipulator endl na wyjściu gwarantuje "wymiecenie" (flushing) bufora i przejście do nowego wiersza

Politechnika Warszawska

Typy wbudowane i strumienie (cd2)

- wywołanie operatora wejścia także powoduje wymiecenie bufora wyjściowego:

```
char nazwisko[44];
cout<<"Podaj nazwisko: "; cin>>nazwisko;
```

Uwaga: takie czytanie do tablicy znaków jest niebezpieczne

Procedura konwersji wyjściowej: `ost<<exp;// typ T`

- Wyprowadzaną wartość exp typu T poddaj konwersji do postaci znakowej wg aktualnych ustawień atrybutów strumienia; domyślne ustawienia: wyprowadzanie bez "dekoracji" od aktualnej pozycji strumienia.

Procedura konwersji wejściowej: `ist>>t; // typ T`

- Pomiń znaki "białe"; czytaj sekwencję znaków zgodną z reprezentacją znakową T do pierwszego znaku (wyłącznie) poza reprezentacją; ustaw stan poprawności operacji.

Typy wbudowane i strumienie (cd3)


```
int main() {
  char c;  int i, j, k, l;
  char s[20]; // std::string s;
  double d; // niebezpieczne czytanie do s
  cin>> i >> c >> d >> s >> j >> k >> l;
  cout<<i << c << d << s << j << k << l << endl;
return 0;
}
```

```
  11 22 33 44 55 66 77
  112233445566
--
  d = ?
```

```
123 X 3.14 metr 5 6 7 8 9
123X3.14metr567       ^ tu jesteśmy w cin
```

- Poprawność operacji wejścia można sprawdzić badając: if(ist>>t) ... // operacja udana
- Stan strumienia określają bity: badbit, failbit, eofbit, goodbit dostępne przy pomocy funkcji podglądu: `ist.bad(); ist.fail(); ist.eof(); ist.good();`
- Operacja `ist.clear();` przywraca współpracę po fail.

Typy wbudowane i strumienie (cd3)


```
int main() {
  char c;  int i, j, k, l;
  char s[20]; // std::string s;
  double d; // niebezpieczne czytanie do s
  cin>> i >> c >> d >> s >> j >> k >> l;
  cout<<i << c << d << s << j << k << l <<endl;
  if(!cin) cin.clear();
  cin >> s; cout <<  s;
return 0;
}
```

```
cal 2.54 1 2 3 4
-8593460A-9.256e+061-8589460-8583460-85460
cal
```

- Poprawność operacji wejścia można sprawdzić badając: if(ist>>t) ... // operacja udana
- Stan strumienia określają bity: badbit, failbit, eofbit, goodbit dostępne przy pomocy funkcji podglądu: `ist.bad(); ist.fail(); ist.eof(); ist.good();`
- Operacja `ist.clear();` przywraca współpracę po fail.

Typy wbudowane i strumienie (cd4)

Atrybuty / manipulatory formatowania

`setw(n)`	określa szerokość pola dla następnej konwersji
`boolalpha`	bool znakowo: false, true zamiast liczbowo 0, 1
`dec`	wartości całkowite dziesiętnie
`hex`	wartości całkowite heksadecymalnie
`oct`	wartości całkowite oktalnie
`showbase`	pokazuj prefiks określający podstawę zapisu wartości całkowitej (np. `0xff, 0377`)
`fixed`	wartości zmiennopozycyjne bez wykładnika
`scientific`	wartości zmiennopozycyjne z wykładnikiem
`showpoint`	pokazuj kropkę dziesiętną zawsze
`left`	wyrównanie lewostronne (dopełnienie z prawej)
`right`	wyrównanie prawostronne (dopełnienie z lewej)
`showpos`	pokazuj znak, także liczby nieujemnej
`skipws`	pomijaj białe znaki na wejściu
`unitbuf`	wymiataj bufor wyjściowy po każdym wstawieniu
`uppercase`	stosuj duże litery (np. `0XFFFF, 0.1E-5`)

Typy wbudowane i strumienie (cd5)


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

int main() {
  cout << 1.23456 << endl;
  cout << scientific << 1.23456 << endl;
  cout << setw(15) << right << 1.23456 <<' ' << 67 << 'T' << endl;
  return 0;
}
```

```
 1.23456
1.234560e+000
  1.234560e+000 67T
```

Podobnie jak ze strumieniami standardowymi współpracuje się ze strumieniami znakowymi:

- w plikach (plik nagłówkowy <fstream>)
- w pamięci (plik nagłówkowy <sstream>)

Pełny zakres usług także do współpracy z plikami binarnymi i wsparcie dla serializacji obiektów

Politechnika Warszawska

Pliki i strumienie w pamięci - przykład


```
using namespace std;
int main()
{ stringstream sst;      // Strumień w pamięci
  sst << 123 << "XYZ";  // zapisz coś
  string s;
  sst >> s;             // s=="123XYZ"
  cout << s << endl;
  cout << "Nazwa pliku: "; cin >> s;
  ifstream ifs(s.c_str()); // Otwórz plik
  if(!ifs) // Nieudana próba otwarcia
  { cerr << "Nie ma pliku " << s << endl;
    return 1;
  }
  while(ifs>>s) cout<<s<<'\n'; // Kopiuj do cout
  return 0;
}
```

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
```

Politechnika Warszawska

Podsumowanie

System współpracy ze strumieniami w konwencjach C++ jest bezpieczniejsza niż w C (pełna kontrola typów)

- w C - niezgodności typów pomiędzy deskryptorami konwersji (np. %d) a argumentami w rodzinie funkcji `printf(.)` przechodzą do czasu wykonania – mogą nawet zostać niezauważone mimo błędnego działania
- w C++ kontrola jest przeprowadzana statycznie podczas kompilacji - mniej błędów w programach

Konwencje operatorowe łatwo przenosi się na nowe typy danych i uwzględnia w dziedziczeniu:

```
Punkt p(1, 2); /*...*/ cout<<p; // ...
```

Trzeba pamiętać o niebezpieczeństwach:

```
while(true) // Możliwe zapętlenie
{   cin>>n; if(n<=0) break;
    /* Przetwarzaj n ... */ }
```

Politechnika Warszawska

Przykład (1)

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

int main()
{ int suma = 0;
  int liczba;
  while(!cin.eof())
  {
    cin >> liczba;
    suma += liczba;
    cout << "Suma aktualna: " << suma << endl;
  }
  cout << "Suma koncowa: " << suma << endl;
  system("pause"); return 0;
}
```

```
1 x 2 y 3
Suma aktualna: 1
Suma aktualna: 2
Suma aktualna: 3
...
Suma aktualna: 9999
...
```

```
1 2 3
Suma ak
Suma ak
Suma ak
^Z
Suma ak
Suma koncowa: 9
```

```
1
Suma a
2
Suma a
3
Suma a
^Z
Suma aktualna: 9
Suma koncowa: 9
```

Politechnika Warszawska

Przykład (2)


```
// Program sumujący liczby na wejściu – wersja 2
#include <iostream>
using namespace std;
int main()
{ int suma = 0;
  int liczba;
  // ios_base::sync_with_stdio(0);
  while(true)
  {
    cin>>liczba;
    if(!cin) break; // Niepowodzenie odczytu
    suma += liczba;
    cout<<"Suma aktualna: " << suma << endl;
  }
  cout<<"Suma koncowa: " << suma << endl;
  return 0;
}
```

```
1
Suma aktualna: 1
2
Suma aktualna: 3
3
Suma aktualna: 6
^Z
Suma koncowa: 6
```

```
1 x 2 y 3
Suma aktualna: 1
Suma koncowa: 1
```


Przykład (3)


```
// Program sumujący liczby na wejściu, inne znaki pomija
#include <cctype>
#include <iostream>
using namespace std;
int main() {
  int suma = 0;
  int liczba;
  char c;
  while(true)
  { cin>>c;          // Nastepny znak widoczny
    if(!cin) break;  // Niepowodzenie odczytu
    if(!isdigit(c)) continue;
    // . . . Rozpoznana cyfra
    cin.unget();     // Wycofanie ostatniego
    cin >> liczba;
    suma += liczba;
    cout<<"Suma aktualna: " << suma << endl;
  }
  cout<<"Suma koncowa: " << suma << endl;
  return 0;
}
```

```
1 x 2 y 3
Suma aktualna: 1
Suma aktualna: 3
Suma aktualna: 6
^Z
Suma koncowa: 6
```