Zaawansowane Techniki Programowania

1

Referencje

Informacje organizacyjne:

Wykład:
środy, 15:00 – 16:30

Strona główna www z komunikatami dla ZPO:
https://troja.uksw.edu.pl/category/ztp2020/

E-mail:
k.trojanowski@uksw.edu.pl
maile zbiorowe i anonimowe są ignorowane (konieczne jest imię i nazwisko)

2

Zasady zaliczeń

Zasady zaliczenia ZPO

1. Trzeba zaliczyć **zajęcia laboratoryjne (ćwiczenia) ZPO**.
2. Potem można próbować zaliczyć **wykład ZPO** (podejść do egzaminu).

Nigdy w odwrotnej kolejności.

3

Zasady zaliczeń

Zasady zaliczenia **zajęć laboratoryjnych**

- Jest 7 zajęć w semestrze.
- Na każdych zajęciach można uzyskać 0, 1, 2, .., 9 lub 10 pkt.
- Na zajęciach zawsze ustalony jest minimalny zestaw zadań do realizacji. Wykonanie tego zestawu w całości daje 6 pkt.

4

Zasady zaliczeń

Zasady zaliczenia **zajęć laboratoryjnych**

- Oprócz zadań zestawu minimalnego podawane są na zajęciach jeszcze inne zadania, za które można zdobyć pozostałe cztery punkty.
- Można mieć tylko 2 razy 0 pkt. za udział w zajęciach. Jeżeli przydarzy się to 3 razy, jedne zajęcia można odrobić na koniec semestru. Odrobienie polega na samodzielnym wykonaniu na ostatnich zajęciach dodatkowego zadania, którego zakres dotyczy całego materiału, niezależnie od tego, które zajęcia zostały zaliczone na zero i są odrabiane.

5

Zasady zaliczeń

Zasady zaliczenia **zajęć laboratoryjnych**

Warunkiem koniecznym zaliczenia całego semestru zajęć laboratoryjnych jest spełnienie łącznie następujących dwóch wymagań:

(1) zebranie co najmniej połowy ze wszystkich możliwych do uzyskania punktów,

(2) ewentualnie niezaliczenie nie więcej niż dwóch zajęć w semestrze.

6

Zasady zaliczeń

Zasady zaliczenia **wykładów**

- Wykład kończy się egzaminem pisemnym.
- Do egzaminu dopuszczone zostaną tylko te osoby, które wcześniej otrzymają zaliczenie z zajęć lab. Należy je uzyskać przed rozpoczęciem sesji egzaminacyjnej.
- Przewidywane są dwa terminy egzaminu.

Zasady zaliczeń

Zasady zaliczenia **wykładów**

- Zestaw egzaminacyjny będzie zawierał pytania z teorii programowania obiektowego oraz dobrych praktyk programistycznych.
- Na egzaminie nie będzie natomiast wymagane napisanie kodu programu.

Strumienie

Manipulatory bezargumentowe

C++ - strumienie

Manipulatory

Są to ***funkcje*** zdefiniowane w klasie `ios` i wywoływane poprzez podanie ich nazw jako elementów wstawianych do lub wyjmowanych ze strumienia. Ich działanie może, ale nie musi, polegać na modyfikacji flagi stanu formatowania strumienia

W zasadzie nie muszą to być wyłącznie funkcje. Mogą to też być obiekty funkcyjne.

Obiekty funkcyjne?

C++ - strumienie

Dygresja – obiekty funkcyjne

Obiekty funkcyjne to „coś co da się wywołać”. ☺

- Istnieje specjalny operator wywołania ‘()’, który można dodać do klasy i odpowiednio go przeciążyć. Obiekt takiego typu staje się obiektem *wywoływalnym* (*callable object, function object, functor*).
- Operator wywołania jest jedynym, który może mieć dowolną liczbę argumentów (np. więcej niż dwa)
- Instrukcja zawierająca nazwę obiektu tej klasy z podanymi argumentami w nawiasach powoduje wywołanie metody przeciążającej operator ‘()’.
- Definicja dla operatora wywołania ‘()’ :

```
typ klasa::operator() (argumenty) {
    // ciało funkcji
}
```


C++ - strumienie

- Wywołania obiektów z przeciążonym operatorem wywołania można wstawić wszędzie tam, gdzie może występować wywołanie funkcji.
- Mają tę przewagę nad funkcjami, że tworząc je można, na przykład poprzez konstruktor, przekazać do nich dodatkową informację prócz samych tylko argumentów wywołania.
- Obiekty funkcyjne zapewniają większą elastyczność algorytmów wyrażonych poprzez szablony.

koniec dygresji o obiektach funkcyjnych

C++ - strumienie

Manipulatory bezargumentowe

hex, oct, dec – podstawa dla wczytywanych/pisanych liczb całkowitych

C++ - strumienie

Wykorzystanie metod:

```
int i = 10;
cout << i << endl;
cout.setf( ios_base::oct );
cout << i << endl;
cout.setf( ios_base::hex,
   ios_base::oct );
cout << i << endl;
cout.unsetf( ios_base::hex );
cout.setf( ios_base::dec );
cout << i << endl;
```

Wykorzystanie manipulatorów:

```
int i = 10;
cout << i << endl;
cout << oct;
cout << i << endl;
cout << hex;

cout << i << endl;
cout.unsetf( ios_base::hex );
cout << dec;
cout << i << endl;
```


C++ - strumienie

Manipulatory bezargumentowe

left, right, internal – wyrównanie przy przypisaniu do lewej, prawej albo obustronnie
fixed, scientific – format dla wypisywanych liczb zmiennoprzecinkowych
showbase, noshowbase – przy wypisywaniu zawsze pokaż podstawę (wiodące zero lub 0x)
showpoint, noshowpoint - zawsze wypisz kropkę dziesiętną i końcowe zera w części ułamkowej

C++ - strumienie

Manipulatory bezargumentowe

Dedykowane dla **ostream:**

flush – powoduje opróżnienie strumienia wyjściowego
endl – powoduje wysłanie od strumienia wyjściowego znaku końca linii i opróżnienie bufora związanego z tym strumieniem, czyli natychmiastowe wyprowadzenie znaków z bufora do miejsca przeznaczenia
ends – powoduje wysłanie do strumienia wyjściowego znaku końca napisu, czyli znaku '\0'

Dedykowane dla **istream:**

ws – powoduje wymuszenie zignorowania wiodących białych znaków przy wczytywaniu

C++ - strumienie

Własne manipulatory bezargumentowe

Tworzymy je jako **funkcje globalne**

1. z dokładnie jednym parametrem, który musi być typu „referencja do strumienia";
2. wartością zwracaną musi być ten sam odnośnik.

```
ostream& mój_manip(ostream& strum)
{
   // … tutaj nasz kod działający na strumieniu
   return strum;
}
```

Taki manipulator zwraca referencję do strumienia, na rzecz którego został wywołany, dlatego można wstawiać do strumienia kaskadowo wraz z innymi argumentami czy manipulatorami.

C++ - strumienie

```
ios_base::fmtflags flagi_domyslne;
ostream& naukowy(ostream& str) {
  flagi_domyslne = str.flags();
  str.setf(ios::showpos |
         ios::showpoint);
  str.setf(ios::scientific,
         ios::floatfield);
  str.precision(12);
  return str;
}
ostream& normalny(ostream& str) {
  str.flags(flagi_domyslne);
  return str;
}
ostream& przec(ostream& str) {
  return str << ", ";
}
```

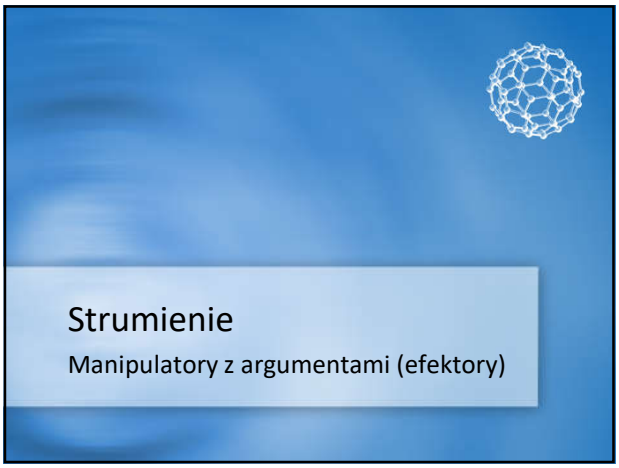
```
int main(int argc, char *argv[])
{
  double x = 123.456;
  cout << naukowy  << x << przec
       << normalny << x << endl;
```

C++ - strumienie

Własne manipulatory bezargumentowe

•We wszystkich przypadkach w poprzednim przykładzie parametr wywołania funkcji definiującej manipulator ma typ **`ostream&`**

•To daje elastyczność:
- nigdzie nie jest powiedziane, że to musi być **`cout`**
- może to być dowolny strumień reprezentowany obiektem **`ostream`** lub obiektem dowolnej klasy dziedziczącej po **`ostream`** (np. **`fstream`**, etc.)



Strumienie

Manipulatory z argumentami (efektory)

C++ - strumienie

Manipulatory z argumentami

Aby używać predefiniowanych manipulatorów argumentowych:

```
#include <iomanip>
```

- **setw(int szer)** – (*set width*) ustawia szerokość pola wydruku dla najbliższej operacji na strumieniu.
  Działa jak opisana wcześniej metoda `width(int)`, ale nie zwraca liczby, tylko odniesienie do strumienia. W ten sposób określana jest minimalna szerokość pola: jeśli wyprowadzana liczba zajmuje więcej znaków, to odpowiednie pole wydruku zostanie zwiększone
- **setfill(int znak)** – ustawia znak, którym będą wypełniane puste miejsca jeśli szerokość pola wydruku jest większa niż liczba wyprowadzanych znaków (domyślnie jest to znak odstępu)

C++ - strumienie

Manipulatory z argumentami

Aby używać predefiniowanych manipulatorów argumentowych:

```
#include <iomanip>
```

- **setprecision(int prec)** - ustawia precyzję jak metoda `precision`, ale nie jest zwracana poprzednia wartość, tylko odniesienie do strumienia
- **setbase(int baza)** - wypisuje liczby całkowite używając podstawy 'baza' (8, 10 lub 16)

C++ - strumienie

Manipulatory z argumentami

Aby używać predefiniowanych manipulatorów argumentowych:

```
#include <iomanip>
```

Uniwersalny manipulator do działania na poszczególnych flagach

setiosflags(long flag)
– ustawia flagę formatowania tak jak jednoargumentowa metoda `setf`, czyli „OR'uje" argument `flag` z flagą stanu formatowania;

resetiosflags(long flag)
– odpowiednik metody `unsetf` – gasi flagę wskazaną w argumencie.

C++ - strumienie

Manipulatory z argumentami

Przykład:

```
int j = 10;
cout << j << endl;
cout << hex;
cout << j << endl;
cout << resetiosflags(ios_base::hex);      // cout.unsetf( ios_base::hex );
cout << setiosflags(ios_base::oct);        // cout.setf( ios_base::oct );
cout << j << endl;
```

```
Wyjście:
10
a
12
```

C++ - strumienie

Własne manipulatory z argumentami

Aby utworzyć własne manipulatory z argumentami używamy techniki tworzenia *efektorów*.

Efektor to para:

1. Prosta *klasa* posiadająca pole oraz konstruktor, który pobiera parametr będący argumentem wywołania manipulatora, oraz
2. Przeciążona *funkcja* `operator<<` do wstawienia odpowiednich danych do strumienia, która jako jeden z argumentów przyjmuje obiekt danej klasy.

C++ - strumienie

Efektor – przykład:

1. Prosta klasa posiadająca pole oraz konstruktor:

```
class Fixw {
  string str;    // pole do przechowania podanej wartości
public:
  Fixw(const string& s, int width) : str(s, 0, width) {};
  friend ostream& operator<<(ostream& os, const Fixw& fw);
};
```

2. Przeciążona funkcja, korzystająca ze zdefiniowanej klasy:

```
ostream& operator<<(ostream& os, const Fixw& fw) {
    return os << fw.str;
}
```


C++ - strumienie

Efektor – przykład:

```
class Fixw {
  string str;
public:
  Fixw(const string& s, int width) : str(s, 0, width) {}
  friend ostream& operator<<(ostream& os, const Fixw& fw);
};
ostream& operator<<(ostream& os, const Fixw& fw) {
    return os << fw.str;
}
```

Wykorzystujemy tu konstruktor klasy 'string':

```
string ( const string& str, size_t pos, size_t n = npos );
```

zawartość obiektu inicjalizujemy łańcuchem tekstowym z którego bierzemy fragment zaczynający się od pozycji 'pos' i o długości 'n'

C++ - strumienie

Wykorzystanie przykładowego efektora:

```
cout << Fixw(str1, i) << endl;
```

za pomocą wywołania konstruktora **`Fixw`** tworzony jest obiekt tymczasowy, który jest przekazywany do globalnej funkcji **`operator<<`**

Obiekt tymczasowy typu **`Fixw`** pozostaje w pamięci aż do zakończenia instrukcji.

C++ - strumienie

Efektor – inny przykład:

```
typedef unsigned long ulong;
class Bin {
  ulong n;
public:
  Bin(ulong nn) { n = nn; }
  friend ostream& operator<<(ostream& os, const Bin& b);
};
friend ostream& operator<<(ostream& os, const Bin& b) {
    const ulong ULMAX = numeric_limits<ulong>::max();
    ulong bit = ~(ULMAX >> 1); // ustawiamy najwyższy bit
    while(bit) {
      os << (b.n & bit ? '1' : '0');
      bit >>= 1;
    }
    return os;
};
```


C++ - strumienie

Wykorzystanie przykładowego efektora:

```
ostringstream s;
s << Bin(0x76543210UL);
```

`~(ULMAX >> 1)` daje wartość `unsigned long` z ustawionym najstarszym bitem. Wartość ta jest przesuwana w prawo w kolejnych iteracjach pętli while.

Efektor Bin wykorzystuje fakt, że przesunięcie binarnej reprezentacji liczby bez znaku w prawo wstawia zera na starsze bity.

UWAGA!

Tyle informacji wystarczy, żeby wykonać zadania dotyczące dostępu do plików za pomocą strumieni, jakie będą realizowane na ćwiczeniach.

Strumienie

Buforowanie danych

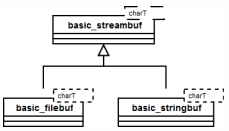
C++ - strumienie

Buforowanie strumieni

Dane znajdujące się w strumieniu należy interpretować zgodnie z ich znaczeniem, ale zawsze należy pamiętać, że są to po prostu sekwencje bajtów.

Za obsługę bajtową danych strumieniowych odpowiedzialny jest obiekt typu **`streambuf`** (dokładny typ zależy od tego, czy jest to strumień reprezentujący standardowe we/wy, plik, itd.)

Do tego obiektu można sięgać bezpośrednio: można np. przesunąć się o określoną liczbę bajtów bez ich formatowania przez strumień



C++ - strumienie

Buforowanie strumieni

Aby umożliwić sięgnięcie do obiektu **`streambuf`**, każdy obiekt **`iostream`** zawiera metodę **`rdbuf()`**:

```
basic_streambuf<Elem, Traits> *rdbuf( ) const;
```

zwraca wskaźnik do bufora strumieniowego.

```
basic_streambuf<Elem, Traits> *rdbuf( basic_streambuf<E, T> *_Sb);
```

umieszcza bufor strumieniowy podany w argumencie w miejsce poprzedniego, a wskaźnik do poprzedniego zwraca jako rezultat.

C++ - strumienie

Buforowanie strumieni – przykład:

```
int main( )
{
    ofstream file( "Test3.txt" );
    streambuf *x = cout.rdbuf( file.rdbuf( ) ); // podmiana..
    cout << "test1" << endl;   // dane są zapisywane do pliku
    cout.rdbuf(x);
    cout << "test2" << endl;
    ifstream in(  "Test3.txt" );
    cout << in.rdbuf();          // jednym ruchem cały plik!
}
```

Okno terminala:

test2

test1

C++ - strumienie

Buforowanie strumieni – przykład:

Aby przekazać wszystkie znaki z jednego strumienia do drugiego napisaliśmy:

```
cout << in.rdbuf();
```

Zastępuje to żmudne (i podatne na błędy) odczytywanie kolejnych znaków lub wierszy, jest też szybsze.

C++ - strumienie

Przypomnienie – metoda **get**:

int_type get( );
basic_istream<Elem, Tr>& get(Elem& _Ch**);**
basic_istream<Elem, Tr>& get(Elem* str**, streamsize** count**);**
basic_istream<Elem, Tr>& get(Elem* str**, streamsize** count**, Elem** _Delim**);**

Przerywa swoje działanie zaraz po znalezieniu ogranicznika, ale tego ogranicznika już nie pobiera ze strumienia.

```
char c[10];
c[0] = cin.get( ); // pobiera jeden element
cin.get( c[1] );   // pobiera jeden element
cin.get( &c[2],3 );// pobiera do napotkania znaku '\n', ale max. 3
cin.get( &c[4], 4, '7' );// pobiera do napotkania znaku '7', ale max. 4
cout << c << endl;
```


C++ - strumienie

Buforowanie strumieni

W klasie **istream** istnieje również wersja metody **get**, która potrafi pisać bezpośrednio do obiektu **streambuf** (a nie tylko do C-napisu):

basic_istream<Elem, Tr>& get(
basic_streambuf<Elem, Tr>& strbuf**);**

basic_istream<Elem, Tr>& get(
basic_streambuf<Elem, Tr>& strbuf**, Elem** _Delim**);**

C++ - strumienie

Przykład:

```
ifstream in(  "Test.txt" );
streambuf &sb = *cout.rdbuf();
while(!in.get(sb).eof())
  cout << char(in.get());
```

in.get(sb) – pobiera wiersz znaków – do napotkania '\n', ale tego znaku już nie pobiera – i wysyła do **sb**

in.get() – pobiera jeden znak i go zwraca

Zawartość pliku Test.txt:

Kiedy Kara Mustafa, wielki mistrz Krzyżaków, szedł z licznymi zastępy przez Alpy na Kraków, do obrony swych posad zawsze będąc skory pobił go pod Grunwaldem król Stefan Batory.

Wada: Jeżeli w pliku trafi się pusta linia, **in.get(sb)** zwróci znak błędu -1 i nie przejdzie do odczytu następnej linii.

C++ - strumienie

Przykład:

```
ifstream in(  "Test3.txt" );
streambuf &sb = *cout.rdbuf();
while(!in.get(sb).eof()) {
  if (in.fail()) // wystąpił błąd (znaleziono pusty wiersz)
    in.clear(); // czyścimy flagę błędu
  cout << char(in.get());
}
```

in.get(sb) – pobiera wiersz znaków – do napotkania '\n', ale tego znaku już nie pobiera – i wysyła do **sb**

in.get() – pobiera jeden znak i go zwraca

C++ - strumienie

Przeszukiwanie strumieni we/wy

W każdym typie strumienia **iostream** istnieje pojęcie „następnego" znaku do odczytania lub zapisania.

Możemy przesuwać się po znakach w strumieniu w przód i w tył, wskazując określone bezwzględne położenie albo przesunięcie względem położenia bieżącego.

Określanie położenia:

pos_type tellp( ); (do użycia z **ostream**)
pos_type tellg( ); (do użycia z **istream**)

C++ - strumienie

Przykład:

```
ifstream file;
char c;
int i;

file.open( "basic_istream_tellg.txt" );
i = file.tellg( );
file >> c;  // odczytywany jest znak
cout << c << " " << i << endl;     // pojawią się: znak 0

i = file.tellg( );
file >> c;   // odczytywany jest znak
cout << c << " " << i << endl;     // pojawią się: znak 1
```

C++ - strumienie

Przesunięcie się w strumieniu:

```
basic_ostream& seekp( pos_type _Pos );   (do użycia z ostream)
basic_istream& seekg( pos_type _Pos );   (do użycia z istream)
```

Metody jednoargumentowe pozwalają przesunąć się w określone miejsce:

```
ofstream x( "iotest.txt" );
int i = x.tellp( );          // zwróci 0
cout << i << endl;
x << "1234567890";
i = x.tellp( );              // zwróci 10
cout << i << endl;
x.seekp( i-5 );
x << " ";            // gdzie zostanie umieszczony znak spacji?
```


43

C++ - strumienie

Przesunięcie się w strumieniu:

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

Metody dwuargumentowe pozwalają przesunąć się o określony offset względem początku, końca lub bieżącego położenia:

- `ios::beg` – od początku (w ANSI C stdio: SEEK_SET)
- `ios::cur` – od aktualnego położenia (w ANSI C stdio: SEEK_CUR)
- `ios::end` – od końca strumienia (w ANSI C stdio: SEEK_END)

Wartość przesunięcia jako liczba dodatnia przesuwa w prawo a ujemna w lewo.

44

C++ - strumienie

Przesunięcie się w strumieniu – przykład:

```
ofstream x( "iotest.txt" );
int i = x.tellp( ); // pobieramy aktualne położenie w pliku
cout << i << endl;
x << "1234567890";
i = x.tellp( );     // pobieramy aktualne położenie w pliku
cout << i << endl;

x.seekp( i - 5 );  // zmieniamy aktualne położenie w pliku
x << " ";

x.seekp(  -2, ios::end ); // jeszcze raz zmieniamy..
x << "#";
```

// Po wykonaniu kodu plik zawiera: 12345 78#0

45

C++ - strumienie

```
ifstream file;
char c, c1;

file.open( „test.txt" );
file.seekg(2);
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 wejściowy test.txt:
0123456789

W oknie konsoli:
2
0
9

46

C++ - strumienie

Tworzenie obiektu strumienia umożliwiającego pisanie i czytanie

```
ifstream in2("test.txt", ios::in | ios::out);
ostream out2(in2.rdbuf());
cout << in2.rdbuf() << endl;
out2 << "ile Roman ładny dyndał na moreli";
out2.seekp(0, ios::beg);
out2 << "jeż leje lwa paw leje lżej";
in2.seekg(0, ios::beg);
cout << in2.rdbuf() << endl;
```

Plik wejściowy test.txt: !!!!!@@@@@#####$$$$$%%%%%-----
W oknie konsoli:
!!!!!@@@@@#####$$$$$%%%%%-----
jeż leje lwa, paw leje lżej---ile Roman ładny dyndał na moreli?

Dlaczego tekst pierwszy dodawany jest na końcu pliku?

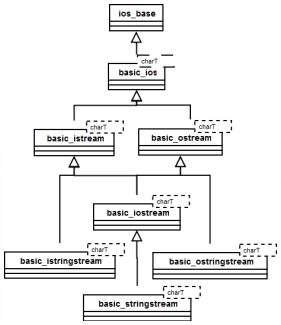
47

C++ - strumienie

Strumienie powiązane z łańcuchami

<sstream>

Bardzo przydatne np. do parsowania napisów (analizy składniowej napisów – list słów o ustalonej składni)



48

C++ - strumienie

Łańcuchowe strumienie wejściowe

```
#include <cassert>
#include <cmath>  // fabs()
#include <limits> // epsilon()
#include <sstream>
#include <string>
int main( ) {
  istringstream s("47 1.414 To tylko taki test");
  int i;
  double f;
  s >> i >> f;
  assert(i == 47);
  double relerr = (fabs(f) - 1.414) / 1.414;
  assert(relerr <= numeric_limits<double>::epsilon());
```


49

C++ - strumienie

Łańcuchowe strumienie wejściowe

```
int main( ) {
  istringstream s("47 1.414 To tylko taki test");
  int i;
  double f;
  s >> i >> f;
  assert(i == 47);
  double relerr = (fabs(f) - 1.414) / 1.414;
  assert(relerr <= numeric_limits<double>::epsilon());

  string buf2;
  s >> buf2;
  assert(buf2 == "To");
  cout << s.rdbuf() << endl; // " tylko taki test"
```


50

C++ - strumienie

Łańcuchowe strumienie wejściowe

Pobieranie wartości do zmiennych odbywa się zgodnie z ich typem:

```
istringstream s("47 1.414 To tylko taki test");
int i;
double f;
s >> i >> f;        //      i=47            f = 1.414
```

Gdyby zmienić kolejność liczb w ciągu:

```
istringstream s("1.414 47 To tylko taki test");
```

wtedy:

```
s >> i >> f;        //      i=1             f = 0.414
```


51

C++ - strumienie

Łańcuchowe strumienie wyjściowe

```
int main() {
  cout << "napisz wartości int, float i string: ";
  int i;
  float f;
  cin >> i >> f;
  cin >> ws; // odrzuć ewentualne białe znaki
  string stuff;
  getline(cin, stuff); // pobierz do końca wiersza
  ostringstream os;
  os << "integer = " << i << endl;
  os << "float = " << f << endl;
  os << "string = " << stuff << endl;
  string result = os.str();
  cout << result << endl;
```


52

C++ - strumienie

Użycie dwukierunkowego strumienia łańcuchowego

```
int main() {
  string text = "W Moskwie na
   Placu Czerwonym";
  stringstream ss(text);
  ss.seekp(0, ios::end);
  ss << " rozdają samochody.";
  assert(ss.str() ==
    "W Moskwie na Placu Czerwonym
   rozdają samochody.");
```

```
ss.seekg(18, ios::beg);
string word;
ss >> word;
assert(word == "Czerwonym");
ss.seekp(2, ios::beg);
ss << "Kijowie";
ss.seekg(29, ios::beg);
ss >> word;
assert(word == "rozdają");
ss.seekp(37, ios::beg);
ss << "rowery ..";
cout << ss.str() << endl;
```

Fałsz: w Kijowie nie ma placu Czerwonego

53

Strumienie

Binarny dostęp do danych

54

C++ - strumienie

Binarny dostęp do danych:

Do odczytu danych ze strumienia w taki sposób, aby można je było interpretować jako ciąg binarny[1] służy metoda klasy **istream**:

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

Metoda odczytuje określoną liczbę bajtów *_Count* i zapisuje ją pod wskazany adres – wskaźnik *_Str*. Jeżeli wcześniej napotka koniec pliku, przerywa odczyt i kończy działanie ustawiając odpowiednią flagę błędu.

Ciąg binarny – ciąg bajtów (*nie bitów!*).

Interpretacja danych dowolnych typów jako ciąg binarny polega na odczytaniu tych danych z pominięciem informacji o ich strukturze, tj. bajt po bajcie (*nie bit po bicie!*).

55

C++ - strumienie

Binarny dostęp do danych:

Do zapisu danych do strumienia służy metoda klasy **ostream**:

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

Aby zastosować tę metodę do korzystania z pliku w trybie binarnym, należy obszary pamięci, do który dane mają trafić (lub skąd mają być pobrane), zrzutować na wskaźniki typu `const char_type*` (zapis do strumienia) lub `char_type*` (odczyt ze strumienia).

56

C++ - strumienie

Binarny dostęp do danych:

```
struct Bloczek {
       int a;
       double b;
       char c;
};

int main( )
{
 ofstream ofx( "test.txt", ios::binary);  ifstream ifx( "test.txt",ios::binary);
 Bloczek s={3,3.14,'a'};                  Bloczek s2;
 const char *pom = (const char*)&s;       char *pom2 = (char*)&s2;
 ofx.write((char*)&s, sizeof(s));         ifx.read((char*)&s2, sizeof(s2));
 ofx.write(pom, sizeof(s));               ifx.read(pom2, sizeof(s2));
 ofx.close();                             cout << s2.a << s2.b << s2.c;
```


57