Język C++

część 1

pierwsze programy w C++
nieobiektowe rozszerzenia języka C++

Jarosław Gramacki
Instytut Informatyki i Elektroniki

- środowisko pracy
  - nasz ulubiony kompilator gcc w środowisku Linux
    version 3.3.5 (Debian 1:3.3.5-13)
  - serwer `mykonos.iie.uz.zgora.pl`
  - **An Introduction to GCC**
    for the GNU Compilers gcc and g++
    Revised and updated
    Brian Gough, Foreword by Richard M. Stallman

    `http://www.network-theory.co.uk/docs/gccintro/index.html`

    ```
    - ...
    - 7 Compiling a C++ program
    -     * 7.1 Compiling a simple C++ program
    -     * 7.2 C++ compilation options
    -     * 7.3 Using the C++ standard library
    -     * 7.4 Templates
    -           o 7.4.1 Using C++ standard library templates
    -           o 7.4.2 Providing your own templates
    -           o 7.4.3 Explicit template instantiation
    -           o 7.4.4 The export keyword
    - ...
    ```

  - Welcome to the GCC home page!
    `http://gcc.gnu.org/`
    `http://gcc.gnu.org/onlinedocs/`

- gdzie szukać informacji
  - –materiały referencyjne (C++)

    http://www.cplusplus.com/ref/

    http://www.cppreference.com/

    Newsgroups: comp.lang.c++

  - –materiały referencyjne

    (Standard C++ Library Reference / Standard Template Library STL)

    z reguły mowa jest tam głównie o STL

    http://www.unc.edu/depts/case/pgi/pgC++_lib/stdlib.htm

    http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcstdlib/html/vcoriStandardCLibraryReference.asp

    http://www.sgi.com/tech/stl/

    http://www.cppreference.com/

    http://www.msoe.edu/eecs/ce/courseinfo/stl/

    Newsgroups: comp.std.c++

  - –materiały referencyjne (gcc / g++)

    http://gcc.gnu.org/libstdc++/

    http://gcc.gnu.org/onlinedocs/libstdc++/documentation.html

    . . .

    ... no to start

Jeszcze nie znamy pojęć związanych z programowaniem obiektowym a już programujemy (obiektowo) w C++

biblioteka I/O

- operacje wejścia-wyjścia z użyciem `iostream` (nagłówek `<iostream>` dawniej `<iostream.h>`)
  - intensywne korzystanie z mechanizmów charakterystycznych dla języka obiektowego (klasy, obiekty, metody, referncje, przeciążanie funkcji i operatorów)
  - strumienie (obiekty) `cin cout cerr.` Inicjowane automatycznie
    obiekt `cin` **klasy** `istream; cin::istream`
    obiekt `cout` **klasy** `ostream; cout::ostream`
    operatory (przeciążone operatory bitowe) `<< >>`
  - korzystając z `iostream` można po utworzeniu własnego typy (klasy), przeciążać inne operatory tak, aby łatwo manipulować zmiennymi tego typu (obiektami)
  - standardowe funkcje wymagały sprecyzowania typu (kody formatujące np. %d). Operatory << >> po analizie w trakcie kompilacji same wykonują niezbędne konwersje typów

  Pierwszy przykład:

```
double d = 12.34;
cout << "Zmienna d rowna sie: " << d << endl;  //zm. double
```


- operacje wejścia-wyjścia z użyciem `iostream`
  - `iostream` zawiera BARDZO dużo innych BARDZO użytecznych funkcji (klas) np. **metoda** (funkcja, funkcja składowa) `get` **klasy** `istream` (w notacji C++ `istream::get()`)
  - `iostream` dokładnie – temat na oddzielny wykład (wykłady)
  - standardowe podejście (`<stdio.h>, printf(), scanf()`) jest oczywiście akceptowane, ale ... ???

- operacje wejścia-wyjścia z użyciem `iostream`

```
#include <iostream>        // NIE <iostream.h>

using namespace std;

// kilku linijkowy komentarz co robi funkcja
int main ()
{
  cout << "Hello, world!\n";
  std::cout << "Hello, world!\n";

  cout << "Hello";         // rozbicie na dwie linie
  cout << "world!\n";

 return 0;
}
```

bez tego trzeba by zawsze używać tzw. w pełni określoną nazwę obiektów.
Na foliach (dla oszczędności miejsca) będziemy to pomijali !!!

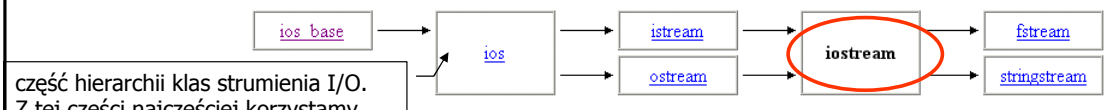
`<<` to **operator** wstawiania do strumienia

`cout` to **obiekt** strumienia standardowego wyjścia

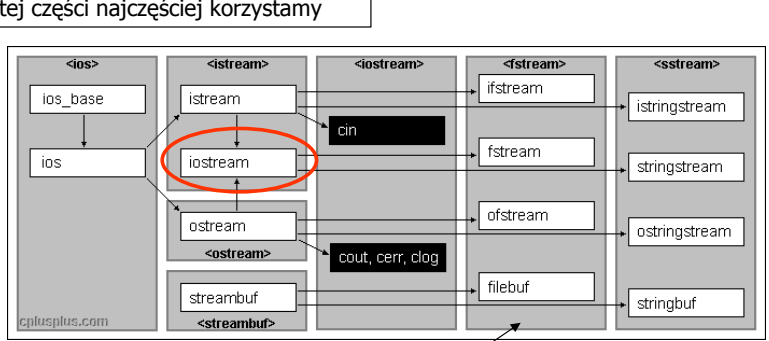
  - Uwaga: m.in. "dzięki" bibliotece iostream (ogromnej) programy w C++ kompilują się zdecydowanie dłużej niż programy w ANSI C

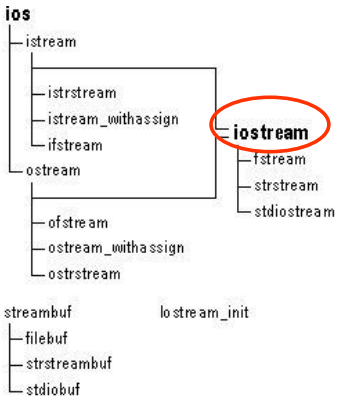
```
// biblioteka C
#include <stdio.h>
#include <stdlib.h>

// nowy styl, nagłówki pochodzą teraz z biblioteki standardowej C++
#include <cstdio>
#include <cstdlib>
```


- operacje wejścia-wyjścia z użyciem `iostream`
  - tylko dla ogólnej ilustracji: hierarchia klas biblioteki `iostream` w różnych układach i przykład jednej metody składowej klasy `istream` (wielokrotnie przeciążonej) `istream::get()`
  -
```
int get();
istream& get (char& c );
istream& get (char* s, streamsize n );
istream& get (char* s, streamsize n, char delim );
istream& get (streambuf& sb);
istream& get (streambuf& sb, char delim );
```



część hierarchii klas strumienia I/O.
Z tej części najczęściej korzystamy





część hierarchii klas do przetwarzania plików

- operacje wejścia-wyjścia z użyciem `iostream`

```
#include <iostream>

using namespace std;       // używany obszar nazw (namespace) std

void main()
{
    int a, b;
    char bufor[ 40 ];

    cout << "Napisz liczbe calkowita : ";
    std::cout << "Napisz liczbe calkowita : ";

    cin >> a;
    cout << "Napisales : " << a << endl;   //end of line, równoważne \n

    cout << "Napisz dwie liczby calkowite : ";
    cin >> a >> b;                         //odczyt dwóch liczb
    cout << "a = " << a << ", b = " << b << endl;

    cout << "Napisz slowo : ";
    cin >> bufor;
    cout << "Napisales : " << bufor << endl;
}
```

Umieszczaj spację **po każdym** przecinku

  - obszary nazw chronią przed konfliktami nazwa z istniejącym elementami oprogramowania
  - wprowadzono je aby bezkonfliktowo można było rozwijać (niezależne) biblioteki klas, w których używa się tych samych nazw klas i nazw funkcji
  - nazwa obszaru **`std`** jest zarezerwowana dla biblioteki standardowej

- operacje wejścia-wyjścia z użyciem `iostream`

write, read, gcount: niesformatowane I/O

```
// zastosowanie metod 1. get, put,  2. write, read, gcount
// zastosowanie manipulatorów strumieniowych hex, dec, oct, setbase

// zastosowano następującą wersję metody get:
// istream& get (char* s, streamsize n );

...
char bufor1[256];
char bufor2[] = "WAKACJE";
char bufor3[25];

char c;
int n;

while (c = cin.get(bufor1, sizeof(bufor1) != EOF)
 cout.put( c );

cout.put( 'A' );

cout.write( bufor2, 5);  // WAKAC pierwsze 5 bajtów z bufor2

cin.read( bufor3, 20);              // 20 znaków z wprowadzonych danych
cout << "Wprowadziles"\n";
cout.write( bufor3, cin.gcount() );// określa ilość wprowadzonych znaków
                                   // do strumienia cin

cout << "Podaj liczbe: ";
cin >> n;
cout << n << "to szesnastkowo: "
     << hex << n << "a ósemkowo: "         // "zmanipuluj" strumień
     << oct << n << "i znowu dzisietnie: "
     << setbase( 10 ) << n << endl;
```

- operacje wejścia-wyjścia z użyciem `iostream`

```
// zastosowanie metody eof

#include <iostream>

int main()
{
   int character; // use int, because char cannot represent EOF

   // prompt user to enter line of text
   cout << "Before input, cin.eof() is " << cin.eof() << endl
        << "Enter a sentence followed by end-of-file:" << endl;

   // use get to read each character; use put to display it
   while ( ( character = cin.get() ) != EOF )
      cout.put( character );

   // display end-of-file character
   cout << "\nEOF in this system is: " << character << endl;
   cout << "After input, cin.eof() is " << cin.eof() << endl;

   return 0;
}
```

wartość znacznika EOF danego strumienia

```
// Wynik działania

Before input, cin.eof() is 0      // bo nie wystąpił koniec pliku w cin
Enter a sentence followed by end-of-file:
 Wprowadzam jakiś wiersz          // wcisnąć Enter na końcu wiersza
 Wprowadzam jakiś wiersz          // wcisnąc Ctrl-Z jako koniec pliku
EOF in this system is: -1
After input, cin.eof() is: 1      // bo napotkano znak końca pliku
```


- operacje wejścia-wyjścia z użyciem `iostream`

```
// ustalanie precyzji liczb zmiennoprzcinkowych

#include <iostream>
#include <iomanip>        // dla setiosflags
#include <math>           // dla sqrt()

int main()
{
  double p = sqrt(2.0);

  cout << setiosflags( ios::fixed )
       << "pierwiastek z dokładnością do trzech miejsc: "
       << endl;

  cout.precision ( 3 );
  cout << p << endl;

cout << setprecision( 3 ) << p << endl;    // inny sposób

return 0;
}
```

Manipulator strumienia. Ustawianie flagi

Znacznik stanu formatowania. Wydruk wartości zmiennoprzecinkowej w notacji dziesiętnej z określoną ilością cyfr po prawej stronie kropki

```
// inne przykłady:

// ustalanie szerokości pola              (ios::width)
// kropka dziesiętna                      (ios::showpoint)
// wyrównanie do lewej, prawej, do środka (ios::left, ios:right,
                                           ios::internal)
// itd. jest tego bardzo dużo
```

- operacje wejścia-wyjścia z użyciem `iostream`

```
// ustalanie szerokość pola

Syntax:

#include <fstream>
int width();
int width( int w );

cout.width( 5 );
cout << "2";

Output:

____2            // 4 spacje przed '2'
```

```
Wprowadz zdanie:
To jest test metody width
  To
 jest
  test
   meto
      dy
     widt
         h
```

```
int main() {
   int w = 4;
   char zdanie[ 10 ];

   cout << "Wprowadz zdanie:" << endl;
   cin.width( 5 );         // pobierz tylko 5 znaków

   // ustaw szerokość pola
   while ( cin >> zdanie ) {
      cout.width( w++ );
      cout << zdanie << endl;
      cin.width( 5 );      // pobierz kolejne 5 znaków
   }

return 0;
}
```


- napisy z użyciem klasy `string`

```
// używamy klasy string

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

int main()
{
  string s1, s2;                    // puste napisy
  string s3 = "Hello, World.";      // napis zainicjalizowany
  string s4("I am");                // j.w.
  s2 = "Today";                     // przypisanie
  s1 = s3 + " " + s4;               // połączenie
  s1 += " 8 ";                      // j.w.
  cout << s1 + s2 + "!" << endl;
}
```

obiekty s1, s2 klasy **string**

klasa string i przetwarzanie strumieni napisów to temat na oddzielny wykład !

tu wszędzie "pracują" wyłacznie **obiekty** oraz (obiektowe) **operatory** !

- obsługa plików z użyciem `fstream`

```
// kopia pliku linia po linii

#include <string>         // getline
#include <fstream>
using namespace std;

int main()
{
  ifstream in ("File1.cpp");       // otwórz do czytania
  ofstream out("File2.cpp");       // otwórz do pisania

  string s;
  while( getline(in, s) )          // odczyt z pliku
  // while( getline(cin, s) )      // odczyt z klawiatury

    out << s << "\n";              // wypisz do pliku out
}
```

obiekty klasy ifstream, ofstream oraz właściwe dla nich **konstruktory**

- obsługa plików z użyciem `fstream`

```
// kopia pliku do pamięci / bufora (do obiektu klasy string)
// wada: uzyskujemy "big blob of text"

#include <string>
#include <iostream>
#include <fstream>
using namespace std;

int main() {
  ifstream in("File.cpp");
  string s, line;                  // nasz bufor
  while( getline(in, line) )
    s += line + "\n";              // brak "ręcznej" alokacji pamięci
  cout << s;                       // załatwia to za nas klasa string
}
```

Nieobiektowe zmiany / rozszerzenia języka C++

- deklaracje funkcji i prototypów
  - język C++ jest bardzie rygorystyczny niż język C jeśli chodzi o deklaracje liczby i typów argumentów funkcji oraz typów zwracanych przez funkcję
  - w języku C++ prototypy są bezwzględnie obowiązkowe (wczesna wersja języka C miała znana jest pod nazwą „C bez prototypów")
  - dzięki prototypom „działają" takie (nowe) mechanizmy jak m.in.:
    + wartości domyślne argumentów funkcji
    + przeciążanie funkcji
  - funkcje bezargumentowe np: `int fun()`
    + ANSI C przyjmie, że funkcja może otrzymać dowolna liczbę argumentów dowolnego typu czyli `int fun (int a, ...)`
    + C++ przyjmie, że funkcja nie pobiera żadnego argumentu czyli `int fun(void)`
    + z uwagi na powyższe, w C++ używa się powszechnie wersji `int fun()`

- deklaracje zmiennych wewnątrz kodu

  - w języku ANSI C deklaracja zmiennych tylko na początku bloku instrukcji (po {)
  - w C++ w dowolnym miejscu
  - uwaga jednak na zasięg i widoczność zmiennych – możliwe błędy !
    + zmienne `poczatek, koniec, suma, i` dostępne są aż do końca `main()`
    + zmienna `chwilowa` dostępna tylko wewnątrz bloku `for`
  - nowa możliwość deklaracji zmiennych ma głębszy sens w języku C++ niż tylko jedynie udogodnienie dla programisty. Umożliwia np. inicjalizację obiektu za pomocą **konstruktora**

ale o tym będzie jeszcze mowa

- deklaracje zmiennych wewnątrz kodu

```
void main()
{
    cout << "Mnozenie przez liczbe 2" << endl;

    cout << „Podaj wartosc poczatkowa : ";
    int poczatek;
    cin >> poczatek;

    cout << „Podaj wartosc koncowa : ";
    int koniec;
    cin >> koniec;

    int suma = 0;
    for( int i = poczatek; i <= koniec; i++ )
    {
      int chwilowa = i*2; // Deklaracja zmiennej "chwilowa".

      suma += chwilowa;
      cout << " i = " << i << " i*2 = " << chwilowa
           << " suma = " << suma << endl;
    }

    cout << "--- Koniec petli ---" << endl;

    cout << "poczatek = " << poczatek << endl;
    cout << "koniec = " << koniec << endl;
    cout << "i = " << i << endl;
    cout << "suma = " << suma << endl;

    // Ponizszy wiersz zostanie odrzucony:
    // odwolanie do zmiennej poza zasiegiem jej widocznosci.
    cout << "chwilowa = " << chwilowa << endl; // BLAD
}
```

(ANSI ?) C

```
#include <stdio.h>
#include <ctype.h>

long zamien( s )
char *s;
{
    int i, znak;
    long n;

    /* Pomijamy ewentualne spacje */
    for( i = 0; isspace( s[ i ] ); ++i )
    ;

    znak = 1;
    if( s[ i ] == '+' )
      ++i;
    else if( s[ i ] == '-' )
    {
      znak = -1;
      ++i;
    }


    for( n = 0; isdigit( s[ i ] ); ++i )
    n = 10 * n + s[ i ] - '0';

    return znak * n;
}
```

tak się kiedyś pisało. Wersja nie-ANSI C

```
main()
{
  long n;
  char lancuch[ 80 ];

  printf( "Napisz liczbe : " );
  gets( lancuch );
  n = zamien( lancuch );
  printf( "Wynik konwersji = %ld \n", n );
}
```


C++

```
#include <iostream>
#include <ctype.h>

long zamien( char *s )
{
  // Pomijamy ewentualne spacje
  for( int i = 0; isspace( s[ i ] ); ++i )
  ;

  int znak = 1;

  if( s[ i ] == '+' )
    ++i;
  else if( s[ i ] == '-' )
  {
    znak = -1;
    ++i;
  }


    for( long n = 0; isdigit( s[ i ] ); ++i )
    n = 10 * n + s[ i ] - '0';

    return znak * n;
}
```

```
main()
{
  char lancuch[ 80 ];

  cout << "Napisz liczbe : ";

  cin >> lancuch;
  //lub cin.get(bufor, sizeof(bufor))
  / uzyto funkcji istream::get()

  long n = zamien( lancuch );
  cout << "Wynik konwersji = " << n << endl;}
```

- definicja nowych typów danych z użyciem `struct i enum`

  - w języku ANSI C typy danych zdefiniowane za pomocą `struct i enum` nie zachowują się dokładnie tak, jak typy podstawowe (`int, char`, etc.). Konieczne jest poprzedzanie zdefiniowanego typu słowami `struct i enum`
  - nie jest to wygodne a wręcz wydaje się nadmiarowe
  - użycie `typedef` tylko częściowo rozwiązuje problem
  - w C++ typy zdefiniowane przez `struct i enum` zachowują się dokładnie tak, jak typy podstawowe
  - własność tę posiadają też **klasy**. Po jej zdefiniowaniu, nie powtarzamy wszędzie i pracowicie słowa kluczowego `class`

- definicja nowych typów danych z użyciem `struct i enum`

ANSI C

```
//Deklaracja nowych typów
enum dzien (poniedziałek, wtorek, środa, czwartek, piątek,
            sobota, niedziela);
typedef enum dzien t_dzien;

struct wezel {
   int wartosc;
   struct wezel* lewy;
   struct wezel* prawy;
};
typedef struct wezel t_wezel;

t_dzien d1, d2, d3;
t_wezel w1, w2, w3;

void pokaz_dzien (t_dzien d);
t_wezel* znajdz_wartosc (t_wezel* w);
```

- definicja nowych typów danych z użyciem `struct i enum`

C++

```
//Deklaracja nowych typów
enum dzien (poniedziałek, wtorek, środa, czwartek, piątek,
            sobota, niedziela);
typedef enum dzien t_dzien;

struct wezel {
   int wartosc;
   struct wezel* lewy;
   struct wezel* prawy;
};
typedef struct wezel t_wezel;

t_dzien dzien d1, d2, d3;
t_wezel wezel w1, w2, w3;

void pokaz_dzien (t_dzien dzien d);
t_wezel* wezel* znajdz_wartosc (t_wezel* wezel* w);
```


- konwersja typów (rzutowanie)
  - nowy styl rzutowania w C++. Nowe operatory zastępujące "stary" styl rzutowania
  - jednoargumentowy operator rzutowania `static_cast< typ >`

```
int main()
{
  double d = 7.34;
  int x = static_cast<int>( d );
  ...
}
```

szczegóły: samodzielnie

  - istnieją też operatory `const_cast, reinterpret_cast`

- konwersja typów (rzutowanie)

  - rzutowanie jako takie pozostaje bez zmian. Zmienia się jedynie **notacja** (tzw. **notacja funkcyjna**)
  - nowej notacji nie wolno jednak używać zawsze. Dotyczy to rzutowania na typ „nieprosty” np. dla typów będących wskaźnikami
  - w takich przypadkach trzeba zastosować `typedef`
  - zaletę nowej notacji widać przy bardziej złożonych wyrażeniach
  - w szczególności używana ona jest przy konwersji **obiektów** jednego typu na inny typ

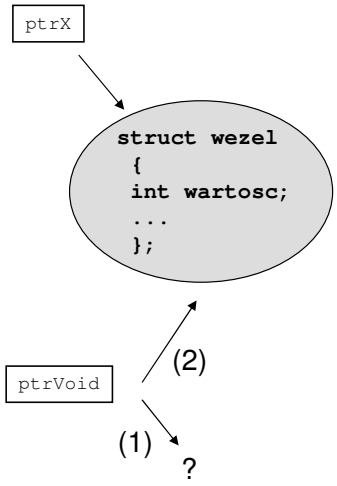
```
int main()
{
  float a = float(200);   // notacja funkcyjna

  float b = (float)200;  // odpowiednik:
}
```


- konwersja typów (rzutowanie)

```
double d = 2.3;
int i = 5, j;
char* str;
int* ptri = &i;

j = i * (int) d; //ANSI C
j = i * int (d); //C++

d = (double) j/5.0; //ANSI C
d = double (j)/5.0; //C++

str = (char*) ptri; // OK
str = char* (ptri); // BLAD

//ale gdy...
typedef char* pchar;
str = (char*) ptri; // OK
str = pchar (ptri); // teraz OK
```

- konwersja typów (rzutowanie)
  - zaletę nowej notacji widać przy bardziej złożonych wyrażeniach, gdzie łatwo popełnić błąd związany z priorytetem operatorów

```
struct wezel {
   int wartosc;
   p_wezel* lewy;
   p_wezel* prawy;
};

typedef struct wezel* p_wezel;

p_wezel ptrX = new wezel; // new - operator C++
ptrX -> wartosc = 10;

void* ptrVoid;  //(1)
ptrVoid = ptrX; //(2)

int i;

i = ((p_wezel)ptrVoid) -> wartosc; // ANSI C
i = p_wezel (ptrVoid)  -> wartosc; //C++
```



biblioteka STL
(ang. Standard Template Library)

- STL w pigułce

pokaz siły języka C++

STL będzie omawiane później.
To temat na oddzielny wykład(y)

```
// kontener vector (inne używane nazwy: klasa vector, wzorzec vector)
// kopiuj cały plik (liniami) do wektora napisów

#include <string>
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int main()
{
  vector<string> v;                  // kontener stringów
  ifstream in("File.cpp");
  string line;

  while( getline(in, line) )
    v.push_back( line );             // dodaj linię na koniec kontenera

  // dodaj numer linii na wydruku:
  for(int i = 0; i < v.size(); i++)
    cout << i << ": " << v[i] << endl;     // przeciążony operator []
}                                          // pobiera kolejne dane wektora
```

z kontenerów duuużo łatwiej korzystać
niż napisać je samodzielnie !

kilka (z kilkudziesięciu) innych metod klasy `vector`:
`push_front(), insert(), erase()`

- STL w pigułce

```
// kopiuj cały plik (słowami) do wektora napisów
// (założenie: biały znak jako separator słów)

#include <string>
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int main()
{
  vector<string> words;   // kontener na słowa
  ifstream in("File.cpp");
  string word;

  while( in >> word )        // korzystamy z miłej cechy operatora >>
    words.push_back( word ); // która uwzględnia białe znaki jako separator

  for(int i = 0; i < words.size(); i++)
    cout << words[i] << endl;
}
```

- STL w pigułce

```
// tym razem wektor liczb

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

int main() {
  vector<int> v;
  for(int i = 0; i < 10; i++)
    v.push_back(i);

  for(int i = 0; i < v.size(); i++)
    cout << v[i] << ", ";
  cout << endl;

  for(int i = 0; i < v.size(); i++)
    v[i] = v[i] * 10;             // modyfikacja elementu w kontenerze
                                  // jest możliwa !

  for(int i = 0; i < v.size(); i++)
    cout << v[i] << ", ";
  cout << endl;
}
```


- STL w pigułce

```
// wykorzystanie iteratora klasy vector

#include <iostream>
#include <string>
#include <vector>
using namespace std;

int main(void)
{
  vector<string> words;           // wektor słów
  string str;
  do {
    cout << "Write a word (^D ends): ";
    cin >> str;
    if ( !str.empty() )
       words.push_back(str);      // włóż do wektora
  } while ( str != "quit" );

  vector<string>::iterator iter;  // deklaracja iteratora klasy vector
  for (iter = words.begin(); iter != words.end(); iter++)
        cout << *iter << endl;

 return 0;
}
```