Programowanie II R

dr Grzegorz Łach

#2 przypomnienie

konwersja typów

ang. type cast

```
double a = 12.5;
int b = a;

cout << b; // wyświetla "12"
```

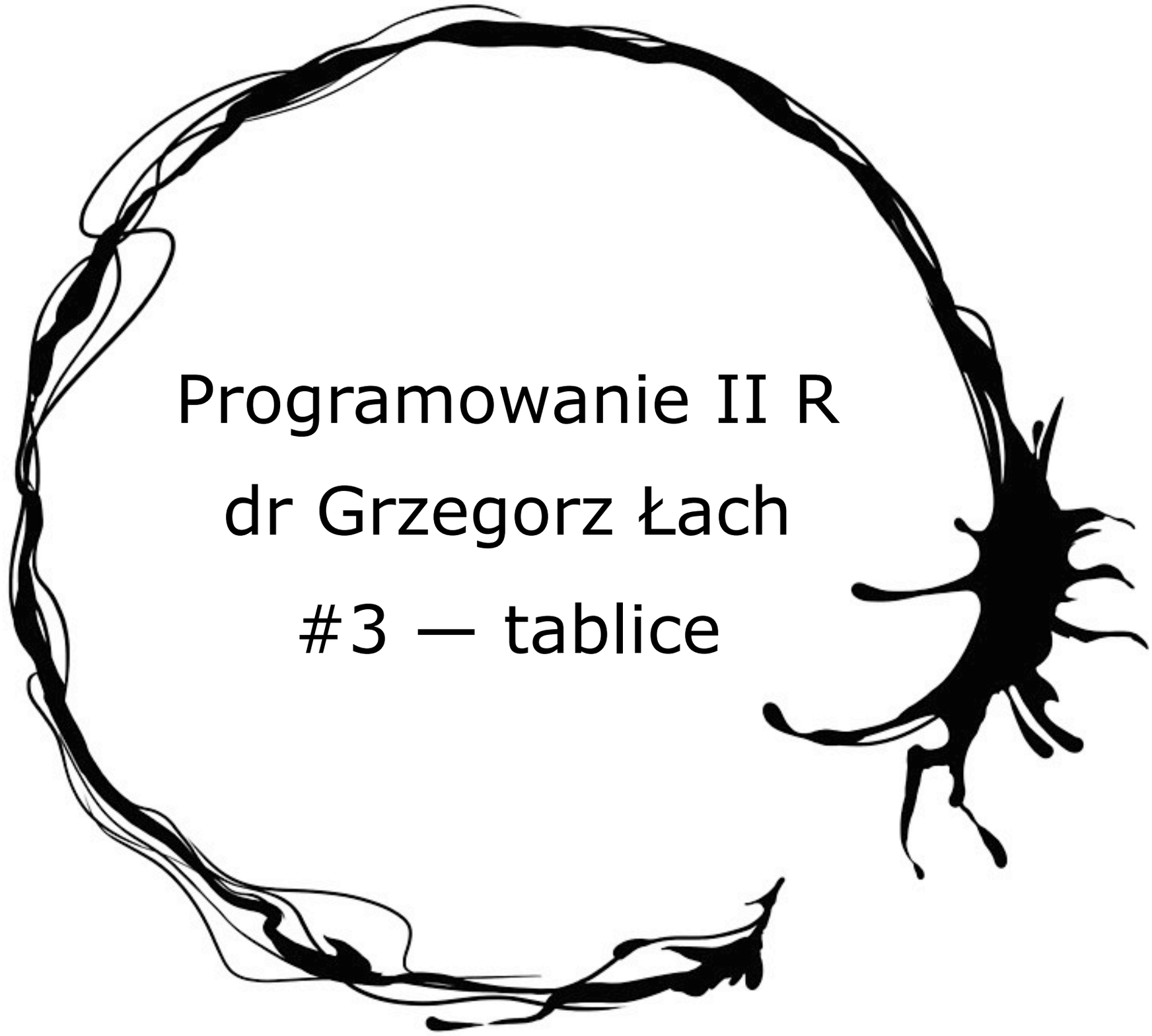
```
double a = 12.5;
int b = a%2;

cout << b;
```

```
double a = 12.5;
int b = ((int) a)%2;

cout << b; // wyświetla „0"
```

stałe

```
85              // decimal
0213            // octal
0x4b            // hexadecimal
30              // int
30u             // unsigned int
30l             // long
30ul            // unsigned long
```

```
3.14159     // Legal
314159E-5   // Legal
1.5F        // Legal
510E        // Illegal: incomplete exponent
210f        // Illegal: no decimal or exponent
.e55        // Illegal: missing integer or fraction
```



Programowanie II R

dr Grzegorz Łach

#3 — tablice

wskaźniki (ang. *pointers*)

```
typ zmiennaA;    // deklaracja zmiennej
typ *wskaznikA; // deklaracja wskaźnika do zmiennej

wskaznikA = &zmiennaA;
```

wskaźniki

```
typ zmiennaA;    // deklaracja zmiennej
typ *wskaznikA; // deklaracja wskaźnika do zmiennej

wskaznikA = &zmiennaA;
```

```
1  int * calkowita;
2  char * znak;
3  double * zmiennoprzecinkowa;
```

wskaźniki do zmiennych różnych typów mają różne typy
(ale konwersja jest domyślna
i nie powoduje komunikatu o błędzie)

wskaźniki

```
typ zmiennaA;    // deklaracja zmiennej
typ *wskaznikA; // deklaracja wskaźnika do zmiennej

wskaznikA = &zmiennaA;
```

```
wskaznikA = &zmiennaA;
```

zmienna ➔ wskaźnik

```
typ zmiennaB;    // deklaracja zmiennej tego samego typu

zmiennaB = *wskaznikA;
```

wskaźnik ➔ zmienna

wskaźniki

```
1  // my first pointer
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    int pierwsza, druga;
8    int * wskaznik;
9
10   wskaznik = &pierwsza;
11   *wskaznik = 10;
12   wskaznik = &druga;
13   *wskaznik = 20;
14   cout << „pierwsza: " << pierwsza << '\n';
15   cout << „druga: " << druga << '\n';
16   return 0;
17 }
```

```
pierwsza: 10
druga: 20
```

„*wskaźnik-do-zmiennej” to JEST wskazywana zmienna

wskaźniki — zagadka

```
1  // more pointers
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    int pierwsza = 5, druga = 15;
8    int * p1, * p2;
9
10   p1 = &pierwsza;
11   p2 = &druga;
12   *p1 = 10;
13   *p2 = *p1;
14   p1 = p2;
15   *p1 = 20;
16
17   cout << „pierwsza: " << pierwsza << '\n';
18   cout << „druga: " << druga << '\n';
19   return 0;
20 }
```

wskaźniki — zagadka

```
1  // more pointers
2  #include <iostream>
3  using namespace std;
4
5  int main ()
6  {
7    int pierwsza = 5, druga = 15;
8    int * p1, * p2;
9
10   p1 = &pierwsza;
11   p2 = &druga;
12   *p1 = 10;
13   *p2 = *p1;
14   p1 = p2;
15   *p1 = 20;
16
17   cout << „pierwsza: " << pierwsza << '\n';
18   cout << „druga: " << druga << '\n';
19   return 0;
20 }
```

```
pierwsza: 10
druga: 20
```

deklarowanie wskaźników — uwaga(1):

```
int * p1, * p2;
```

OK

```
int * p1, p2;
```

niepoprawnie

deklarowanie wskaźników — uwaga(2):

```
int zmienna;
int * wskaznik = &zmienna;
```

OK

```
int zmienna;
int * wskaznik;
wskaznik = &zmienna;
```

OK

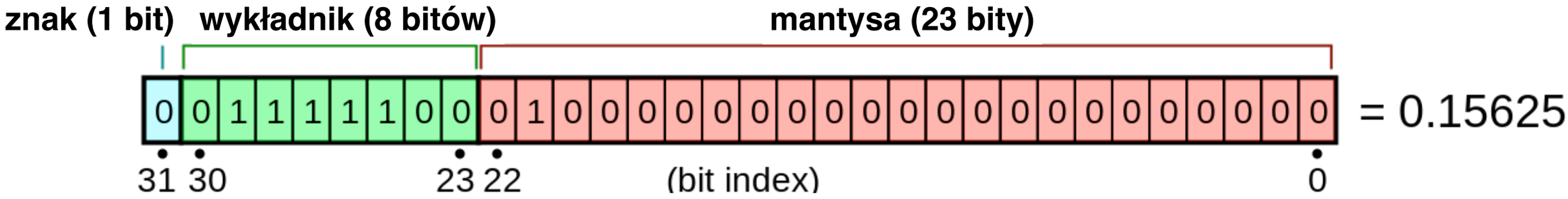
```
int zmienna;
int * wskaznik;
*wskaznik = &zmienna;
```

niepoprawnie

```
int zmienna;
int * wskaznik1 = &zmienna;
int * wskaznik2 = wskaznik1;
```

OK

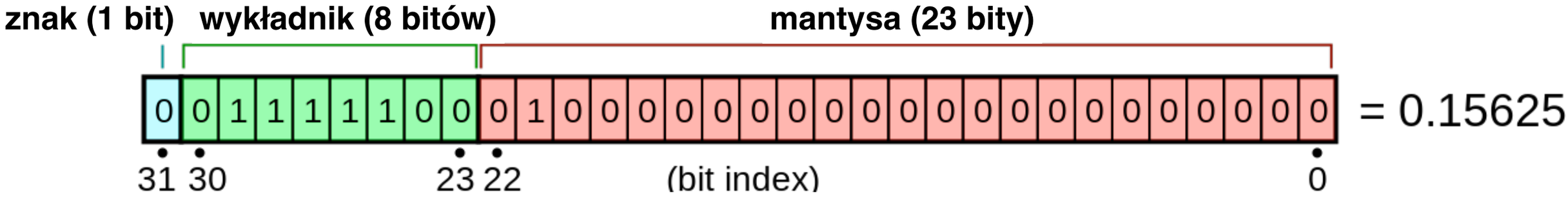
Jak reprezentowane są zmienne *float* (standard IEEE 754)



$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127} \times (1.b_{22}b_{21}\ldots b_0)_2$$

$$\text{value} = (-1)^{\text{sign}} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

Jak reprezentowane są zmienne *float* (standard IEEE 754)



$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\ldots b_{23})_2 - 127} \times (1.b_{22}b_{21}\ldots b_0)_2$$

$$\text{value} = (-1)^{\text{sign}} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i}\, 2^{-i}\right)$$

$$\begin{aligned}
1 + 2^{-23} &\approx 1.000\,000\,119, \\
2 - 2^{-23} &\approx 1.999\,999\,881, \\
2^{-126} &\approx 1.175\,494\,35 \times 10^{-38} \\
2^{+127} &\approx 1.701\,411\,83 \times 10^{+38}
\end{aligned}$$

„Quake III Arena”

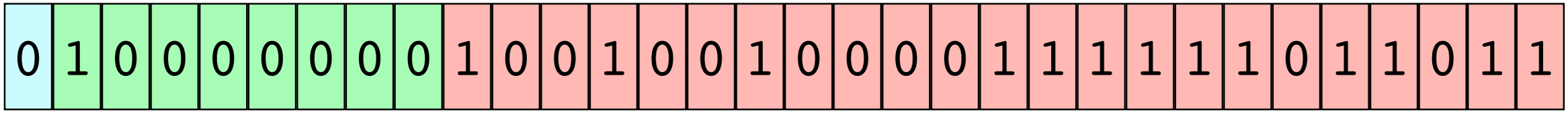
szybki algorytm obliczania 1/sqrt(x)
aka „what the f*ck?”

```
float Q_rsqrt( float number )
{
long i;
float x2, y;
const float threehalfs = 1.5F;

x2 = number * 0.5F;
y  = number;
i  = * ( long * ) &y;                       // evil floating point bit level hacking
i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
y  = * ( float * ) &i;
y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

return y;
}
```

```
long i;
float x2, y;
const float threehalfs = 1.5F;

x2 = number * 0.5F;
y  = number;
i  = * ( long * ) &y;                       // evil floating point bit level hacking
i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
y  = * ( float * ) &i;
y  = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
```

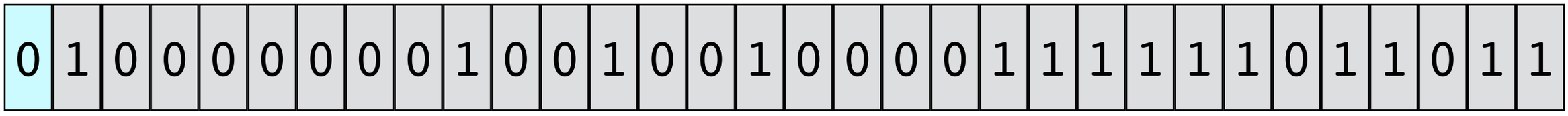
jeśli argumentem będzie np. 3.14159265..., to:

y



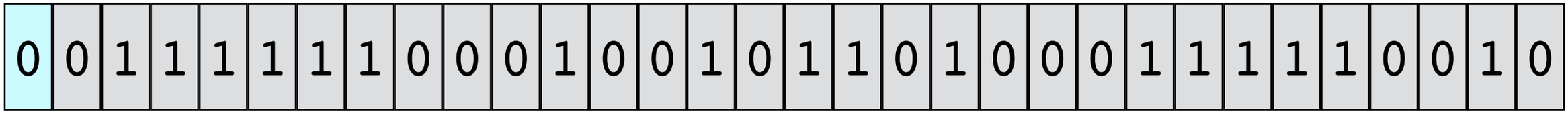

```
i   = * ( long * ) &y;
```

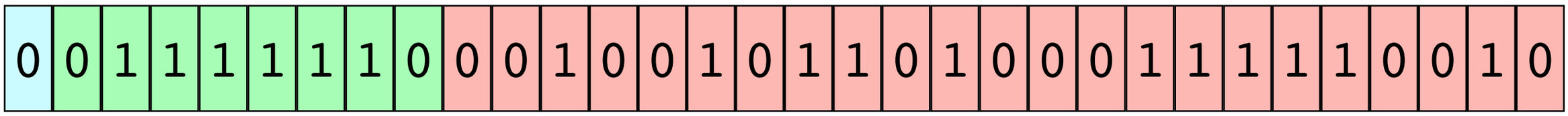
i




```
i   = 0x5f3759df - ( i >> 1 );
```

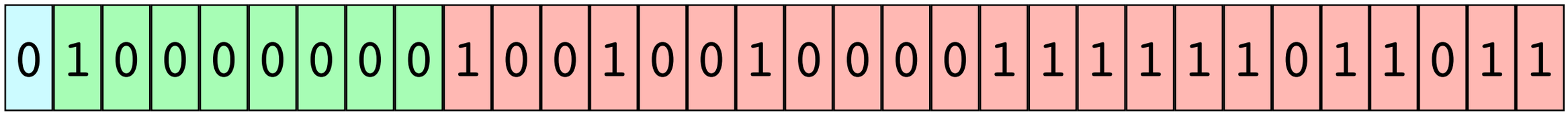
i





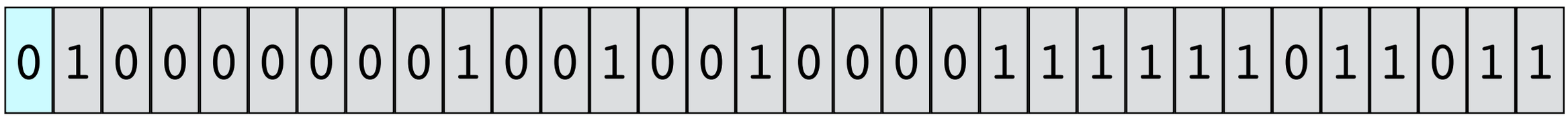
```
                    float y = 3.141592;
```

y



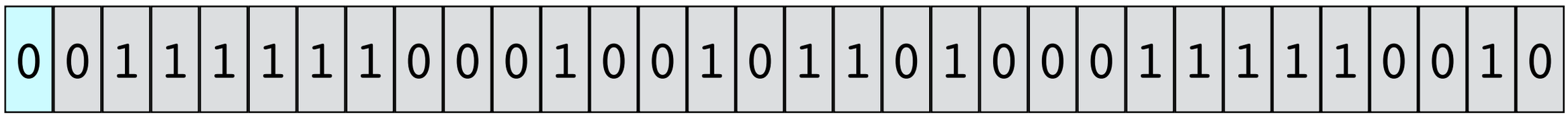

```
i   = * ( long * ) &y;
```

i

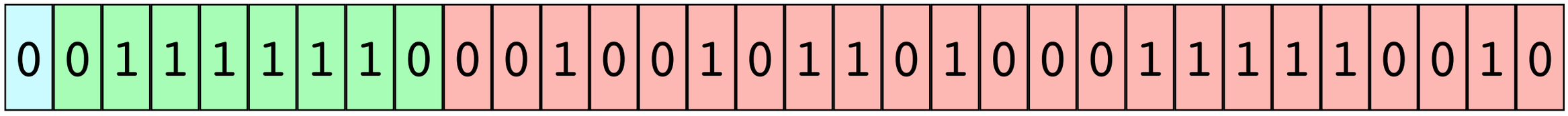



```
i   = 0x5f3759df - ( i>>1 );
```

i




```
y   = * ( float * ) &i;
```



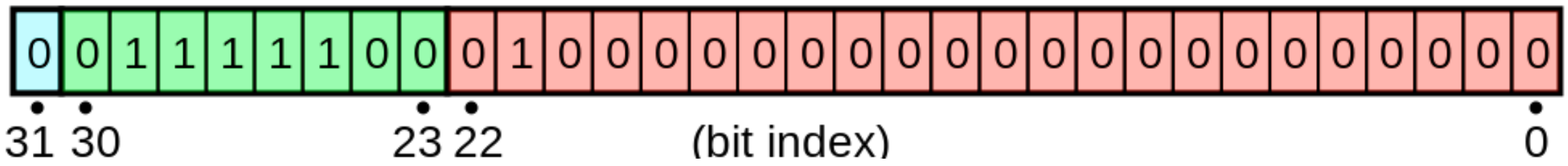

```
                                y = 0.573516;
```

1 iteracja metody Newtona — `0.563957;`

2 iteracja metody Newtona — `0.564189;`

dokładny wynik — `0.56418958...`

Jak reprezentowane są zmienne *double*

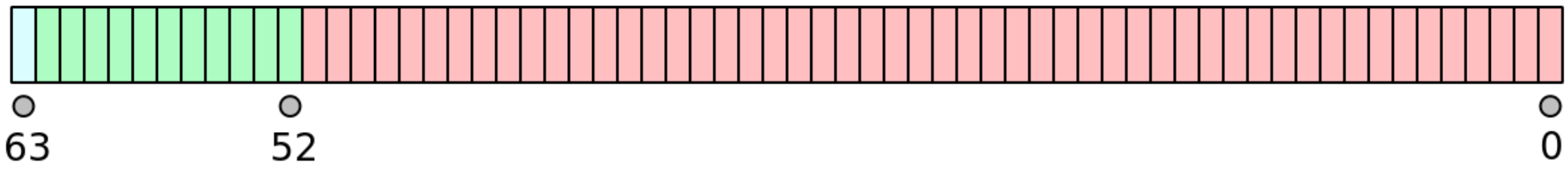


$$1 + 2^{-23} \approx 1.000\,000\,119,$$

$$2 - 2^{-23} \approx 1.999\,999\,881,$$

$$2^{-126} \approx 1.175\,494\,35 \times 10^{-38}$$

$$2^{+127} \approx 1.701\,411\,83 \times 10^{+38}$$

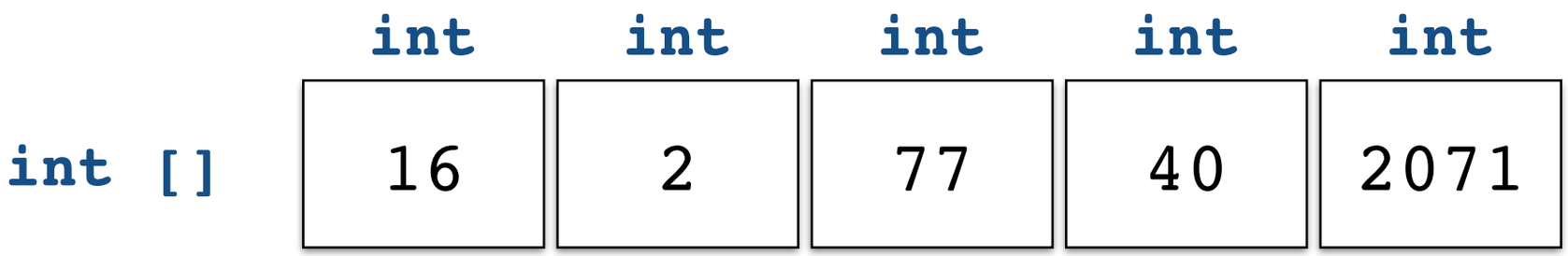


Tablice statyczne (deklarowane)
— rozmiar tablicy musi być znany podczas kompilacji
— deklarowane jak zmienne
— zakres ważności definicji — blok ({…})

Tablice dynamiczne (tworzone operatorem **`new`**)
— rozmiar tablicy nie musi być znany z góry
— nie mają nazwy, dostęp przez wskaźniki
— zakres ważności — od **`new`** do **`delete`**

Kontenery STL (np. **`<vector>`**, **`<deque>`**, **`<list>`**)
— rozmiar nie musi być znany z góry
— rozmiar może być zmieniany (zwiększany, zmniejszany)
— można (ale nie trzeba) zawczasu zarezerwować pamięć

tablice



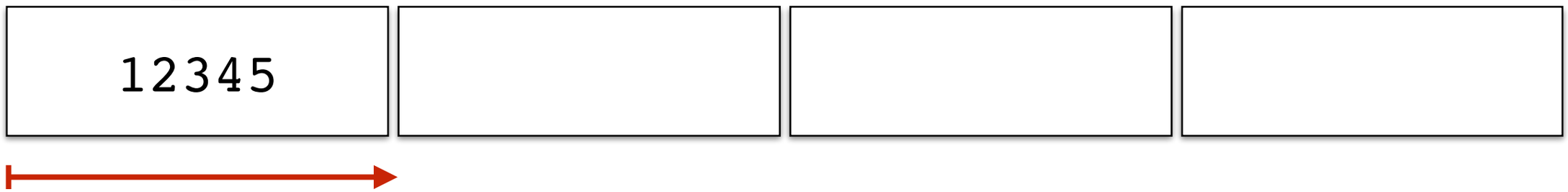

```
1  // przykład użycia tablic
2  #include <iostream>
3  using namespace std;
4
5  int foo [] = {16, 2, 77, 40, 2071};
6  int n, result=0;
7
8  int main ()
9  {
10   for ( n=0 ; n<5 ; ++n )
11   {
12     result += foo[n];
13   }
14   cout << result;
15   return 0;
16 }
```

```
2206
```

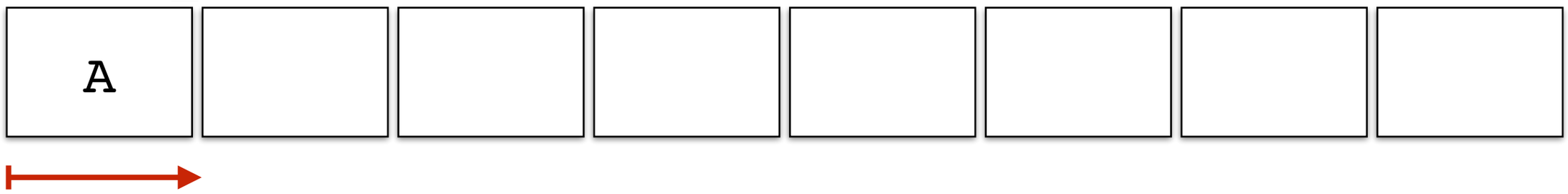
arytmetyka wskaźników

```
1  int * wskaznik_int;
2  char * wskaznik_char;
3  float * wskaznik_float;
```

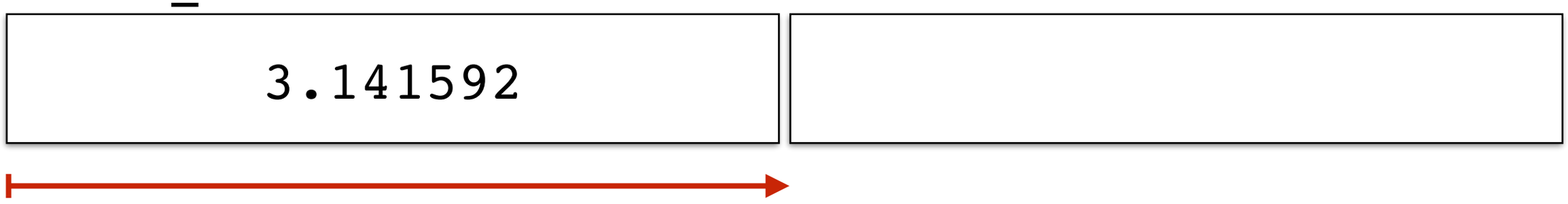
wskaznik_int+=1



wskaznik_char+=1



wskaznik_double+=1



tablice

```
// przykład użycia tablic
#include <iostream>
using namespace std;

int foo [] = {16, 2, 77, 40, 12071};
int n, result=0;

int main ()
{
  for ( n=0 ; n<5 ; ++n )
  {
    result += foo[n];
  }
  cout << result;
  return 0;
}
```

```
12206
```

dynamiczne alokowanie tablic

```
1 int * foo;
2 foo = new int [5];
```

```
foo = new int [5];
```

```
foo = new (nothrow) int [5];
```

nie kończy działania w przypadku braku miejsca w pamięci
(zwraca nullptr)

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

int main ()
{
  int i,n;
  int * p;
  cout << „Podaj ilość liczb: ”;
  cin >> i;
  p=new (nothrow) int[i];
  if (p == nullptr)
    cout << "Error: memory could not be allocated";
  else
  {
    for (n=0; n<i; n++)
    {
      cout << "liczba: ";
      cin >> p[n];
    }
    for (n=0; n<i; n++)
      cout << p[n] << ", ";
    delete[] p;
  }
  return 0;
}
```

```
Podaj ilość liczb: 5
liczba: 75
liczba: 436
liczba: 1067
liczba: 8
liczba: 32
75, 436, 1067, 8, 32,
```

co jeśli z góry nie wiemy ile będzie elementów?

STL (Standard Template Library)

— Kontenery (pojemniki)
— Iteratory
— Algorytmy

co jeśli z góry nie wiemy ile będzie elementów?

STL (Standard Template Library)

— Kontenery (pojemniki)
— Iteratory
— Algorytmy

<vector>

<valarray>

Pojemniki sekwencyjne

pozycja elementu w pojemniku zależy od
czasu i miejsca gdzie został wstawiony,
a nie zależy od jego wartości.

<vector>

<deque>

<list>

Pojemniki asocjacyjne

aktualna pozycja elementu w pojemniku
zależy od jego wartości

<map> <multimap>

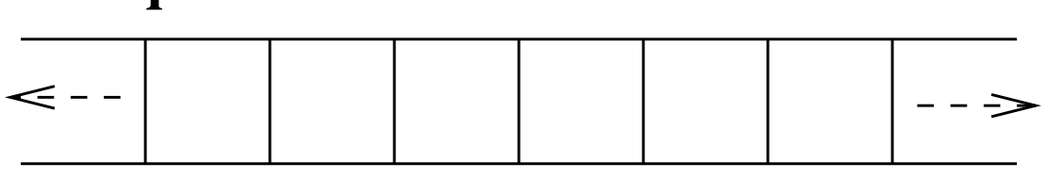
<set> <multiset> <hashset>

STL (Standard Template Library)

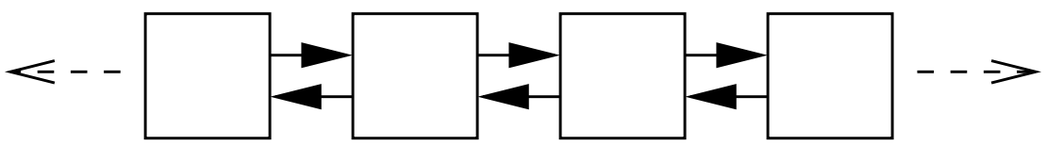
Pojemniki — struktury danych przechowujące zbiory obiektów

Vector

Deque

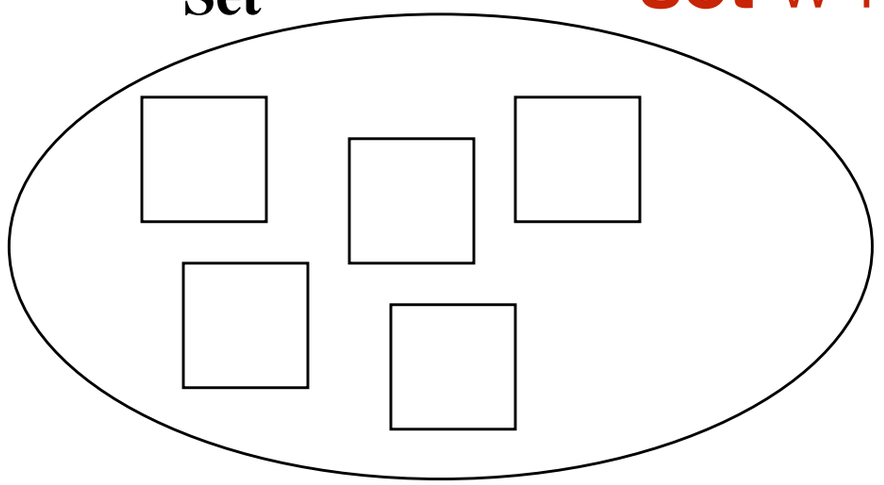


List



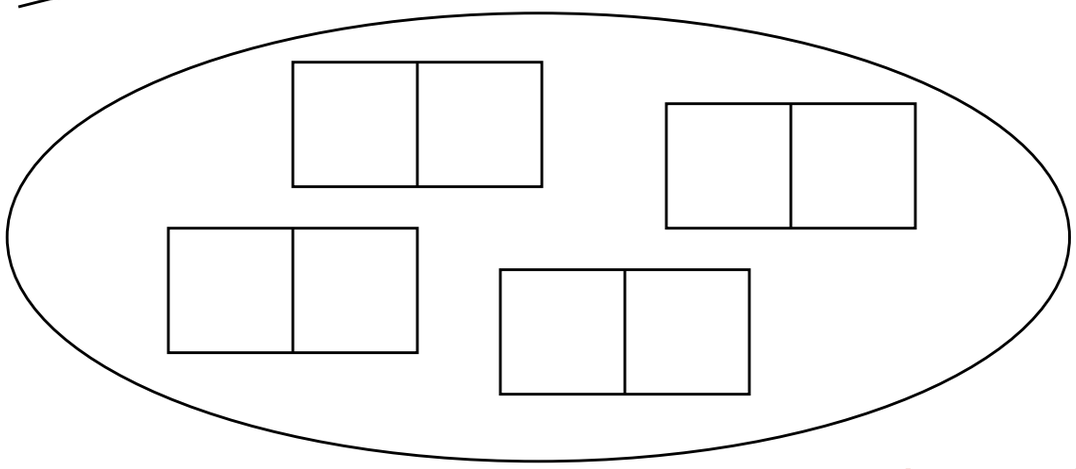
odpowiedniki
list w Pythonie

Set



odpowiednik
set w Pythonie

Map



odpowiednik
dict w Pythonie

Pojemniki sekwencyjne

pozycja elementu w pojemniku zależy od czasu i miejsca gdzie został wstawiony, a nie zależy od jego wartości.

<vector> **<deque>**

<valarray> **<list>**

Pojemniki asocjacyjne

aktualna pozycja elementu w pojemniku zależy od jego wartości

<map> **<multimap>**

<set> **<multiset>** **<hashset>**

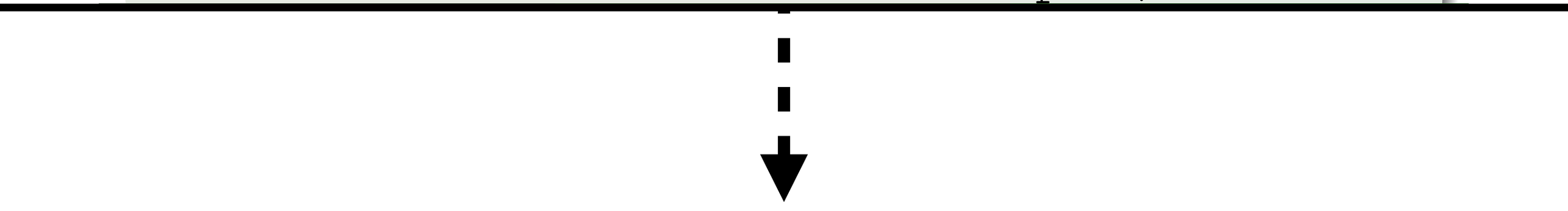
vector

- dynamiczna tablica
- operator indeksowania [] lub metoda `at()`
- szybkie dodawanie(`push_back()`) i usuwanie(`pop_back()`) elementów na końcu wektora

vector

- możliwe stworzenie pustego wektora i rozszerzanie go (zmniejszanie) na bieżąco
- najszybciej działa, jeśli z góry zarezerwujemy obszar pamięci – metoda `reserve()`

- Plik nagłówkowy `#include<vector>`
- Stworzenie pustego pojemnika do przechowywania obiektów typu `int`:
  `vector<int> vec;`
- Stworzenie pojemnika do przechowywania 10 obiektów typu `double`:
  `vector<double> vec(10);`
- Stworzenie pojemnika do przechowywania 10 obiektów typu `double` i inicjalizacja zerami
  `vector<double> vec(10,0.);`
- Iterator do poruszania się po pojemniku `vector`:
  `vector<double>::iterator pos;`



- **`vec.push_back(`**element**`)`** — dodawanie elementu na końcu wektora
- **`vec.pop_back()`** — usuwanie elementu z końca wektora
- **`vec.size()`** — zwraca liczbę elementów
- **`vec.at(i)`** — zwraca element o indeksie `i` (sprawdzanie czy prawidłowy indeks)
- **`vec[i]`** — zwraca element o indeksie `i` (bezsprawdzania czy prawidłowy indeks)
- **`vec.clear()`** — usuwa wszystkie elementy

vector — przykład bez iteratorów

```
// Deklaracja vector'a o długości 8
vector<int> vec(8);

// Tu vector ma dlugosc 8
cout << "rozmiar= " << vec.size() << endl;

// Wypelnianie vector'a
for(int i=0;i<vec.size();i++){
  vec[i]=(i*i*i);
}

// Wypisywanie na ekran
for(int i=0;i<vec.size();i++){
  cout << vec[i] << " ";
}
cout << endl;
```

Nie zadziała, gdybyśmy chcieli zmienić `vector` na inny pojemnik.

Iteratory — uogólnienie wskaźników na bardziej abstrakcyjne struktury danych

- Iteratory działają jak wskaźniki
- Nie tylko dla sekwencyjnych pojemników ! (smart pointers)
- Zwykły wskaźnik z języka C++ jest iteratorem

vector — przykład z iteratorem

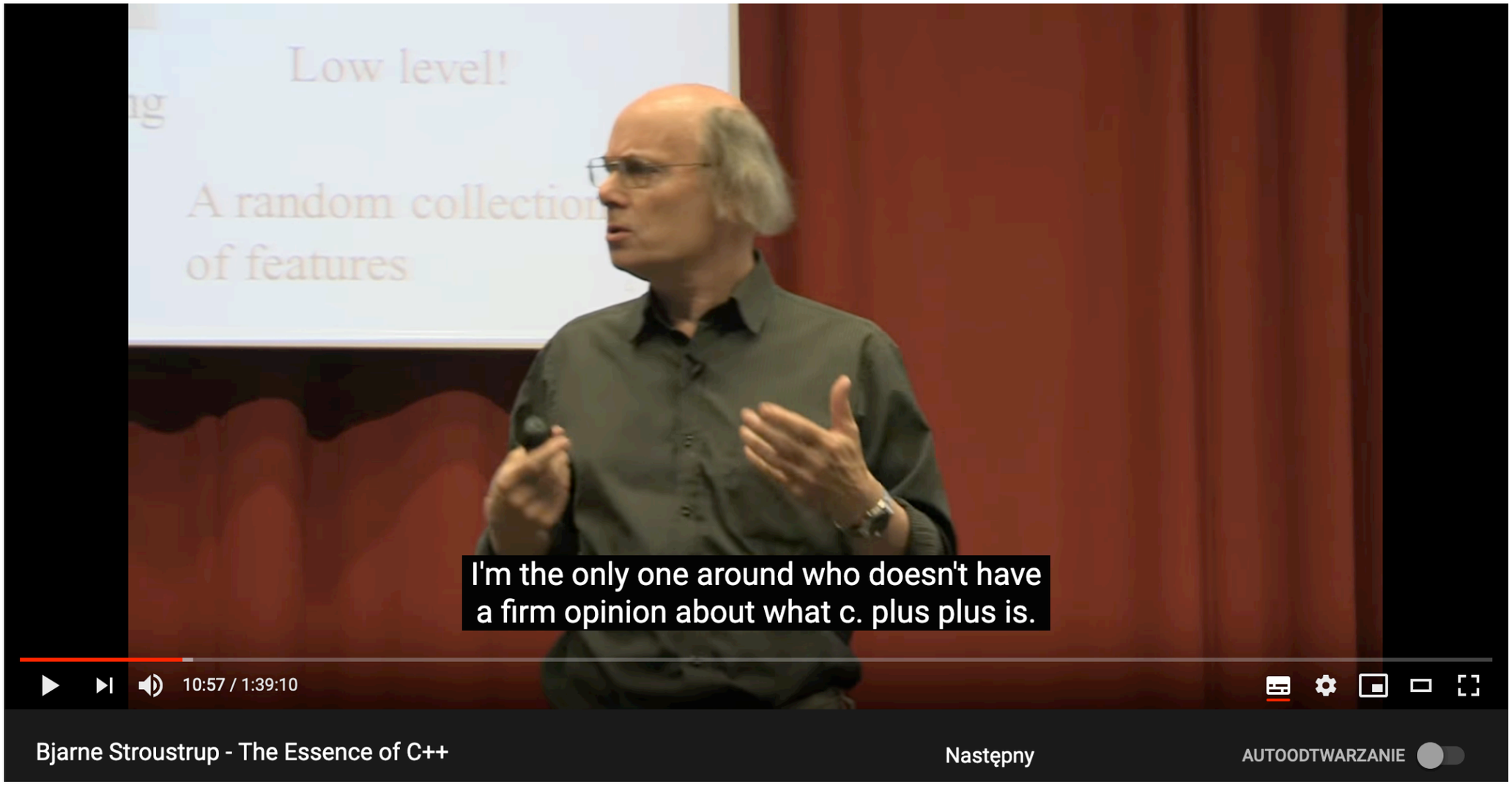
```
#include<iostream>
#include<vector>
using namespace std;
int main(){
  // Deklaracja vector'a
  vector<int> vec;
  // Wypelnianie vector'a
  for(int i=0;i<=8;i++){
    vec.push_back(i*i*i);
  }
  // Wypisywanie na ekran
  vector<int>::iterator pos;
  for(pos=vec.begin();pos<vec.end();pos++){
    cout << *pos << " ";
  }
  cout << endl;
  return 0;
}
```

- **`vec.push_back(`**element**`)`** — dodawanie elementu na końcu wektora
- **`vec.pop_back()`** — usuwanie elementu z końca wektora
- **`vec.size()`** — zwraca liczbę elementów
- **`vec.at(i)`** — zwraca element o indeksie **`i`** (sprawdzanie czy prawidłowy indeks)
- **`vec[i]`** — zwraca element o indeksie **`i`** (bezsprawdzania czy prawidłowy indeks)

?

- **`vec.clear()`** — usuwa wszystkie elementy

- C++99
- C++03
- C++11
- C++14
- C++17
- C++20



C++11: pętla `for` — „jak w Pythonie”

Modyfikacja pętli `for`

Przykład: Wypisywanie zawartości vector’a

```
vector<double> v(5,-1.5);
for (double elem : v){
   cout << elem << " ";
}
```

Analogicznie wypisywanie zawartości zwykłej tablicy

```
double tab[5]={1,2,3,4,5};
for (double elem : tab){
   cout << elem << " ";
}
```

C++11: pętla `for` — „jak w Pythonie”

```
// the C++03 way
for (std::vector<int>::iterator i = v.begin(); i != v.end(); ++i);

// the C++11 way
for (int &item: v);
// item will become, in order, all the things stored in v
// notice how we're referencing the item,that allows us to change it

for (const int &item: v);
// can't change item, we reference it for speed

for (int item: v);
// can't change item, we're passing it by value
```

http://www.cplusplus.com/articles/EzywvCM9/

`auto` - zastępczy typ zmiennej

Od standardu C++11:

`auto` nazwa_zmiennej = wartosc;

Zawsze działa:

```
int liczba = 15;
```

W C++11 możliwe także konstrukcje typu:

```
auto liczba = 15;
```

Modyfikacja pętli `for` i `auto`

```
vector<double> liczby(10,-1.5);

// Sumowanie kolejnych elementów wektora

double suma=0;
for (auto elem : liczby){
  suma+=elem;
}
cout << "suma= " << suma << endl;

// Modyfikacja wektora (mnożenie przez 2):
for (auto &elem :  liczby){

  elem*=2;
}
```

```
#include "set.h"
#include "hashset.h"
```

Member	Description
***s*.add(*value*);**	adds given value to set
***s*.clear();**	removes all elements of set
***s*.contains(*value*)**	`true` if given value is found
***s*.isEmpty()**	`true` if set contains no elements
***s*.isSubsetOf(*set*)**	`true` if *set* contains all of this one
***s*.remove(*value*);**	removes given value from set
***s*.size()**	number of elements in set
***s*.toString()**	e.g `"{3, 42, -7, 15}"`
ostr* << *s	print set to stream

vector

```
#include "set.h"
#include "hashset.h"
```

Member	Set	HashSet	Description
`s.add(value);`	O(log *N*)	**O(1)**	adds given value to set
`s.clear();`	O(*N*)	O(*N*)	removes all elements of set
`s.contains(value)`	O(log *N*)	**O(1)**	`true` if given value is found
`s.isEmpty()`	O(1)	O(1)	`true` if set contains no elements
`s.isSubsetOf(set)`	O(*N* log *N*)	O(*N*)	`true` if *set* contains all of this one
`s.remove(value);`	O(log *N*)	**O(1)**	removes given value from set
`s.size()`	O(1)	O(1)	number of elements in set
`s.toString()`	O(*N*)	O(*N*)	e.g `"{3, 42, -7, 15}"`
`ostr << s`	O(*N*)	O(*N*)	print set to stream

https://www.youtube.com/watch?v=STkHzXM9H-s


12
1
2
3
4
5
6
7
8
9
10
11
127
41
108
29
17
92
74
26

struktury

```
1  // array of structures
2  #include <iostream>
3  #include <string>
4  #include <sstream>
5  using namespace std;
6
7  struct movies_t {
8    string title;
9    int year;
10 } films [3];
11
12 void printmovie (movies_t movie)
13 {
14   cout << movie.title;
15   cout << " (" << movie.year << ")\n";
16 }
```

struktury

```
1  // array of structures
2  #include <iostream>
3  #include <string>
4  #include <sstream>
5  using namespace std;
6
7  struct movies_t {
8    string title;
9    int year;
10 };
11
12 void printmovie (movies_t movie)
13 {
14   cout << movie.title;
15   cout << " (" << movie.year << ")\n";
16 }
```

struktury

```
int main ()
{
  string mystr;
  int n;

  for (n=0; n<3; n++)
  {
    cout << "Enter title: ";
    getline (cin,films[n].title);
    cout << "Enter year: ";
    getline (cin,mystr);
    stringstream(mystr) >> films[n].year;
  }

  cout << "\nYou have entered these movies:\n";
  for (n=0; n<3; n++)
    printmovie (films[n]);
  return 0;
}
```

```
Enter title: Blade Runner
Enter year: 1982
Enter title: The Matrix
Enter year: 1999
Enter title: Taxi Driver
Enter year: 1976

You have entered these movies:
Blade Runner (1982)
The Matrix (1999)
Taxi Driver (1976)
```