C++17 i STL

Komponenty numeryczne

Globalne funkcje numeryczne

- W pliku nagłówkowym **`<cfloat>`** znajdują się definicje makr opisujących właściwości liczb zmiennoprzecinkowych. Należą do nich (wśród innych) następujące makra:
  - DBL_MIN – najmniejsza wartość typu double;
  - DBL_MAX – największa wartość typu double (ok. 1.798e+308);
  - DBL_EPSILON – najmniejsza wartość typu double, taka że 1.0+DBL_EPSILON != 1.0 po zaokrągleniach w arytmetyce zmiennopozycyjnej.

Globalne funkcje numeryczne

- W nagłówku **<cmath>** znajdują się standardowe funkcje matematyczne. Należą do nich (wśród innych) następujące funkcje:
  - abs(), ceil(), floor() – wartość bezwzględna i zaokrąglenia;
  - sqrt(), pow(), exp(), log() – potęgowanie, logarytmowanie;
  - sin(), cos(), tan(), atan() – funkcje trygonometryczne.
- Istnieją przeciążone wersje tych funkcji przyjmujące jako argument wartości typów float, double, long double oraz complex; typ zwrotny każdej z tych funkcji jest taki sam jak typ argumentu.

Liczby zespolone

- Liczby zespolone składają się z dwóch części — rzeczywistej oraz urojonej.
- Część urojona posiada tę właściwość, iż podniesiona do kwadratu daje w wyniku liczbę ujemną; część urojoną liczby zespolonej stanowi współczynnik i, będący pierwiastkiem kwadratowym liczby –1.
- STL w C++ udostępnia szablon klasy `complex<>` w pliku nagłówkowym **`<complex>`**, umożliwiający wykonywanie obliczeń na liczbach zespolonych.

Liczby zespolone

- Klasa `complex<>` jest zdefiniowana w następujący sposób:
  `template <typename T> class complex;`
- Parametr szablonu o nazwie T używany jest jako typ skalarny zarówno dla części rzeczywistej, jak też urojonej danej liczby zespolonej.
- STL udostępnia trzy specjalizacje klasy `comlex<>`:
  - `comlex<float>`,
  - `comlex<double>`,
  - `comlex<long double>`.

Tworzenie liczb zespolonych

- Konstruktory umożliwiają przekazanie wartości początkowej poprzez określenie jej części rzeczywistej oraz urojonej. W przypadku gdy nie zostaną one określone, nastąpi ich zainicjalizowanie zerem określonego typu.
- // liczba zespolona z częścią rzeczywistą oraz urojoną
  // - część rzeczywista: 4.0
  // - część urojona: 3.0
  `complex<double> c1(4.0, 3.0);`
- // liczba zespolona utworzona przy wykorzystaniu współrzędnych biegunowych
  // - moduł: 5.0
  // - kąt fazy: 0.75
  `complex<float> c2(polar(5.0,0.75));`

Tworzenie, kopiowanie przypisanie do liczby zespolonej

- Operatory przypisania stanowią jedyny sposób modyfikacji wartości istniejącej liczby zespolonej.
- Przypisania liczb zespolonych (z ewentualną operacją arytmetyczną):
  c1 = c2
  c1 += c2
  c1 -= c2
  c1 *= c2
  c1 /= c2

Tworzenie, kopiowanie i przypisanie do liczby zespolonej

- Funkcja pomocnicza o nazwie `polar()` umożliwia utworzenie liczby zespolonej, zainicjalizowanej za pomocą współrzędnych biegunowych (modułu oraz kąta fazy podanego w radianach). Przykład:
```
complex<double> c2(std::polar(4.2, 0.75));
```
- Funkcja pomocnicza o nazwie `conj()` umożliwia utworzenie liczby zespolonej zainicjalizowanej za pomocą wartości sprzężonej z inną liczbą zespoloną (wartość sprzężona z daną liczbą zespoloną tworzona jest poprzez zanegowanie jej części urojonej):
```
complex<double> c1(1.1, 5.5);
complex<double> c2(conj(c1));
```

Funkcje związane z dostępem do wartości liczby zespolonej

- Część rzeczywista i urojona liczby zespolonej c:
  `c.real()`, `real(c)`
  `c.imag()`, `imag(c)`
- Moduł liczby zespolonej:
  `abs(c)`
- Norma liczby zespolonej (kwadrat modułu):
  `norm(c)`
- Kąt fazy liczby zespolonej:
  `arg(c)`
- Liczba sprzężona z liczbą zespoloną:
  `conj(c)`

Arytmetyka liczb zespolonych

- Operacje arytmetyczne na liczbach zespolonych c1 i c2:
  - c1 – negacja
  c1 + c2 – suma
  c1 - c2 – różnica
  c1 * c2 – iloczyn
  c1 / c2 – iloraz

Porównywanie liczb zespolonych

- W celu porównania dwóch liczb zespolonych możesz jedynie sprawdzić, czy są one równe.
- Operatory == oraz != zostały zdefiniowane jako funkcje globalne, dlatego też jeden z argumentów może być wartością skalarną.
- Jeśli w charakterze argumentu użyjesz skalara, jest on interpretowany jako część rzeczywista, zaś część urojona posiada wartość 0.
- Inne operacje porównania (operatory <, <=, > i >=) nie są zdefiniowane. W konsekwencji niemożliwe jest użycie liczby zespolonej typu `complex<>` jako elementu kontenera asocjacyjnego.

Operatory wejścia-wyjścia dla liczb zespolonych

- Klasa `complex<>` udostępnia powszechnie stosowane operatory wejścia-wyjścia << oraz >>.
- Operacje strumieniowe:
  ```
  strm << c
  strm >> c
  ```
- Operator wyjściowy zapisuje liczbę zespoloną przy uwzględnieniu stanu bieżącego strumienia w postaci pary:
  (część_rzeczywista, część_urojona)
- Operator wejściowy umożliwia odczytanie liczb zespolonych zapisanych w jednej z poniższych postaci:
  (część_rzeczywista, część_urojona)
  (część_rzeczywista)
  część_rzeczywista

Funkcje trygonometryczne i wykładnicze dla liczb zespolonych

- Funkcje trygonometryczne dla liczb zespolonych:
  `sin(c)` – sinus
  `cos(c)` – cosinus
  `tan(c)` – tangens
  `asin(c)` – arcus sinus
  `acos(c)` – arcus cosinus
  `atan(c)` – arcus tangens
- Funkcje wykładnicze dla liczb zespolonych:
  `pow(c1, c2)` – potęgowanie
  `sqrt(c)` – pierwiastek
  `exp(c)` – funkcja eksponencjalna
  `log(c)` – logarytm naturalny

Algorytmy numeryczne

- Aby móc użyć algorytmów numerycznych, należy dołączyć plik nagłówkowy **`<numeric>`**.
- Za pomocą algorytmów numerycznych można przetwarzać nie tylko pojedyncze wartości numeryczne ale także całe sekwencje tych wartości.
- Założenie: elementy numeryczne są zgromadzone w kolekcji sekwencyjnej.

Algorytmy numeryczne – obliczanie wartości wypadkowej ciągu

- Obliczanie wartości wypadkowej jednego ciągu:

```
T accumulate (InputIterator beg,
              InputIterator end, T initValue)
T accumulate (InputIterator beg,
              InputIterator end,
              T initValue, BinaryFunc op)
```

- Pierwsza postać algorytmu oblicza i zwraca sumę wartości `initValue` oraz wszystkich elementów z zakresu [beg,end). W szczególności, dla każdego elementu wykonuje ona operację:
  `initValue = initValue + elem`
- Druga postać algorytmu oblicza i zwraca wynik wywołania operacji op wobec wartości `initValue` oraz wszystkich elementów z zakresu [beg,end). W szczególności, dla każdego elementu wykonuje ona operację:
  `initValue = op(initValue, elem)`

Algorytmy numeryczne – obliczanie iloczynu skalarnego dwóch ciągów

- Obliczanie iloczynu skalarnego dwóch ciągów:

```
T inner_product (InputIterator beg1, InputIterator end1,
                 InputIterator beg2, T initValue)
T inner_product (InputIterator beg1, InputIterator end1,
                 InputIterator beg2, T initValue ,
                 BinaryFunc op1 , BinaryFunc op2)
```

- Pierwsza postać algorytmu oblicza i zwraca iloczyn skalarny wartości `initValue` oraz wszystkich elementów z zakresu [beg1,end1) w kombinacji z elementami z zakresu rozpoczynającego się od pozycji `beg2` . W szczególności, dla każdej pary elementów wykonuje ona operację:

```
initValue = initValue + elem1 * elem2
```

- Druga postać algorytmu dla każdej pary elementów wykonuje operację:

```
initValue = op1(initValue, op2(elem1, elem2))
```

Liczby pseudolosowe

- Standardowe narzędzia do generowania liczb pseudolosowych znajdują się w pliku nagłówkowym **<random>**.
- Liczby losowe są sekwencją wartości wytworzonych przez generator pseudolosowy przy użyciu matematycznych wzorów.
- Zastosowanie liczb losowych:
  - symulacje,
  - gry,
  - algorytmach probablistyczne,
  - kryptografia,
  - testowanie.

Mechanizmy losowości

- **Równomierny generator liczb losowych** to obiekt funkcyjny zwracający wartości całkowite bez znaku z rozkładem jednostajnym dla zadanego zakresu.
- **Mechanizm liczb losowych** jest równomiernym generatorem liczb, który można utworzyć w domyślnym stanie E{} lub w stanie określonym przez ziarno (ang. seed) E{s}.
- **Adaptacja mechanizmu liczb losowych** to mechanizm generowania liczb losowych, który pobiera wartości wytwarzane przez inny mechanizm generowania liczb losowych i przepuszcza je przez własny algorytm w celu utworzenia zbioru wartości o innych cechach losowości.
- **Rozkład liczb losowych** to obiekt funkcyjny zwracający wartości o rozkładzie zgodnym z powiązaną matematyczną funkcją gęstości prawdopodobieństwa.

Mechanizmy losowości

- Dla programisty generator liczb losowych to mechanizm połączony z rozkładem – mechanizm wytwarza równomiernie rozłożoną sekwencję wartości, a rozkład modyfikuje jej kształt do pożądanej postaci.
- Przykład (rozkład normalny `normal_distribution` z domyślnym mechanizmem `default_random_engine`):

```
auto gen = bind(
    normal_distribution<double>{15,4.0},
    default_random_engine{}
);
…
for (int i=0; i<500; ++i) cout << gen();
```

Mechanizmy losowości

- Najczęściej programiści potrzebują prostego równomiernego rozkładu liczb całkowitych z określonego przedziału:

```
// utwórz domyślny mechanizm losowości
std::default_random_engine dre;
// użyj mechanizmu do wygenerowania liczb
// całkowitych ze zbioru {10, …, 20}
std::uniform_int_distribution<int>
    di(10,20);
…
std::cout << di(dre) << std::endl;
```

Mechanizmy losowości

- Często też programiści potrzebują prostego równomiernego rozkładu liczb zmiennoprzecinkowych z określonego przedziału:

```
// utwórz domyślny mechanizm losowości
std::default_random_engine dre;
// użyj mechanizmu do wygenerowania liczb
// całkowitych z przedziału [-1, 1)
std::uniform_real_distribution<double>
    dr(-1, 1);
…
std::cout << dr(dre) << std::endl;
```

Mechanizmy i rozkłady

- **Mechanizmy** służą jako stanowe źródło losowości. Są to obiekty funkcyjne, zdolne do generowania dodatnich wartości z jednorodnym rozkładem wzdłuż określonego przedziału.
- **Rozkłady** służą do określania gęstości wartości losowych w zakresie określonym parametrami podanymi przez użytkownika.
- Liczbę losową generuje się za pośrednictwem wywołania operatora wywołania funkcji `operator()` na rzecz obiektu dystrybucji, z mechanizmem losowości jako argumentem wywołania.
- Jeśli nie chcemy zaczynać od przewidywalnej wartości liczbowej, powinniśmy jawnie ustawić przypadkowy stan generatora (na podstawie jakiegoś czynnika niezależnego od samego kodu programu, na przykład liczby milisekund pomiędzy dwoma kliknięciami przycisku myszy) - do konstruktora mechanizmu losowości trzeba przekazać **zarodek losowości**:

```
unsigned int seed = ... ;
std::default_random_engine dre(seed);
```

Ostrożnie z tymczasowymi obiektami mechanizmu losowości

- Nie powinno się przekazywać do algorytmów obiektu mechanizmu losowości tworzonego wyłącznie na potrzeby wywołania algorytmu (czyli na przykład obiektu tymczasowego). – za każdym razem przy tworzeniu i inicjalizowaniu mechanizmu losowości przybiera on taki sam w pełni określony stan.

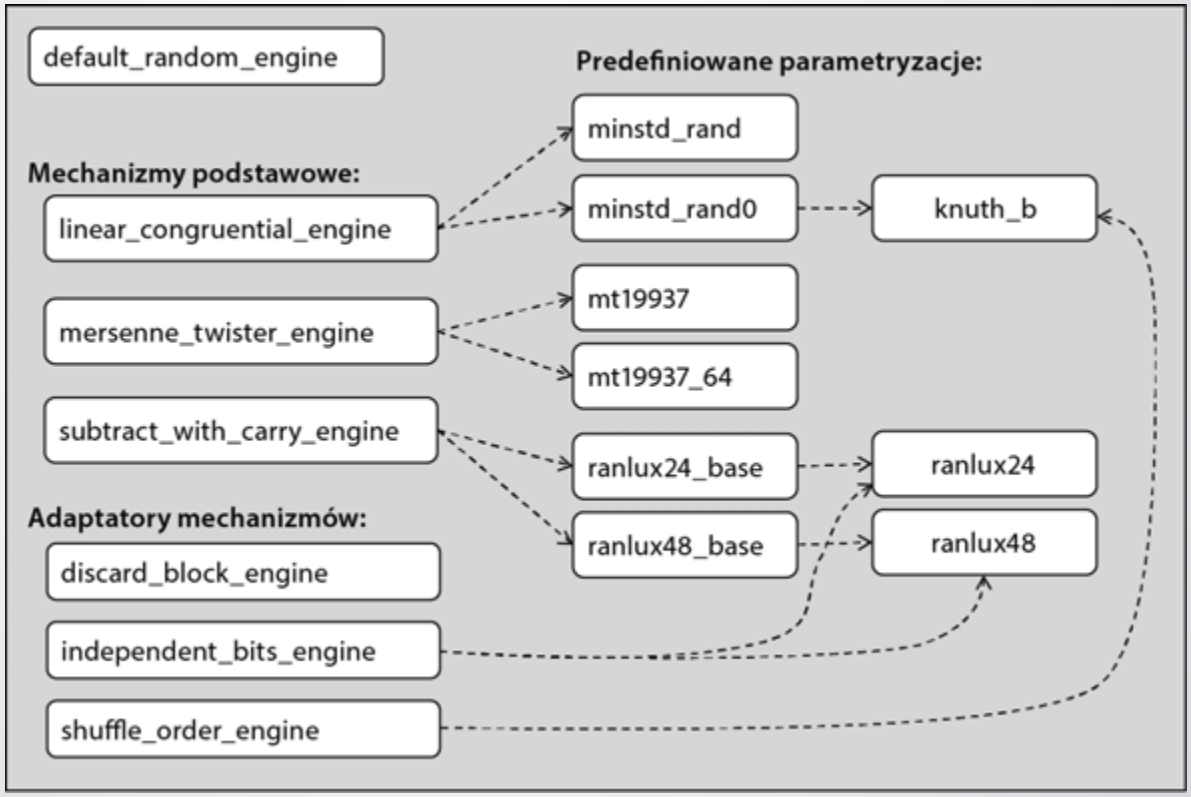
Nie używaj mechanizmów losowości bez rozkładów

- Naiwny programista zapytałby teraz: po co nam rozkłady, czy nie wystarczy używać bezpośrednio mechanizmów losowości i generowanych przez nich wartości losowych? A jeśli zakres generowanych wartości jest nieodpowiedni, czy nie wystarczy po prostu ograniczyć go operacją modulo (operator %) na wartościach wynikowych?
- Technika wyznaczania liczb losowych przez wyrażenie rand()%n w praktyce zawodzi z dwóch powodów:
  - wiele implementacji generatorów pseudolosowych daje reszty z małych n niebędące mocno losowe (na przykład nie jest rzadkością, że kolejne wyniki rand() są na przemian parzyste i nieparzyste);
  - jeśli wartość n jest duża, a wartość maksymalna generowanych wartości nie jest podzielna bez reszty przez n, niektóre reszty z dzielenia będą pojawiać się częściej niż inne.

Mechanizmy losowości

- Biblioteka standardowa C++ udostępnia 16 mechanizmów wartości losowych, które w połączeniu z dystrybucjami nadają się do generowania liczb losowych albo tasowania sekwencji wartości.
- Mechanizm losowości jest stanowy – jego stan definiuje sekwencję wartości losowych (które nie są liczbami losowymi); każde wywołanie funkcji (za pośrednictwem operatora wywołania funkcji operator()) zwraca kolejną wartość losową z sekwencji i zmienia wewnętrzny stan obiektu mechanizmu losowości.
- Przejścia pomiędzy stanami i generowane wartości są ściśle określone – jedynym wyjątkiem jest tu mechanizm domyślny, który może być wybrany różnie na różnych platformach.

Mechanizmy losowości



Mechanizmy losowości

- Mechanizm losowości jest obiektem funkcyjnym, to znaczy obiektem zachowującym się jak funkcja.
- Przykład:

```
void printNumbers (default_random_engine& dre)
{
    for (int i=0; i<6; ++i)
        cout << dre() << " „;
    cout << endl;
}
```

Mechanizmy losowości

- Mechanizm losowości jest stanowym źródłem losowości – jego stan definiuje sekwencję wartości losowych (które nie są liczbami losowymi).
- Każde wywołanie funkcji (za pośrednictwem operatora wywołania funkcji operator()) zwraca kolejną wartość losową z sekwencji i zmienia wewnętrzny stan obiektu mechanizmu losowości.
- Jeśli aplikacja potrzebuje wartości nieprzewidywalnych, trzeba zadbać o ustawienie stanu mechanizmu losowości na bazie jakiegoś niezależnego czynnika albo zachowania zewnętrznego.
- Przy ponownej deklaracji tego samego mechanizmu zostanie on zainicjalizowany tym samym stanem początkowym, wygeneruje więc tę samą sekwencję wartości losowych. Użycie tego samego stanu mechanizmu można też osiągnąć przez wielokrotne przekazanie mechanizmu do funkcji przez wartość.

Mechanizmy losowości – generowane wartości

- Wygenerowane przez mechanizm losowości wartości są nieujemnymi wartościami całkowitymi.
- Dokładny typ zwracanych wartości jest określony jawnie dla wszystkich mechanizmów poza domyślnym `default_random_engine`, gdzie typ jest zależny od implementacji.
- Dla każdego mechanizmu losowości typ generowanych wartości określa wewnętrzna definicja typu `result_type`, a granice zakresu generowanych wartości podają statyczne funkcje składowe `min()` i `max()` (zakres obejmuje oba ekstrema).

Mechanizmy losowości w szczegółach

- **Mechanizmy podstawowe** implementujące różne algorytmy generowania wartości losowych:
  - **linear_congruential_engine**,
  - **mersenne_twister_engine**,
  - **subtract_with_carry_engine**.
- **Adaptatory mechanizmów** stosowane do mechanizmów podstawowych:
  - **discard_block_engine** adaptująca mechanizm przez każdorazowe odrzucanie zadanej liczby wygenerowanych wartości,
  - **independent_bits_engine** adaptująca mechanizm pod kątem generowania wartości losowych o określonej liczbie bitów,
  - **shuffle_order_engine** adaptująca mechanizm przez permutację kolejności wartości generowanych przez mechanizm.
- **Adaptatory z predefiniowanymi parametrami**.

Rozkłady

- Rozkłady przekształcają wartości losowe generowane przez mechanizmy losowości na prawdziwie użyteczne liczby losowe.
- Dystrybucja prawdopodobieństwa generowanych liczb zależy od typu rozkładu, dodatkowo parametryzowanego zgodnie z potrzebami programisty.
- Zdefiniowane rozkłady:
  - rozkład jednostajny (uniform_int_distribution, uniform_real_distribution);
  - rozkład Bernoulliego (bernoulli_distribution, binomial_distribution, geometric_distribution);
  - rozkład Poissona (poisson_distribution, exponential_distribution);
  - rozkład normalny (normal_distribution, chi_squared_distribution, student_t_distribution);
  - rozkład próbkujący (discrete_distribution).

Rozkłady

- Przykłady:

```
uniform_int_distribution<> d(0, 20);
d.a();
d.b();

std::knuth_b e;

std::uniform_real_distribution<> ud(0, 10);

std::normal_distribution<> nd;
```

Urządzenie losowe

- Jeśli implementacja ma możliwość dostarczenia prawdziwie losowego generatora liczb, to źródło tych liczb ma postać równomiernego generatora liczb losowych o nazwie random_device.

Losowanie liczb w stylu C

- W pliku nagłówkowym **<cstdlib>** znajduje się prosta podstawa do generowania liczb losowych:
  - funkcja rand() – liczba pseudolosowa z przedziału od 0 do RAND_MAX;
  - funkcja srand(unsigned) – podanie ziarna dla generatora liczb pseudolosowych.
- Utworzenie dobrego generatora liczb losowych nie jest łatwe i niestety nie wszystkie systemy zawierają dobrą funkcję rand().
- Często wyniki do przyjęcia daje operacja:
  `int((rand() / (RAND_MAX + 1.0)) * n)`

Tablica wartości numerycznych

- Standardowa biblioteka C++ od wersji C++98 udostępnia klasę `valarray`, służącą do przetwarzania tablic ciągu wartości numerycznych. Definicja tej klasy znajduje się w pliku nagłówkowym **`<valarray>`**.
- Tablica taka jest reprezentacją matematycznego pojęcia liniowej sekwencji wartości.
- Tablica wartości numerycznych może być wykorzystana w charakterze podstawy operacji zarówno na macierzach, jak też wektorach, jak również do odpowiednio wydajnego przetwarzania równań wielomianowych.
- W ujęciu technicznym tablice `valarray` są jednowymiarowymi tablicami elementów ponumerowanych sekwencyjnie od zera.

Tablica wartości numerycznych

- Najważniejszym powodem utworzenia typu `valarray` było dostarczenie znanych z języka Fortran narzędzi do pracy z gęstymi wielowymiarowymi tablicami z typowymi dla tego języka możliwościami optymalizacji.
- Przykłady deklaracji i inicjalizacji:
  valarray<double> v0; // do v0 można przypisać później
  valarray<float> v1(1000); // 1000 elementów o wartości 0.0F
  valarray<int> v2(−1,2000); // 2000 elementów o wartości −1
  valarray<int> v5 {−1,2000}; // dwa elementy
- Na `valarray` można pracować za pomocą iteratorów:
  - begin(v) zwraca iterator o dostępie swobodnym do pierwszego elementu v;
  - end(v) zwraca iterator o dostępie swobodnym do miejsca za ostatnim elementem v.

Tablica wartości numerycznych

- Konstrukcja `valarray` ma wspomagać wykonywanie obliczeń, a więc obsługuje bezpośrednio wiele operacji numerycznych:
  - n = va.size() – n jest liczbą elementów w va
  - t = va.sum() – t jest sumą elementów va obliczoną przy użyciu operatora +=
  - t = va.min() – t jest najmniejszym elementem va znalezionym przy użyciu operatora <
  - T = va.max() – t jest największym elementem va znalezionym przy użyciu operatora <
  - a = va.shift(n) – liniowe przesunięcie w lewo elementów
  - a = va.cshift(n) – cykliczne przesunięcie w lewo elementów
  - a = va.apply(f) – zastosowanie f: wartość każdego elementu a to f(va[i])

Tablica wartości numerycznych

- Operacje dwuargumentowe (+, -, *, /, %, &, ^, <<, >>, &&, ||) są zdefiniowane dla obiektów typu `valarray` i kombinacji `valarray` z jego typem skalarnym. Typ skalarny jest traktowany jak `valarray` o odpowiednim rozmiarze, którego każdy element ma wartość tego skalara.
- Na przykład:

```
valarray<double> v1 = …;
valarray<double> v2 = …;
double d = …;
valarray<double> v3 = v*v2; // v3[i] = v[i]*v2[i] dla wszystkich i
valarray<double> v4 = v*d; // v4[i] = v[i]*d dla wszystkich i
valarray<double> v5 = d*v2; // v5[i] = d*v2[i] dla wszystkich i
valarray<double> v6 = cos(v); // v6[i] = cos(v[i]) dla wszystkich i
```

Tablica wartości numerycznych

- Na tablicach wartości numerycznych `valarray` można przeprowadzać porównania (==, !=, <, <=, >, >=).
  Przykład:
  if (v1 == v2) { … } // sprawdza czy wszystkie elementy są równe
- Na tablicach wartości numerycznych `valarray` można uruchamiać funkcje jednoargumentowe (abs, sin, cos, atan, exp, log, itp.).
  Przykład:
  v = log(va) // wykonuje log() na wszystkich elementach va
- Sprawdzanie zakresu nie jest wykonywane – skutek użycia funkcji próbującej pobrać element z pustej tablicy `valarray` jest niezdefiniowany.