Common Lisp: Typy i struktury danych

Aleksandra Woźniak

Seminarium: Rodzina języków Lisp

20 października 2010

Spis treści

Liczby

One of the reasons Lisp is a nice language for math is its numbers behave more like ***true mathematician numbers*** *than the approximations of numbers that are easy to implement in finite computer hardware.*

P. Seibel, „Practical Common Lisp”

Typy liczbowe

- liczby całkowite (`integer`)
- liczby rzeczywiste: reprezentacja zmiennopozycyjna (`float`)
- ilorazy (`ratio`)
- liczby zespolone (`complex numbers`)

Typy liczbowe

- liczby całkowite (`integer`)
- liczby rzeczywiste: reprezentacja zmiennopozycyjna (`float`)
- ilorazy (`ratio`)
- liczby zespolone (`complex numbers`)

Typy liczbowe

- liczby całkowite (`integer`)
- liczby rzeczywiste: reprezentacja zmiennopozycyjna (`float`)
- ilorazy (`ratio`)
- liczby zespolone (`complex numbers`)

Typy liczbowe

- liczby całkowite (`integer`)
- liczby rzeczywiste: reprezentacja zmiennopozycyjna (`float`)
- ilorazy (`ratio`)
- liczby zespolone (`complex numbers`)

Typy liczbowe

- liczby całkowite (`integer`)
- liczby rzeczywiste: reprezentacja zmiennopozycyjna (`float`)
- ilorazy (`ratio`)
- liczby zespolone (`complex numbers`)

Typy liczbowe

Liczby całkowite

- sposób zapisu: oczywisty
- wsparcie dla dużych liczb całkowitych

Liczby rzeczywiste:

- sposób zapisu: w formie `1.0` lub `1e0`
- mamy kilka typów reprezentacji zmiennoprzecinkowych: `single`, `float`, `long` i `double`

Typy liczbowe

Liczby całkowite

- sposób zapisu: oczywisty
- wsparcie dla dużych liczb całkowitych

Liczby rzeczywiste:

- sposób zapisu: w formie `1.0` lub `1e0`
- mamy kilka typów reprezentacji zmiennoprzecinkowych: `single`, `float`, `long` i `double`

Typy liczbowe

Liczby całkowite

- sposób zapisu: oczywisty
- wsparcie dla dużych liczb całkowitych

Liczby rzeczywiste:

- sposób zapisu: w formie `1.0` lub `1e0`
- mamy kilka typów reprezentacji zmiennoprzecinkowych: `single`, `float`, `long` i `double`

Typy liczbowe

Liczby całkowite

- sposób zapisu: oczywisty
- wsparcie dla dużych liczb całkowitych

Liczby rzeczywiste:

- sposób zapisu: w formie `1.0` lub `1e0`
- mamy kilka typów reprezentacji zmiennoprzecinkowych: `single`, `float`, `long` i `double`

Typy liczbowe

Liczby całkowite

- sposób zapisu: oczywisty
- wsparcie dla dużych liczb całkowitych

Liczby rzeczywiste:

- sposób zapisu: w formie `1.0` lub `1e0`
- mamy kilka typów reprezentacji zmiennoprzecinkowych: `single`, `float`, `long` i `double`

Typy liczbowe

Ilorazy, ułamki

- zapis: `3/4`

Liczby zespolone:

- sposób zapisu: `#c(3 1)`, `#C(1 1)`

Typy liczbowe

Ilorazy, ułamki

- zapis: `3/4`

Liczby zespolone:

- sposób zapisu: `#c(3 1)`, `#C(1 1)`

Typy liczbowe

Ilorazy, ułamki

- zapis: `3/4`

Liczby zespolone:

- sposób zapisu: `#c(3 1)`, `#C(1 1)`

Typy liczbowe

Automatyczna konwersja pomiędzy typami

Operacje na liczbach

- predykaty sprawdzające typ: `floatp`, `integerp`, `ratiop`, `complexp`
- konwersja i zaokrąglenia:
  - `float`
  - `floor`, `ceiling`
  - `round`
  - `truncate`
  - dzielenie liczb rzeczywistych: `mod`, `rem`

Operacje na liczbach

- predykaty sprawdzające typ: `floatp`, `integerp`, `ratiop`, `complexp`
- konwersja i zaokrąglenia:
  - `float`
  - `floor`, `ceiling`
  - `round`
  - `truncate`
  - dzielenie liczb rzeczywistych: `mod`, `rem`

Operacje na liczbach

- predykaty sprawdzające typ: `floatp`, `integerp`, `ratiop`, `complexp`
- konwersja i zaokrąglenia:
  - `float`
  - `floor`, `ceiling`
  - `round`
  - `truncate`
  - dzielenie liczb rzeczywistych: `mod`, `rem`

Operacje na liczbach

- predykaty sprawdzające typ: `floatp`, `integerp`, `ratiop`, `complexp`
- konwersja i zaokrąglenia:
  - `float`
  - `floor`, `ceiling`
  - `round`
  - `truncate`
  - dzielenie liczb rzeczywistych: `mod`, `rem`

Operacje na liczbach

- predykaty sprawdzające typ: `floatp`, `integerp`, `ratiop`, `complexp`
- konwersja i zaokrąglenia:
  - `float`
  - `floor`, `ceiling`
  - `round`
  - `truncate`
  - dzielenie liczb rzeczywistych: `mod`, `rem`

Porównywanie liczb

- = jako porównywanie wartości, `eql` jako porównywanie wartości i typu
- /=, <, <=, >, >=

Porównywanie liczb

- = jako porównywanie wartości, `eql` jako porównywanie wartości i typu
- /=, <, <=, >, >=

Arytmetyka i kilka innych funkcji

- standardowo: `+`, `-`, `*`,
- `min`, `max`
- `evenp`, `oddp`
- `zerop`, `minusp`, `plusp`
- `sq`, `sqrt`
- logarytmy: `log`
- funkcja wykładnicza: `exp`, `expt`
- funkcje trygonometryczne: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`

Arytmetyka i kilka innych funkcji

- standardowo: `+`, `-`, `*`,
- `min`, `max`
- `evenp`, `oddp`
- `zerop`, `minusp`, `plusp`
- `sq`, `sqrt`
- logarytmy: `log`
- funkcja wykładnicza: `exp`, `expt`
- funkcje trygonometryczne: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`

Arytmetyka i kilka innych funkcji

- standardowo: `+`, `-`, `*`,
- `min`, `max`
- `evenp`, `oddp`
- `zerop`, `minusp`, `plusp`
- `sq`, `sqrt`
- logarytmy: `log`
- funkcja wykładnicza: `exp`, `expt`
- funkcje trygonometryczne: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`

Arytmetyka i kilka innych funkcji

- standardowo: `+`, `-`, `*`,
- `min`, `max`
- `evenp`, `oddp`
- `zerop`, `minusp`, `plusp`
- `sq`, `sqrt`
- logarytmy: `log`
- funkcja wykładnicza: `exp`, `expt`
- funkcje trygonometryczne: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`

Arytmetyka i kilka innych funkcji

- standardowo: `+`, `-`, `*`,
- `min`, `max`
- `evenp`, `oddp`
- `zerop`, `minusp`, `plusp`
- `sq`, `sqrt`
- logarytmy: `log`
- funkcja wykładnicza: `exp`, `expt`
- funkcje trygonometryczne: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`

Arytmetyka i kilka innych funkcji

- standardowo: `+`, `-`, `*`,
- `min`, `max`
- `evenp`, `oddp`
- `zerop`, `minusp`, `plusp`
- `sq`, `sqrt`
- logarytmy: `log`
- funkcja wykładnicza: `exp`, `expt`
- funkcje trygonometryczne: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`

Arytmetyka i kilka innych funkcji

- standardowo: `+`, `-`, `*`,
- `min`, `max`
- `evenp`, `oddp`
- `zerop`, `minusp`, `plusp`
- `sq`, `sqrt`
- logarytmy: `log`
- funkcja wykładnicza: `exp`, `expt`
- funkcje trygonometryczne: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`

Arytmetyka i kilka innych funkcji

- standardowo: `+`, `-`, `*`,
- `min`, `max`
- `evenp`, `oddp`
- `zerop`, `minusp`, `plusp`
- `sq`, `sqrt`
- logarytmy: `log`
- funkcja wykładnicza: `exp`, `expt`
- funkcje trygonometryczne: `sin`, `cos`, `tan`, `asin`, `acos`, `atan`

Przykład

Pierwiastki równania kwadratowego i wartości zespolone

Listy

Program jest listą.

Listy

Notacja: `'(1 2 3 4 5)` – uwaga na **apostrof**!

Listy

Notacja: `'(1 2 3 4 5)` – uwaga na **apostrof**!

Operacje na listach

- `cons`
- `car, cdr`
- `member`

Cons

Czym właściwie jest cons?

- cons jest obiektem
- intuicyjnie, składa się z pary wskaźników: `car`, `cdr`
- zatem: listy nie są osobnym typem danych – są raczej zbiorem połączonych ze sobą `cons`ów

Cons

Czym właściwie jest cons?

- cons jest obiektem
- intuicyjnie, składa się z pary wskaźników: `car`, `cdr`
- zatem: listy nie są osobnym typem danych – są raczej zbiorem połączonych ze sobą `cons`ów

Cons

Czym właściwie jest cons?

- cons jest obiektem
- intuicyjnie, składa się z pary wskaźników: `car`, `cdr`
- zatem: listy nie są osobnym typem danych – są raczej zbiorem połączonych ze sobą `cons`ów

Cons

Czym właściwie jest cons?

- cons jest obiektem
- intuicyjnie, składa się z pary wskaźników: `car`, `cdr`
- zatem: listy nie są osobnym typem danych – są raczej zbiorem połączonych ze sobą `cons`ów

Porównywanie list

`eql` **vs.** `equal`

Operacje na listach

- kopiowanie: `copy-list`
- konkatenacja: `append`

Przykład

Prosta kompresja i dekompresja list.

Pobieranie elementów z listy

- `nth` – dostęp do n-tego elementu listy
- Uwaga! Listy są indeksowane od 0.
- `last`
- Koszmarna ciekawostka: `(caddr x)` <=> `(car (cdr (cdr x)))`

Pobieranie elementów z listy

- `nth` – dostęp do n-tego elementu listy
- Uwaga! Listy są indeksowane od 0.
- `last`
- Koszmarna ciekawostka: `(caddr x)` <=> `(car (cdr (cdr x)))`

Pobieranie elementów z listy

- `nth` – dostęp do n-tego elementu listy
- Uwaga! Listy są indeksowane od 0.
- `last`
- Koszmarna ciekawostka: `(caddr x)` <=> `(car (cdr (cdr x)))`

Pobieranie elementów z listy

- `nth` – dostęp do n-tego elementu listy
- Uwaga! Listy są indeksowane od 0.
- `last`
- Koszmarna ciekawostka: `(caddr x)` <=> `(car (cdr (cdr x)))`

Funkcje mapujące

- `mapcar`: mapowanie „po jednym elemencie”
- `maplist`: mapowanie „po ogonach”

Notacja kropkowa

- `cons` pozwala także na tworzenie „list niewłaściwych”
- przykład: `(cons 'a 'b)` jest raczej strukturą dwuelementową niż listą

Notacja kropkowa

- `cons` pozwala także na tworzenie „list niewłaściwych”
- przykład: `(cons 'a 'b)` jest raczej strukturą dwuelementową niż listą

Listy asocjacyjne: przykład użycia notacji kropkowej

- tworzenie: jako lista „par” - Uwaga! Notacja kropkowa!
- pobieranie elementów: `assoc`

Listy asocjacyjne: przykład użycia notacji kropkowej

- tworzenie: jako lista „par” - Uwaga! Notacja kropkowa!
- pobieranie elementów: `assoc`

Ćwiczenia

1. Generowanie wszystkich ogonów listy, generowanie jednoelementowanych podlist listy.
2. Funkcja `my-member`.

Tablice

- tworzenie: `(make-array '(2 3) :initial-element nil)`
- Siedem wymiarów tablicy powinno wystarczyć każdemu.
- pobieranie elementów tablicy: `aref`

Tablice

- tworzenie: `(make-array '(2 3) :initial-element nil)`
- Siedem wymiarów tablicy powinno wystarczyć każdemu.
- pobieranie elementów tablicy: `aref`

Tablice

- tworzenie: `(make-array '(2 3) :initial-element nil)`
- Siedem wymiarów tablicy powinno wystarczyć każdemu.
- pobieranie elementów tablicy: `aref`

Wektory

Wektor to po prostu jednoelementowa tablica.

- tworzenie: `(make-array 5)` albo `(vector 1 2 3)`
- pobieranie elementów: `aref`, albo (szybciej!) `svref`

Wektory

Wektor to po prostu jednoelementowa tablica.

- tworzenie: `(make-array 5)` albo `(vector 1 2 3)`
- pobieranie elementów: `aref`, albo (szybciej!) `svref`

Znaki i napisy – 1

- napis = wektor znaków
- reprezentacja:
  - `"to jest napis"`
  - to jest znak: `#\c`
- porównywanie znaków: `char<`, `char<=`, `char=`, `char>=`, `char>`, `char/=`
- przykład: sortowanie napisów

Znaki i napisy – 1

- napis = wektor znaków
- reprezentacja:
  - `"to jest napis"`
  - to jest znak: `#\c`
- porównywanie znaków: `char<`, `char<=`, `char=`, `char>=`, `char>`, `char/=`
- przykład: sortowanie napisów

Znaki i napisy – 1

- napis = wektor znaków
- reprezentacja:
  - `"to jest napis"`
  - to jest znak: `#\c`
- porównywanie znaków: `char<`, `char<=`, `char=`, `char>=`, `char>`, `char/=`
- przykład: sortowanie napisów

Znaki i napisy – 1

- napis = wektor znaków
- reprezentacja:
  - `"to jest napis"`
  - to jest znak: `#\c`
- porównywanie znaków: `char<`, `char<=`, `char=`, `char>=`, `char>`, `char/=`
- przykład: sortowanie napisów

Znaki i napisy – 2

- pobieranie elementu napisu: `aref` lub `char`
- równość napisów:
  - `equal`
  - `string-equal` – Uwaga: case-insensitive!
- funkcja `format`
- konkatenacja: `concatenate`

Znaki i napisy – 2

- pobieranie elementu napisu: `aref` lub `char`
- równość napisów:
  - `equal`
  - `string-equal` – Uwaga: case-insensitive!
- funkcja `format`
- konkatenacja: `concatenate`

Znaki i napisy – 2

- pobieranie elementu napisu: `aref` lub `char`
- równość napisów:
  - `equal`
  - `string-equal` – Uwaga: case-insensitive!
- funkcja `format`
- konkatenacja: `concatenate`

Znaki i napisy – 2

- pobieranie elementu napisu: `aref` lub `char`
- równość napisów:
  - `equal`
  - `string-equal` – Uwaga: case-insensitive!
- funkcja `format`
- konkatenacja: `concatenate`

Sekwencje

Listy i wektory mają ze sobą wiele wspólnego. . .

Operacje na sekwencjach

- `length`
- `subseq`
- `reverse`
- `sort`
- `every, some`
- `position`

Operacje na sekwencjach

- `length`
- `subseq`
- `reverse`
- `sort`
- `every, some`
- `position`

Operacje na sekwencjach

- `length`
- `subseq`
- `reverse`
- `sort`
- `every, some`
- `position`

Operacje na sekwencjach

- `length`
- `subseq`
- `reverse`
- `sort`
- `every, some`
- `position`

Operacje na sekwencjach

- `length`
- `subseq`
- `reverse`
- `sort`
- `every`, `some`
- `position`

Operacje na sekwencjach

- `length`
- `subseq`
- `reverse`
- `sort`
- `every`, `some`
- `position`

Inne operacje na sekwencjach

- `find` – podobna do `member`
- `remove`, `remove-duplicates`
- `:from-end`
- `:start` – numer pierwszego rozpatrywanego elementu
- `:end` – numer pierwszego NIE rozpatrywanego elementu

Inne operacje na sekwencjach

- `find` – podobna do `member`
- `remove`, `remove-duplicates`
- `:from-end`
- `:start` – numer pierwszego rozpatrywanego elementu
- `:end` – numer pierwszego NIE rozpatrywanego elementu

Inne operacje na sekwencjach

- `find` – podobna do `member`
- `remove`, `remove-duplicates`
- `:from-end`
- `:start` – numer pierwszego rozpatrywanego elementu
- `:end` – numer pierwszego NIE rozpatrywanego elementu

Inne operacje na sekwencjach

- `find` – podobna do `member`
- `remove`, `remove-duplicates`
- `:from-end`
- `:start` – numer pierwszego rozpatrywanego elementu
- `:end` – numer pierwszego NIE rozpatrywanego elementu

Inne operacje na sekwencjach

- `find` – podobna do `member`
- `remove`, `remove-duplicates`
- `:from-end`
- `:start` – numer pierwszego rozpatrywanego elementu
- `:end` – numer pierwszego NIE rozpatrywanego elementu

Struktury

A structure can be considered as a deluxe kind of vector.

P. Graham, „ANSI Common Lisp”

Struktury: podstawy

- tworzenie struktury: `defstruct`
- Uwaga! Po stworzeniu struktury dostajemy kilka funkcji w prezencie.
- `make-sth` – konstruktor
- funkcje pobierające element struktury
- predykat sprawdzający, czy podany argument jest typu tej struktury
- funkcja kopiująca

Struktury: podstawy

- tworzenie struktury: `defstruct`
- Uwaga! Po stworzeniu struktury dostajemy kilka funkcji w prezencie.
- `make-sth` – konstruktor
- funkcje pobierające element struktury
- predykat sprawdzający, czy podany argument jest typu tej struktury
- funkcja kopiująca

Struktury: podstawy

- tworzenie struktury: `defstruct`
- Uwaga! Po stworzeniu struktury dostajemy kilka funkcji w prezencie.
- `make-sth` – konstruktor
- funkcje pobierające element struktury
- predykat sprawdzający, czy podany argument jest typu tej struktury
- funkcja kopiująca

Struktury: podstawy

- tworzenie struktury: `defstruct`
- Uwaga! Po stworzeniu struktury dostajemy kilka funkcji w prezencie.
- `make-sth` – konstruktor
- funkcje pobierające element struktury
- predykat sprawdzający, czy podany argument jest typu tej struktury
- funkcja kopiująca

Struktury: podstawy

- tworzenie struktury: `defstruct`
- Uwaga! Po stworzeniu struktury dostajemy kilka funkcji w prezencie.
- `make-sth` – konstruktor
- funkcje pobierające element struktury
- predykat sprawdzający, czy podany argument jest typu tej struktury
- funkcja kopiująca

Struktury: podstawy

- tworzenie struktury: `defstruct`
- Uwaga! Po stworzeniu struktury dostajemy kilka funkcji w prezencie.
- `make-sth` – konstruktor
- funkcje pobierające element struktury
- predykat sprawdzający, czy podany argument jest typu tej struktury
- funkcja kopiująca

Struktury: przykład

Punkty w przestrzeni trójwymiarowej.

Tablice hashujące

- tworzenie: `make-hash-table`
- pobieranie elementu skojarzonego z danym kluczem: `gethash`
- Uwaga: `gethash` zwraca dwa elementy
- usuwanie elementu: `remhash`
- funkcja mapująca: `maphash`

Tablice hashujące

- tworzenie: `make-hash-table`
- pobieranie elementu skojarzonego z danym kluczem: `gethash`
- Uwaga: `gethash` zwraca dwa elementy
- usuwanie elementu: `remhash`
- funkcja mapująca: `maphash`

Tablice hashujące

- tworzenie: `make-hash-table`
- pobieranie elementu skojarzonego z danym kluczem: `gethash`
- Uwaga: `gethash` zwraca dwa elementy
- usuwanie elementu: `remhash`
- funkcja mapująca: `maphash`

Tablice hashujące

- tworzenie: `make-hash-table`
- pobieranie elementu skojarzonego z danym kluczem: `gethash`
- Uwaga: `gethash` zwraca dwa elementy
- usuwanie elementu: `remhash`
- funkcja mapująca: `maphash`

Tablice hashujące

- tworzenie: `make-hash-table`
- pobieranie elementu skojarzonego z danym kluczem: `gethash`
- Uwaga: `gethash` zwraca dwa elementy
- usuwanie elementu: `remhash`
- funkcja mapująca: `maphash`

Symbole

Symbols [are] variable names existing as objects in their own right.

P. Graham, „ANSI Common Lisp”

Symbol: więcej, niż tylko nazwa

- nazwa
- pakiet
- wartość
- funkcja
- *plist*, *property list*

Symbol: więcej, niż tylko nazwa

- nazwa
- pakiet
- wartość
- funkcja
- *plist, property list*

Symbol: więcej, niż tylko nazwa

- nazwa
- pakiet
- wartość
- funkcja
- *plist, property list*

Symbol: więcej, niż tylko nazwa

- nazwa
- pakiet
- wartość
- funkcja
- *plist, property list*

Symbol: więcej, niż tylko nazwa

- nazwa
- pakiet
- wartość
- funkcja
- *plist*, *property list*

Podsumowanie

Bibliografia

- P. Graham, „ANSI Common Lisp”
- P. Seibel, „Practical Common Lisp”