PARADYGMATY I JĘZYKI PROGRAMOWANIA

27.03.2014

Zmienne. Typy danych. cd.

Treść

- Zmienne
  - nazwa, zakresy,...
- Typy danych
  - wstęp
    - definicje
    - klasyfikacja
  - Sprawdzanie typów
    - równoważność
    - wnioskowanie typów
  - Tablice
  - Rekordy, warianty
  - Napisy
  - Wskaźniki i typy rekurencyjne

Tablice, rekordy itp.

Tablice

- Jednorodne zbiory danych, indeksowane
  - `tablica(index)|tablica[index]|tablica[ind1][ind2],...`
- indeksy: zależnie od języka; generalnie typ wyliczeniowy; Ada: boole'owski, znakowy, wyliczeniowy
- położenie elementu względem elementu pierwszego jest obliczane w czasie wykonania (koszt)
- Rodzaje tablic (w zależności od wiązania zakresów indeksów, wiązania pamięci oraz miejsca alokacji)
  - statyczne (statycznie ustalone zakresy indeksów i statyczna alokacja)
  - ustalone, dynamiczne, na stosie (zakres indeksów ustalony, alokacja w czasie wykonania)
  - dynamiczne, na stosie (zakres indeksów i alokacja ustalane są w czasie wykonania)
  - ustalone, dynamiczne, na stercie
  - dynamiczne, na stercie (możliwa zmiana zakresów indeksów)

Przykład: Tablice

- Operacje na tablicach w Fortranie 95
  - dodawanie, odejmowanie (a±b), całe tablice!
  - transpozycja (transpose)
  - mnożenie tablic (multiply(a,b))
  - krojenie tablic
  - wyznacznik (det(a))
  - odwracanie (1/a, inverse)
- Python (numeracja od 0)
  - wektor = [3, 4, 1, 5, 6];
  - tablica = [[1,2],3,[5,4,7],[1,2,3],56];
- Perl (numeracja od zera)
  - @lista[1..5]=@lista2[3,4,5,6,7];
- Ruby (numeracja od zera)
  - list=[2, 4, 5, 6, 7]; list.slice(2,2) zwraca [5,6]

Adresy elementów

- Tablice 1 wymiarowe;
- Tablice 2 wymiarowe;
- Tablice wielowymiarowe;

- REALIZACJA w pamięci:
  - C, C++: wierszowa
  - Fortran: kolumnowa
  - Organizacja efektywnych pętli **`for()/do`**

- Napisać formuły dla adresów elementów tablicy dwuwymiarowej A[K,L] w Fortranie i C++.

  Np. dla tablicy 1-o wymiarowej lista[]:

  *adres(lista[k]) = adres(lista[indeks_dolny]) + ((k-indeks_dolny)*rozmiar_elementu;*

Tablice a pamięć

- pamięć ciągła
- wskaźniki do wierszy
  - opcja w C
  - wiersze mogą być składane w dowolnym miejscu pamięci – wygodne dla dużych tablic
  - wygodne dla tablic o różnej długości wierszy
  - wymagana dodatkowa pamięć dla wskaźników

Row-pointers

p	o	n	i	e	d	z	i	a	l	e	k
w	t	o	r	e	k						
s	r	o	d	a							
c	z	w	a	r	t	e	k				
p	i	a	t	e	k						
s	o	b	o	t	a						
n	i	e	d	z	i	e	l	a			


```
char tydzien[10] ={„poniedzialek”,
„wtorek”, „sroda”, „czwartek”,
„piatek”, „sobota”};
```

```
char *tydzien[10] ={„poniedzialek”,
„wtorek”, „sroda”, „czwartek”,
„piatek”, „sobota”};
```

Tablice asocjacyjne

- nieuporządkowane kolekcje danych, indeksowane za pomocą kluczy; pary (klucz, wartość);
  Przykład, Perl:

```
%x=("ja" => 200, "ty"=>234, "my"=>10);
$("oni")=765;
```

- Wygodne w wielu sytuacjach
- Często nazywane haszami (ang. hash)
- Najczęściej w językach interpretowanych

Rekordy

- rekord = najczęściej niejednorodny zbiór danych; poszczególne elementy identyfikowane są nazwą
- C, C++, C#: **`struct`**
- Przykład. Ada.

```
type Dane_osobowe is record
    Imie: String (1..20);
    Imie_drugie: String
    Nazwisko: String (1..10);
end record;
type Dane_pracownika is record
    Pracownik: Dane_osobowe;
    stawka_godzinowa: float;
end record;
Zatrudniony: Dane_pracownika;
```

Rekordy

- operacje na rekordach
  - Przykład. COBOL:

  Rekordy **`INPUT-RECORD i OUTPUT-RECORD`**;
  możliwe polecenie:

  `MOVE CORRESPONDING INPUT-RECORD TO OUTPUT-RECORD`
- adres pola w rekordzie liczony jest względem adresu rekordu
- Kiedy rekordy są tego samego typu?

Rekordy

- Pascal

```
type dwa_znaki = packed_
array [1..2];
type element = record
   symbol : dwa_znaki;
   liczba_atomowa :
integer;
   masa_atomowa : real;
   metal : Boolean
end;
```

- C (struktura)

```
struct element {
   char symbol[2];
   int liczba_atomowa;
   double masa_atomowa;
_Bool metal;
};
```

- Dostęp
  - Pascal

```
var srebro : element;
…
srebro.symbol: = 'Ag';
l_atomow = masa /
   srebro.masa_atomowa;
```

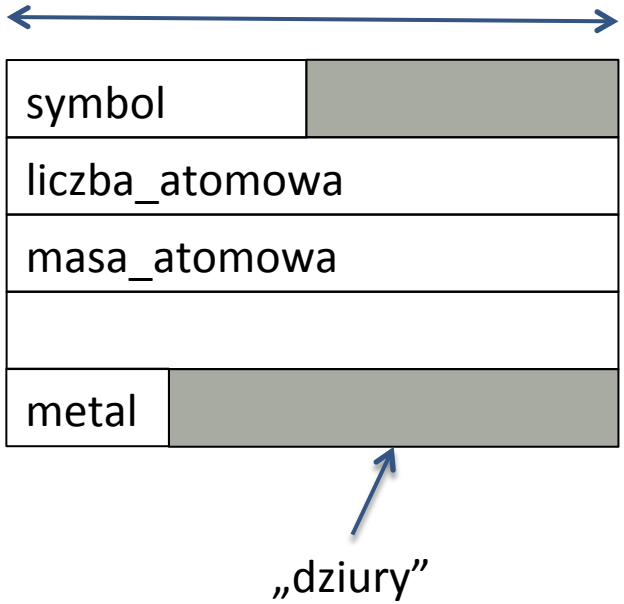
```
(+ Rekordy )
```

  - C

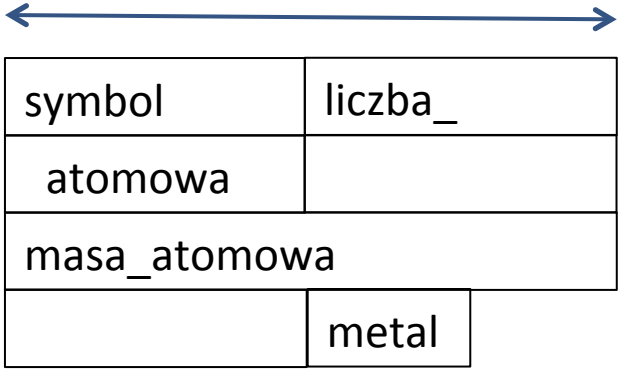
```
element srebro;
  …
srebro.symbol = 'Ag';
l_atomow = masa /
  srebro.masa_atomowa;
```

Rekordy, pamięć

- Pascal., r. zwykły
  - 4 bajty = 32 bity



- Pascal, r. ‘**packed**’



- ew. reorganizacja w celu zminimalizowania zajętości pamieci

Rekordy wariantowe

- Rekordy wariantowe wywodzą się z Algolu 68
- Fortran

```
integer i
real r
logical b
equivalence (i, r, b)
```

pola `i, r, b` nie będą używane w tym samym czasie

- Rekord wariantowy jest podobny do unii w języku C
- W Pascalu zintegrowano unie i rekordy

Rekordy wariantowe

- Pascal

```
type dwa_znaki= packed_
array [1..2];
long_string : packed array
[1..200] of char;
string_ptr : ^long_string;
type element = record
   symbol: dwa_znaki;
   liczba_atomowa :
integer;
   masa_atomowa: real;
   metal: Boolean;
case w_naturze: Boolean of
   true: (
     miejsce:string_ptr;
     abundancja`; real;
   );
   false : (
      czas_zycia : real;
   )
end;
```

- C

```
struct element {
   char symbol[2];
   int liczba_atomowa;
   double masa_atomowa;
   _Bool metal;
   _Bool w_naturze;
   union {
      struct {
         char *miejsce;
         double abundancja;
      } info;
      double czas_zycia;
    } pola_dodane;
} srebro;
```

- Ponieważ `unia` nie jest częścią `struct` wprowadzono dwa dotatkowe poziomy nazw. Pole trzecie to wciąż `srebro.masa_atomowa`, ale dostęp do pola `miejsce` odbywa się przez `srebro.pola_dodane.info.miejsce`

Rekordy a unie

- Algol 68

```
union (int, real, bool) uirb
# uirb może być typu int, real lub boolean
```

…

```
uirb := 1      # integer
```

…

```
uirb := 12.34  # real
```

W celu pobrania elementu należy przygotowac specjalną instrukcję **`case`**:

```
case uirb is
   (int i)    : print(i)
   (real r)   : print(r)
   (bool b)   : print(b)
esac
```

Wielkości w nawiasach, w instrukcji **`case`**, są nazwami wartości zawartych w unii. Ukryty znacznik (typu) zmieniany jest niejawnie; dotyczy unii.

Rekordy a unie

- Znaczone rekordy w Pascal-u

```
type tag = (is_int, is_real, is_bool)
var uirb : record
   case which : tag of
      is_int  : (i : integer);
      is_real : (r : real);
      is_bool : (b : Boolean)
end;
```

- Problemy. Znacznik w Pascal-u może być zmieniony jawnie zwykłą instrukcją przypisania (nie jak w Algolu)

```
uirb.which:=is_real;
uirb.r:=3.00;
uirb.which:=is_int;
…
write(uirb.i);   (* błąd*)
```

Rekordy a unie

- Jest jeszcze gorzej … bo pole znacznika (tag) rekordu jest opcjonalne

```
var uirb : record
   case tag of
      is_int  : (i : integer);
      is_real : (r : real);
      is_bool : (b : Boolean)
end;
…
uirb.r = 3.0;
…
writeln(uirb.i);    (* źle! *)
```

Krotki

- Krotka jest podobna do rekordu, ale dane zawarte w krotce nie są nazwane tak jak pola rekordu; sa indeksowane
- Python – niezmienialne (immutable) krotki

```
krotka = (3, 6, 'kot', 56 );
x = krotka[3];  // 56
```

łączenie: `+`

- ML

```
val krotka = (3, 6, 'banan', 56 );
```

dostęp do elementu pierwszego: `#1(krotka);`

- F#

```
let krotka = (7, 5, 1);
let a, b, c = krotka; // a jest 7, c jest 1
```

Listy

- LISP – pierwszy
  `(A B C D), (A (B C) D)` - listy
  operacje podstawowe:
  `(CAR '(A B C D))` zwraca A
  `(CDR '(A B C D))` zwraca (B C D)
  łączenie:
  `(CONS 'A '(A B C))` zwraca `(A A B C)`
- Podobnie: Scheme, ML;
- Python, Haskell: przetwarzanie list
  np. (kwadraty liczb nieparzystych):

```
[x*x for x in range(10) if x%2 == 1]
```

Unie

- C, C++, Ada, F#
- C, C++ - nie jest wspomagane sprawdzania typów
- Ada: unie wariantowe (wcześniej: Algol 68)
- F#

```
type intReal =
      | IntVal of int
      | RealVal of float;
```

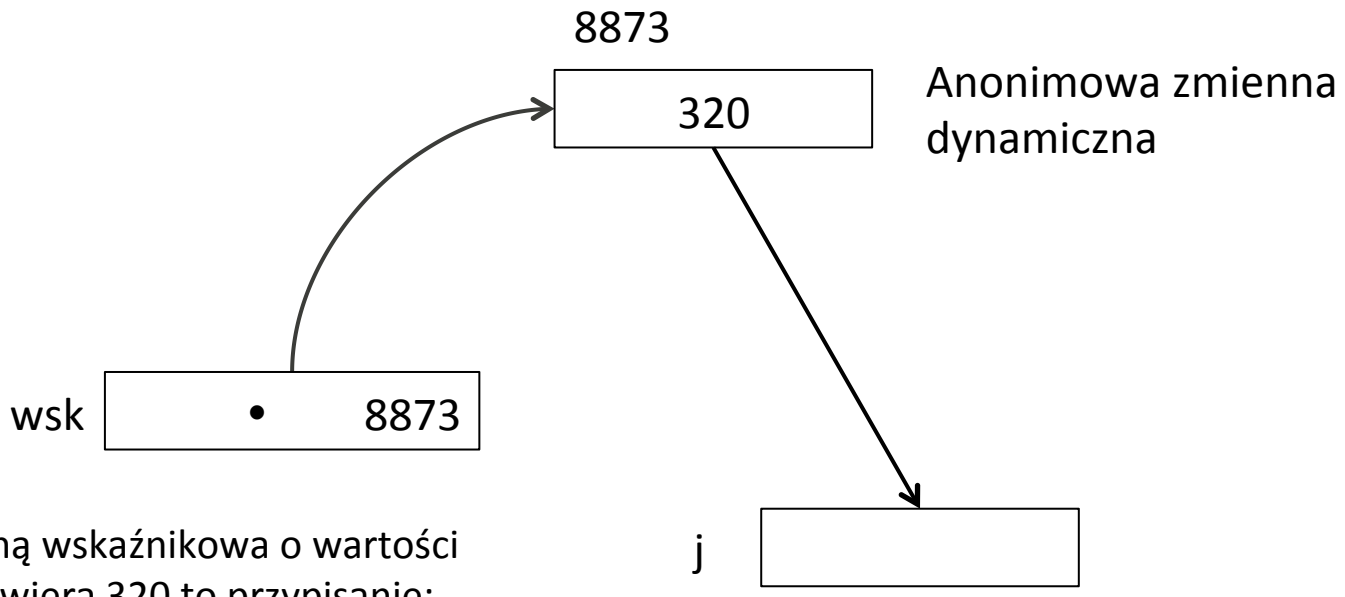
Tutaj **intReal** jest typem unii, a **IntVal** i **RealVal** konstruktorami; piszemy np.

```
let ira = IntVal 20;
let irb = RealVal 3.1415;
```

Wskaźniki

- Zmienne typu wskaźnikowego (pointer) mają zakres wartości adresów pamięci plus **`nil`**. Służą do pośredniego adresowania (programowanie w asemblerze) oraz pozwalają zarządzać pamięcią dynamiczną (sterta, heap).
- Wiele języków (Fortran 77, Java; różne powody) nie używa wskaźników;
- dereferencja, wyłuskanie = pobranie wartości z obszaru wskazywanego
- Wygodne przy budowie strukturalnych typów danych (listy wiązane, drzewa, grafy,...)

Wskaźniki – wyłuskanie wartości



Jeśli **`wsk`** jest zmienną wskaźnikowa o wartości 8873 i adres 8873 zawiera 320 to przypisanie:

```
j = *wsk
```

wstawi do `j` 320.

Wskaźniki

- C, C++:
  - jeśli zmienna wskaźnikowa p wskazuje na rekord, a polem rekordu jest np. **`imie to`** wartość pola można pobrać przez **`(*p).imie`** lub stosując operator **`->`**:
    `p->imie;`
  - operacje **`new, delete`**
  - arytmetyka (**`wsk`** – wskaźnik, **`indeks`** – integer):
    `wsk + indeks; *(wsk+1); *(wsk+indeks);`
    `wsk[indeks];`
- Problem wiszących wskaźników; wycieki pamięci; zgubione zmienne na stercie

Wskaźniki

- Model referencyjny (przez referencję)
  - A:=B oznacza, że A wskazuje na to samo na co wskazuje B
  - języki funkcyjne (ML, Clu, Smalltalk)
- Model wartości (przez wartość)
  - przypisanie A := B oznacza przepisanie wartości *ze zmiennej* A *do zmiennej* B
  - C, Pascal, Ada
- Mieszane modele
  - Java: jeśli A, B są zmiennymi typu wbudowanego – przez wartość; dla typów zdefiniowanych przez użytkownika – przez referencję.

Wskaźniki i struktury

- ML (referencyjny)

```
datatype chrtree = empty | node of char * chrtree *
chr_tree
```

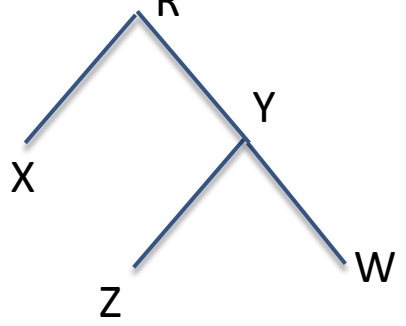
- Lisp

```
`(#\R (#\X ()()) (#\Y (#\Z ()()) (#\W ()())))
```

model referencyjny, bez statycznego typowania

- Pascal ()

```
type chr_tree_ptr = ^chr_tree
   chr_tree = record
      left, right : chr_tree_ptr;
      val : char
   end;
```



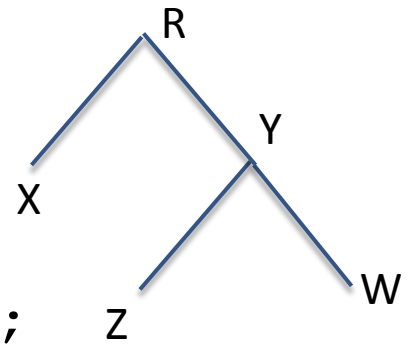
Wskaźniki i struktury

- Ada

```
type chr_tree;
type chr_tree_ptr is access chr_tree;
type chr_tree is record
   left, right : chr_tree_ptr;
   val : character;
end record;
```

- C

```
struct chr_tree {
   struct chr_tree *left, *right;
   char val;
};
```



Wskaźniki i struktury

- Rezerwacja pamięci:
  - Pascal
    ```
    new(my_ptr);
    ```
  - Ada
    ```
    my_ptr := new chr_tree;
    ```
  - C
    ```
    my_ptr = (struct chr_tree *)
             malloc(sizeof(struct chr_tree))
    ```
  - funkcja `malloc` nie jest wbudowana (biblioteka)
  - Java, C++, C#
    ```
    my_ptr = new chr_tree(lista_arg);
    ```

Wskaźniki

- Dostęp
  - Pascal

    `my_ptr^.val := 'X';`
  - C

    `(*my_ptr).val ='X';` lub **`my_ptr->val='X';`**
  - Ada

    `T : chr_tree; P: chr_tree; ...`
    `T.val := 'X'; P.val := 'Y';`
  - Lisp. Dostęp przez wbudowane funkcje

    `cons, car, cdr`

Zarządzanie stertą

- sprzątanie
  - równe obszary (pojedyncze komórki)
    - licznik referencji
    - garbage collection
      - zaznaczenie wszystkich obszarów jako nieużytków;
      - sprawdzenie wskaźników programu i odznaczenie używanych obszarów
      - zwolnienie wciąż zaznaczonych nieużytków
  - różne obszary – więcej problemów

Referencje

- W odróżnieniu od wskaźnika, który odnosi się do adresu pamięci, referencja wskazuje obiekt lub wartość w pamięci (nie można wykonywać działań arytmetycznych na referencjach); obiekty wskazywane referencyjnie są niejawnie wyłuskiwane.
- C++
  - referencja jest stałym wskaźnikiem; musi być inicjowana adresem zmiennej w chwili jej definiowania i nie może być zmieniana
  - niejawne dereferencje w ‘funkcjach’ pozwalają na tworzenie bardziej przejrzystych kodów (wskaźniki wymagają jawnych dereferencji co zmniejsza czytelność programów)

Zbiory

- Nieuporządkowane kolekcje dowolnej liczby wartości wspólnego typu (Pascal, operacje: suma, przecięcie, różnica)
- Realizacja: wektor bitów o długości równej liczbie różnych wartości typu bazowego. Np. dla znaków ASCII 128 bitów = 16 bajtów (0 oznacza przynależność elementu do zbioru, 1 – brak w zbiorze) . Model nie pracuje dla typów bazowych dużej mocy. (Jak długi musi być wektor bitów dla typu 32-integer, 64-integer?)
- Lepsza realizacja: tablice assocjacyjne (hash)

Literatura

- Sebesta, Rozdział 6
- Scott, Rozdzial 7

za tydzień ... !?