Nazwy

I tak mężczyzna dał nazwy wszelkiemu bydłu,
ptakom powietrznym i wszelkiemu zwierzęciu polnemu,
ale nie znalazła się pomoc odpowiednia dla mężczyzny.

Księga Rodzaju 2, 20

Wnioski:

- sprzęt tworzy Elektronik, ale nazwy nadaje programista,
- i nie powinien liczyć, że nadanie nazwy pomoże mu rozwiązać problem.

Nazwy w programach oznaczają:

- stałe,
- zmienne,
- parametry,
- funkcje,
- typy,
- obiekty,
- klasy,
- makra,
- itd.

Nazwy

Nazwa — niepusty ciąg liter i cyfr zaczynający się od litery
(w większości języków)

Przykład:

`x` `indeks` `ind2`

`sumaParzDod` — t.zw. „notacja wielbłądzia"

`suma_parz_dod` — w niektórych językach podkreślnik zalicza się do liter

„Chameau de bactriane"
autorstwa Bouette.

W niektórych językach nie ma rozróżnienia między dużymi i małymi literami w nazwie.

W niektórych językach istnieją ograniczenia na długość nazwy; albo końcowe litery są ignorowane (np. w C99: w nazwie zadeklarowanej poza funkcjami liczy się tylko 31 pierwszych liter/cyfr).

Zmienne

Zmienna posiada:

- nazwę,
- adres,
- wartość,
- typ,
- czas życia,
- zasięg...

} **atrybuty zmiennych**

Nazwa zmiennej

Nazwa zmiennej — zwykle dowolna

ale w niektórych językach musi się zaczynać od czegoś ustalonego;

np. w PHP — od `$...`

w Perlu — od `$...` zmienna skalarna
od `@...` zmienna tablicowa
od `%...` tablica asocjacyjna

Adres zmiennej

zmienna ← abstrakcja — **miejsce w pamięci**

Zmiennej jest przypisany adres w pamięci.

Ale czasem tej samej zmiennej różne adresy:

```
void  silnia(int* sil, int n) {
 int  pom;
 if (n == 0)  *sil = 1;
 else {
  silnia (&pom, n-1);
  *sil = pom*n;
 }
}
```

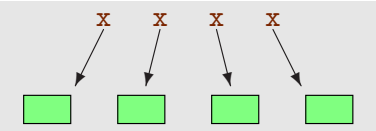
Funkcja rekursywna:

na każdym poziomie powstaje egzemplarz zmiennej lokalnej,

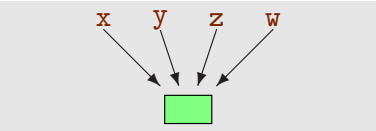
każdy egzemplarz ma inny adres

Adres zmiennej

Jednej zmiennej **może** być przypisane wiele adresów.



Jeden adres **może** być przypisany wielu zmiennym (`aliasy`).



Aliasy są niebezpieczne. Przykładowe konstrukcje tworzące aliasy:

```
.....................
double x;  double* y;
y = &x;
.....................
```

```
int k;
.................
void  qq(int n) {
 k = k+n;
}
.................
qq(k);
```

Teraz `x` i `*y` to ta sama zmienna rzeczywista.

W ciele funkcji nazwy `k` i `n` oznaczają tą samą zmienną całkowitą.

Typ zmiennej

Typ zmiennej to

- zbiór wartości, które ta zmienna może przyjmować, oraz
- zbiór operacji, które na nich można wykonać.

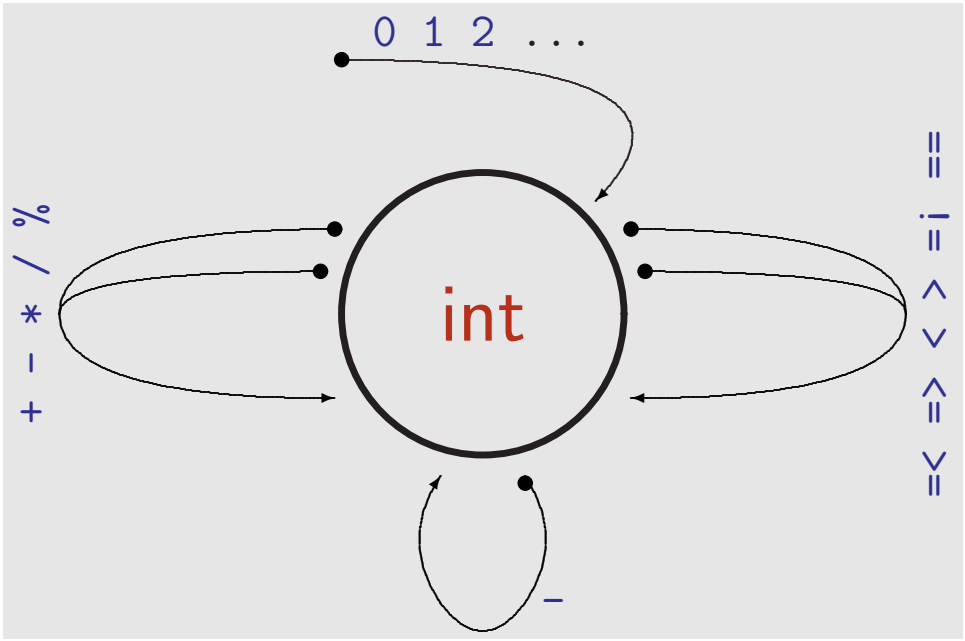
Przykład:
W C: typy `int` i `unsigned int`

- te same zbiory układów bitów, ale
- różnie działające operacje:

w `int`: $10 - 20$ jest liczbą ujemną
w `unsigned int`: $10 - 20$ jest (dużą) liczbą dodatnią

— dlatego to są **różne typy**.

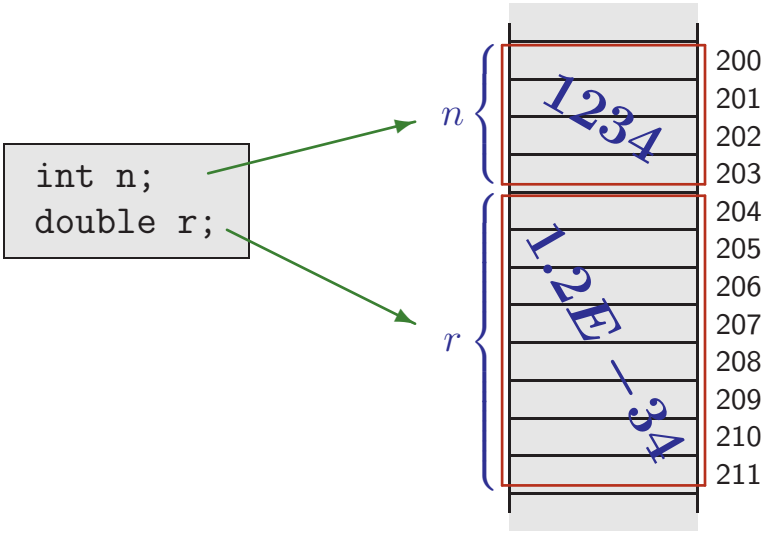
Typ `int` w C



Wartość zmiennej

Wartość zmiennej to zawartość „komórki pamięci" przypisanej tej zmiennej.

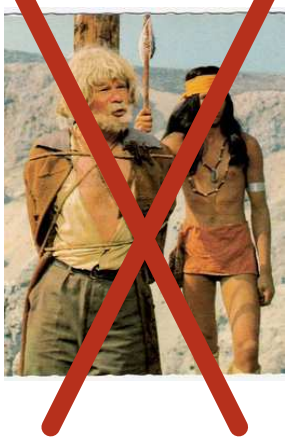
„Komórkę" rozumiemy **abstrakcyjnie**, może się na nią składać kilka bajtów.



O wiązaniu (*binding*)

Wiązanie —

NIE



TAK

związek między elementami programu a hardware'em albo **powiązanie** jakiegoś „bytu" z atrybutem

O wiązaniu (*binding*)

Wiązanie — przypisanie atrybutów, połączenie symbolu ze znaczeniem

Wiązanie **statyczne** —
wykonywane zanim program zacznie działać (w czasie kompilacji)
i już niezmienne w trakcie działania;
np. definicja stałej w C:

```
#define max_wlk_tablicy 1000
```

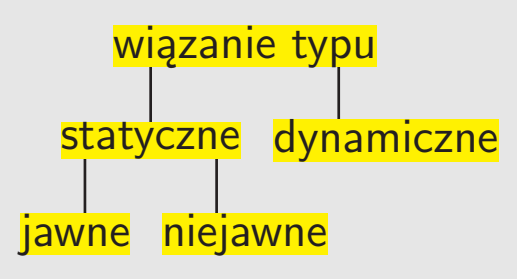
nadaje stałej `max_wlk_tablicy` **wartość statycznie**.

Wiązanie **dynamiczne** —
wykonywane w czasie działania;
np. komenda stworzenia nowego obiektu w Jawie:

Punkt poczatek = new Punkt(0.0, 0.0);

nadaje obiektowi `poczatek` **wartość dynamicznie**.

Wiązanie typu



Przykład:

Jawna statyczna deklaracja typu zmiennej:

```
int  n,i,j;                          — w C
var  tab : array [1..10] of real;    — w Pascalu
```

Niejawna statyczna deklaracja typu zmiennej:

`JAN ...` — w Fortranie (zmienne na `I,J,K,L,M,N` są całkowite, pozostałe są rzeczywiste)

`$tab ...` — w Perlu (zmienne na `$` są skalarne, zmienne na `@` są tablicowe, zmienne na `%` są tablicami asocjacyjnymi)

Wiązanie typu

Niejawna statyczna deklaracja typu zmiennej:

- zaoszczędza pisania deklaracji 
- utrudnia kompilatorowi wykrycie błędu 

błędnie napisana zmienna deklaruje się sama, bez wiedzy programisty

Wiązanie typu

Przykład: program w Fortranie —

```
program suma
sum = 0
n = 0
do
 n = n+1
 sum = sum + n
 if (n == 10) then
  exit
 end if
end do
print*,"Suma ==", sum
end program suma
```

drukuje: `Suma == 55.00000000`

```
program suma
sum = 0
n = 0
do
 n = n+1
 sum = sum + h
 if (n == 10) then
  exit
 end if
end do
print*,"Suma ==", sum
end program suma
```

drukuje: `Suma == 0.00000000`

Wiązanie typu

Dynamiczne powiązanie zmiennej i typu:

- nie ma ani deklaracji, ani umowy o związku nazw z typami; kompilator nie może ustalić typu — będzie to robił dopiero program w trakcie działania;
- typ zmiennej ustala pierwsze przypisanie wartości:

```
x = 'napis';
x = 3.14;
```

zmienna `x`
najpierw staje się napisowa,
potem rzeczywista

Tak jest np. w Javascripcie i PHP.

Wiązanie typu

- **korzyść:** elastyczność

```
void zamiana(int i, int j, char a[]) {
 char pom;
 pom = a[i]; a[i] = a[j]; a[j] = pom;
}
```

— ta funkcja z C **nie będzie działać** na tablicach innych niż znakowe; działałaby bez przeszkód w języku o dynamicznym wiązaniu typu;

- **wady**:
  - (jak zwykle) zmniejszona odporność na błędy; np. gdyby w języku dynamicznie typowanym programista tu napisał `a = pom;`, to tablica `a` przestałaby istnieć;
  - koszt; w czasie działania zmienne muszą „wozić ze sobą" informację o typie;
  - koszt; wielkość pamięci na zmienną musi być zmienna.

Wiązanie typu

Wyprowadzanie typu — język ML i pochodne:

```
- fun kwad(x) = x*x;
val kwad = fn : int -> int
- kwad(16);
val it = 256 : int
- kwad(3.15);
stdIn:3.1-3.11 Error: ...
- fun kwadr(x:real) = x*x;
val kwadr = fn : real -> real
- kwadr(3.14);
val it = 9.8596 : real
- kwadr(16);
stdIn:5.1-5.10 Error: ...
- fun zast(f,x) = f(x);
val zast = fn : ('a -> 'b) * 'a -> 'b
- zast(kwad,16);
val it = 256 : int
```

def. funkcji

`int` jest domyślny

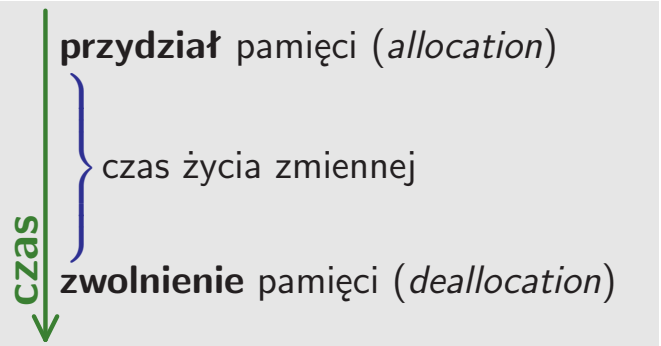
taka funkcja nie może mieć argumentu rzeczywistego

taka funkcja nie może mieć argumentu całkowitego

typ generyczny fkcji

Wiązanie pamięci

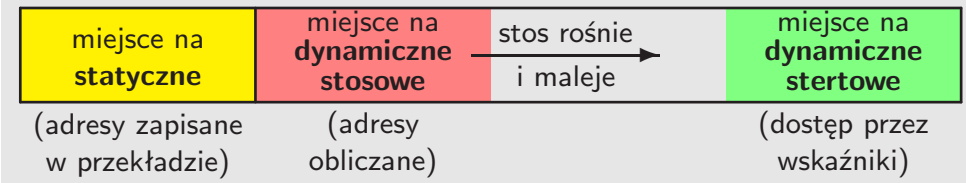
Zmienna jest powiązana z komórką pamięci.



Rodzaje wiązań pamięci:

- statyczne,
- dynamiczne na stosie,
- dynamiczne na stercie jawne,
- dynamiczne na stercie niejawne

Przykładowy schematyczny przydział pamięci różnym rodzajom zmiennych po skompilowaniu programu:



Wiązanie pamięci

Zmienne statyczne:

Adres:
ustalony w kompilacji, niezmienny przez cały czas działania programu

Zastosowanie:
zmienne globalne (deklarowane poza wszystkimi funkcjami)
zmienne lokalne, które mają zachowywać wartość między wywołaniami

Zalety:
szybki dostęp (nie trzeba liczyć adresu)

Wady:
mała elastyczność

Język, stosujący tylko statyczne wiązania, nie może mieć funkcji rekursywnych.

Jeśli program ma dwie funkcje, każda z nich potrzebuje dużej tablicy, ale tylko na chwilę, ale **nigdy** nie działają jednocześnie — mogłyby używać tego samego miejsca w pamięci, jednak ze zmiennymi statycznymi to jest niemożliwe.

Wiązanie pamięci

Zmienne dynamiczne na stosie:

Adres:
obliczany ze stanu stosu w czasie działania programu
przydział pamięci w chwili wywołania funkcji
zwolnienie pamięci w chwili wyjścia z funkcji

Zastosowanie:
zmienne lokalne i parametry funkcji

Zalety:
elastyczność — np. służą do implementacji rekursji

Wady:
koszt obliczania adresu w trakcie działania programu

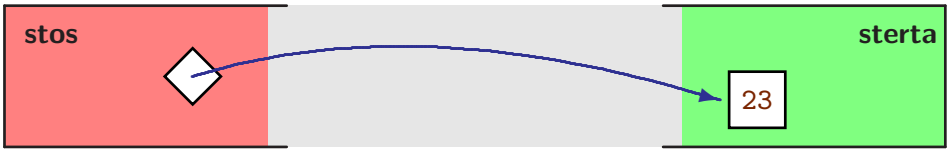
Wiązanie pamięci

Zmienne dynamiczne na stercie:

Adres:

dostęp tylko przez wskaźniki z pamięci statycznej lub stosu
przydział pamięci jawną komendą:

np. w C: `int* wsk = (int*)malloc(sizeof(int)); *wsk=23`
w Jawie: `Integer wsk = new Integer(23);`



zwolnienie pamięci: albo jawną komendą (np. w C `free`),
albo przez automatyczny zbieracz śmieci (np. w Jawie)

Zastosowanie:

konstrukcja list, drzew i innych struktur dynamicznych