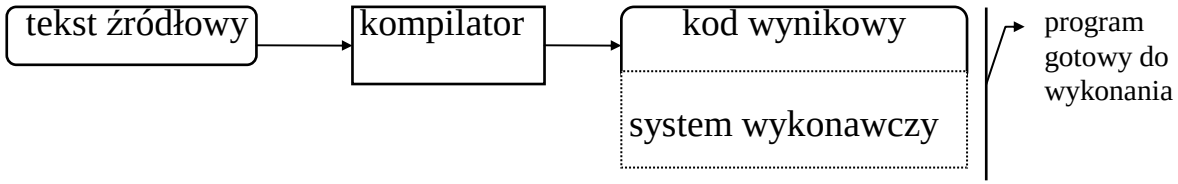
14. Elementy systemu wykonawczego

14.1 Wprowadzenie

- ***system wykonawczy*** (ang. *run-time system*) stanowi zestaw danych i operacji realizujących tzw. „maszynę wirtualną” języka programowania (na „maszynie” tej można uruchamiać wygenerowane w trakcie kompilacji programy wynikowe):



- ***stopień złożoności systemu wykonawczego*** zależy od ***semantyki konstrukcji*** zawartych w języku źródłowym
- ***wpływ*** na kształt systemu wykonawczego mają: ***styl*** źródłowego ***języka*** programowania, ***struktura modularna*** i ***reguły widoczności*** identyfikatorów, istnienie ***konstrukcji*** zapewniających ***współbieżność***, istnienie ***dynamicznych obiektów*** (programowanie zorientowane obiektowo), istnienie ***rekursji*** i in.
- ***składowe*** systemu wykonawczego to: ***system zarządzania pamięcią***, ***system*** zarządzania ***przydziałem procesora*** oraz system obsługi ***operacji we/wy***

14.2 Zarządzanie pamięcią

- pamięć operacyjna przydzielona programowi podczas jego wykonywania może zawierać:

kod programu
dane statyczne
stos
sterta

- ***dane statyczne*** to zadeklarowane w programie statyczne (tzn. nie zmieniające wielkości w trakcie działania programu) zmienne globalne

- struktura ***stosu*** (pamięć i procedury obsługi) jest niezbędnym elementem systemu wykonawczego języka programowania dopuszczającego ***rekurencyjne*** wywoływanie procedur (funkcji)

- ***sterta*** (ang. *heap*) jest strukturą umożliwiającą realizację dynamicznego przydziału pamięci (ang. DSA-*dynamic storage allocation*) w sytuacji, gdy strategia przydziału pamięci na stosie nie wystarcza (nie daje się przedstawić w postaci drzewa).

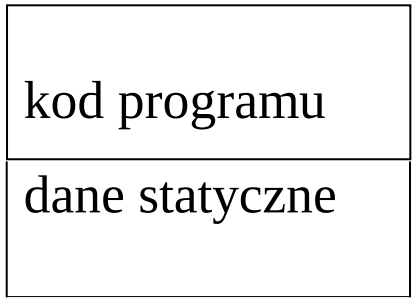
14.2.1 Pojęcie rekordu aktywacji

- elementami umieszczanymi na stosie (i zdejmowanymi z niego) są reprezentacje wywoływanych modułów (procedur lub funkcji) zwane ***rekordami aktywacji***, mogące zawierać następujące pola:

- rekord aktywacji jest umieszczany na szczycie stosu w momencie wywołania modułu i zdejmowany ze szczytu w momencie „powrotu z wywołania” (przekazania sterowania modułowi wywołującemu)
- ***obliczony wynik*** reprezentuje wartość obliczoną w trakcie wykonania (głównie) funkcji; często wynik ten jest przekazywany w rejestrze
- w polu ***parametrów aktualnych*** moduł wywołujący przekazuje dane modułowi wywoływanemu
- ***łącze dynamiczne*** (ang. *dynamic link*) służy do wskazywania początku rekordu aktywacji modułu wywołującego (w celu odtworzenia stanu stosu po powrocie z modułu wywołanego)
- ***łącze statyczne*** (ang. *static link*) umożliwia realizację dostępu do nazw nielokalnych (np. w Pascalu)
- pole ***stanu maszyny*** przechowuje m. in. zawartość rejestrów i licznika rozkazów, pozwalając odtworzyć stan maszyny, gdy sterowanie powróci z modułu wywołanego
- pole ***zmiennych lokalnych*** służy do przechowywania wartości zmiennych zadeklarowanych w treści wywoływanej procedury (funkcji)
- ***zmienne tymczasowe*** są generowane w trakcie przekładu wyrażeń
- wielkość rekordu aktywacji, w większości przypadków może być obliczona podczas kompilacji programu
- adresy zmiennych umieszczonych na stosie obliczane są względem (ang. *variable offset*) wskaźnika szczytu stosu.

14.2.2 Strategia statycznego przydziału pamięci

- przydział ***statyczny*** (ang. *static allocation*):



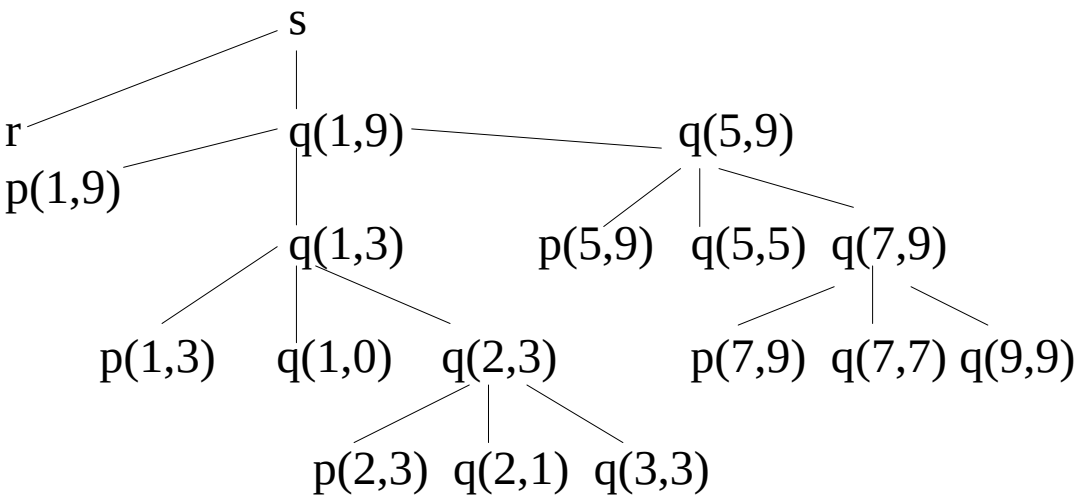
- dane statyczne obejmują zmienne globalne oraz rekordy aktywacji modułów; kompilator oblicza i umieszcza w kodzie programu adresy wszystkich zmiennych
- niedopuszczalne są: rekurencyjne wołania modułów, dynamiczny przydział pamięci
- strategię statyczną zastosowano w kompilatorach języka Fortran

14.2.3 Strategia przydziału pamięci na stosie (ang. *stack allocation*)

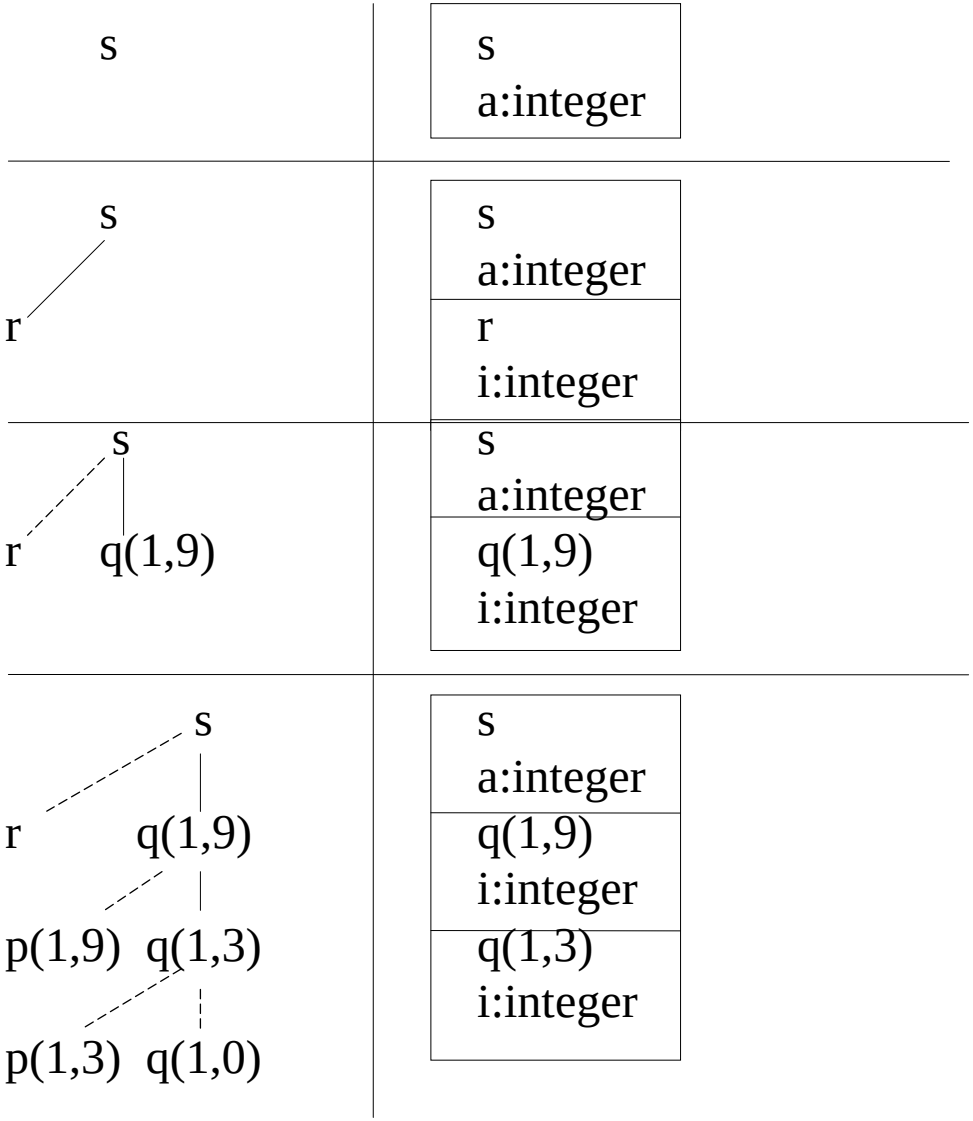
- rozważmy przykładowy program:

```
program s;
   var a : integer;
   procedure r;
      var i:integer;
     begin ...
     end;
   function p (x,y:integer):integer;
      var i,j,x,v:integer;
      begin ....
      end;
   procedure q(m,n:integer);
       var i:integer;
       begin if(n>m)
               then begin
                      i:=p(m,n);
                      q(m,i-1);
                      q(i+1,n);
                    end
       end;
  begin ... r; q(1,9)
  end.
```

oraz drzewo rekordów aktywacji:



- zawartość stosu rekordów aktywacji po dojściu do wywołania `q(2,3)`:



- ciąg operacji związanych z umieszczaniem rekordów aktywacji na stosie i ze zdejmowaniem ich ze stosu nosi nazwę „*calling sequences*”; jego wykonanie podzielone jest między moduł wywołujący a moduł wywoływany
- ***wywołanie***:

1) moduł wywołujący wyznacza i umieszcza na stosie wartości parametrów aktualnych
2) moduł wywołujący składuje na stosie wartość licznika rozkazów i wskaźnika szczytu stosu oraz zwiększa zawartość wskaźnika szczytu stosu tak, by wskazywał początek rekordu aktywacji modułu wywoływanego, sterowanie przejmuje moduł wywołany
3) moduł wywołany składuje w rekordzie informacje o stanie maszyny
4) moduł wywołany inicjuje wartości zmiennych lokalnych i rozpoczyna wykonanie swego kodu.

- ***powrót***:

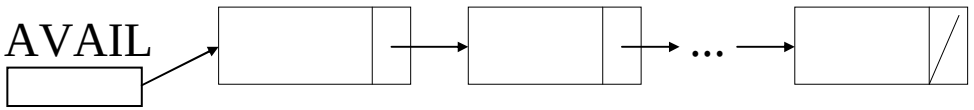
1) moduł wywołany umieszcza obliczoną wartość poniżej rekordu aktywacji modułu wywołującego
2) moduł wywołany odtwarza stan maszyny i wartość wskaźnika szczytu stosu i przekazuje sterowanie do modułu wywołującego
3) moduł wywołujący odczytuje obliczoną wartość.

- obsługa stosu rekordów aktywacji jest niezbędnym elementem systemu wykonawczego w maszynach języków zawierających rekurencyjne wywołania modułów (np. Pascal, C, Java i in.).

14.2.4 Strategia przydziału pamięci na stercie (ang. *heap allocation*)

- jest niezbędna, gdy przepływu sterowania między modułami nie można opisać za pomocą drzewa rekordów aktywacji (np. obiekty, które nie są automatycznie usuwane z pamięci – przykład operacji `new` i `dispose` w Pascalu)
- zarządzanie pamięcią na stercie nosi nazwę ***dynamicznego przydziału pamięci*** *(DPP)*
- podstawowe operacje DPP to: ***alokacja segmentu pamięci*** – `allocate(size, address)`, ***zwolnienie segmentu o wskazanym adresie*** – `deallocate(address)` oraz ***dostęp do wskazanej komórki wskazanego segmentu*** – `access(address, offset)`
- segmenty są pamiętane w ***blokach pamięci***; istnieją różne metody reprezentacji bloków i strategii zarządzania nimi:

1) wolna pamięć może być podzielona na ***bloki o jednakowych rozmiarach***, połączone w listę, której początek jest przechowywany w zmiennej AVAIL:

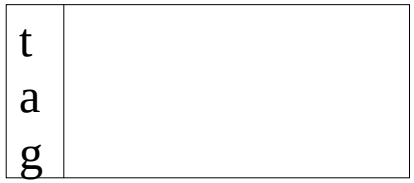


alokacja pamięci polega na odłączeniu bloku z listy bloków wolnych, dealokacja zaś na dołączeniu; problem fragmentacji wewnętrznej

2) DPP może zawierać ***bloki wolne i zajęte o różnych rozmiarach***:

A	wolny	B	C	wolny	D	wolny

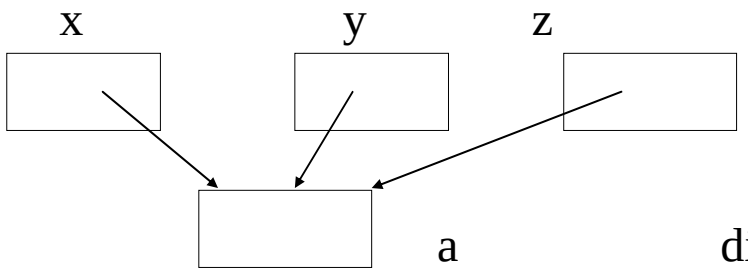
bloki o różnych rozmiarach mogą być reprezentowane metodą znacznikowania (ang. *tag*):



tag<0 : blok zajęty
tag>0 : blok wolny
|tag| = rozmiar bloku

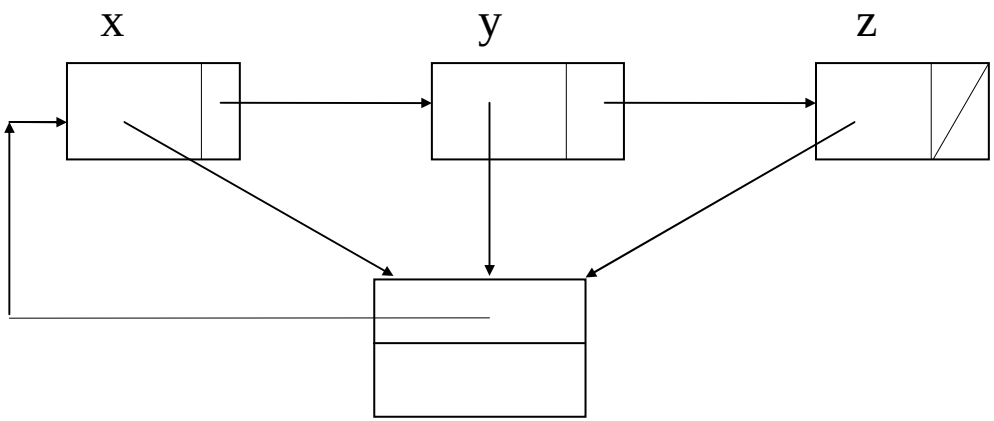
bloki mogą być między sobą powiązane; bloki wolne można przydzielać np. metodami: ***pierwszy pasujący*** (ang. *first-fit*), ***najlepszy pasujący*** (ang. *best fit*); powstają problemy fragmentacji zewnętrznej; sposobem rozwiązania jest ***skupianie bloków*** (ang. *compaction*) – przesuwanie wszystkich bloków zajętych tak, aby stanowiły zwarty segment (problem – uaktualnianie referencji),

- problem ***wiszących referencji*** (ang. *dangling references*):

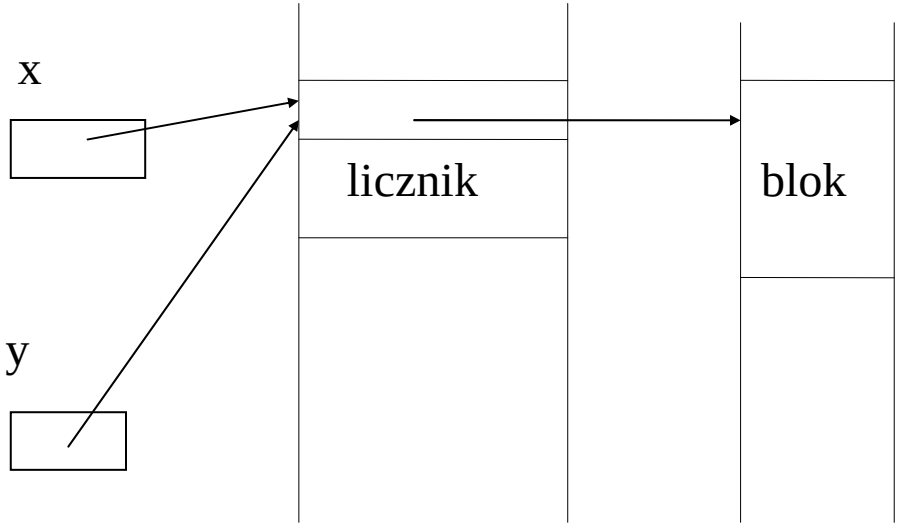



```
dispose(x);
write(y^.a);
```

rozwiązania: lista zmiennych referencyjnych, nagrobki (z licznikami, sygnaturowe)



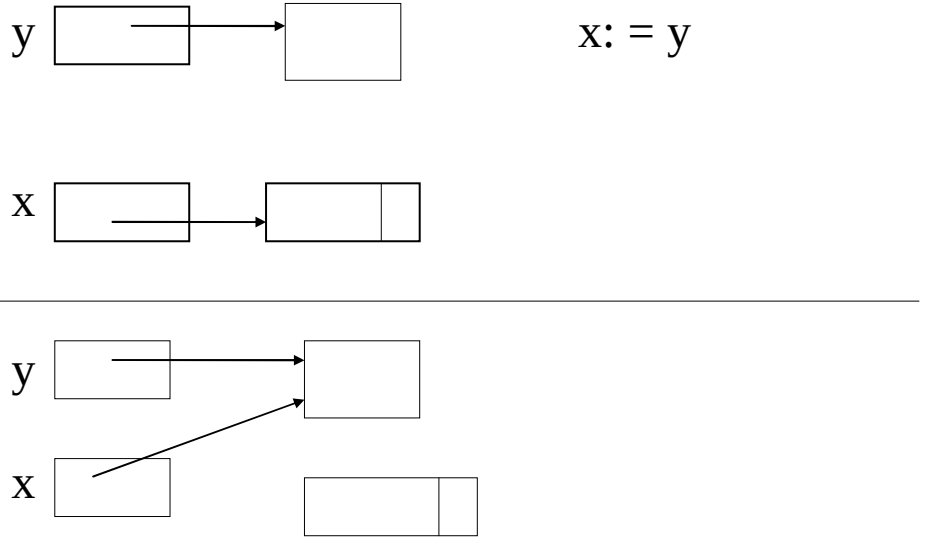
blok zawierający a



obszar nagrobków

obszar bloków

- ***zbieranie nieużytków*** (ang. *garbage collection*), to procedura automatycznego zwalniania bezużytecznej pamięci w systemach, w których nie ma operacji typu `dispose` (np. Lisp):



nieużytki

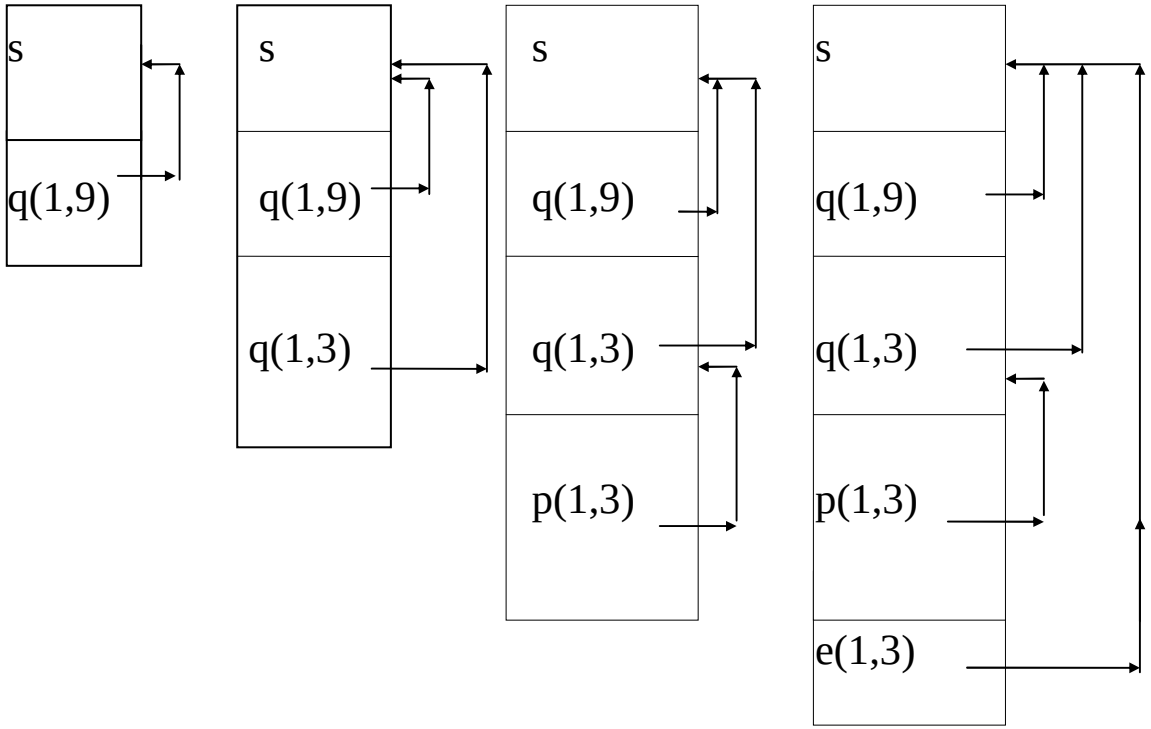
rozwiązania: zliczanie referencji, techniki markowania

14.2.5 Realizacja dostępu do nazw nielokalnych

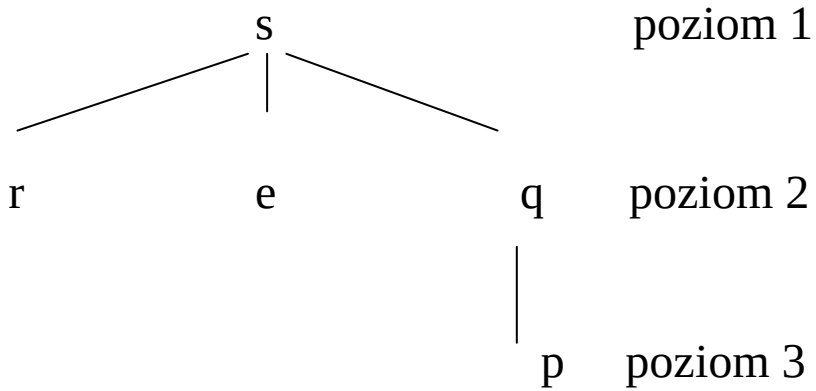
- wynika ze struktury modularnej programu i przyjętych reguł widoczności nazw
- ***reguły widoczności*** dzielą się ogólnie na: ***statyczne*** (ang. *lexical scope*) i ***dynamiczne*** (ang. *dynamic scope*)
- reguły statyczne wynikają z tekstowej struktury deklaracji modułów i reguł przesłaniania nazw (np. Pascal, C, Java, Fortran)
- reguły dynamiczne określają znaczenie nazwy w czasie wykonania programu na podstawie bieżących rekordów aktywacji (np. Lisp, APL, Snobol)
- metodą realizacji reguł statycznych są: ***łącza statyczne*** lub ***wektory display***

- przykładowy program:

```
program s;
  procedure r;
    begin  ... end;
  procedure e (i,j:integer);
    begin ... end;
  procedure q(m,n:integer);
    function p(y,z:integer):integer;
      begin ... e(i,j) ...end;
   begin ... end;
begin  ...  end;
```

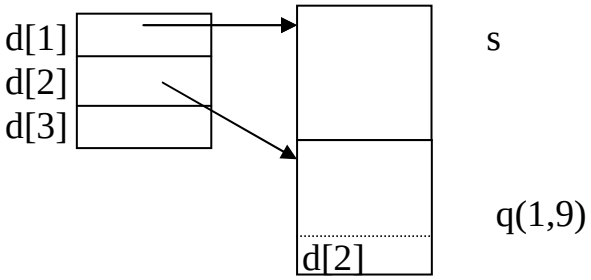


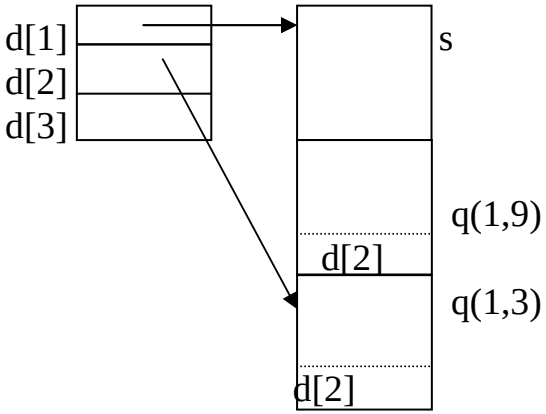
- łącza statyczne (zasady obliczania adresu zmiennej nielokalnej):

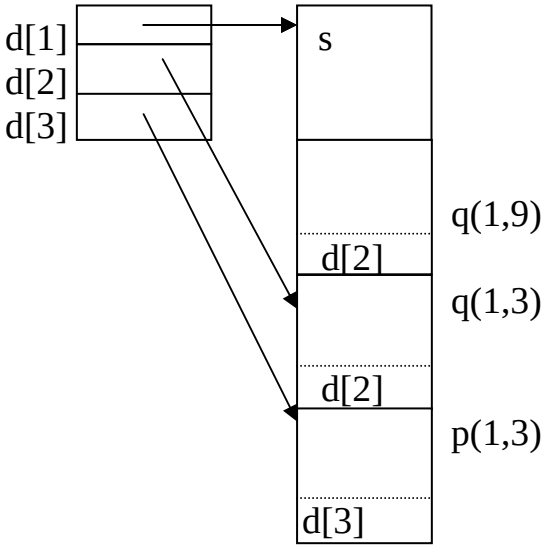


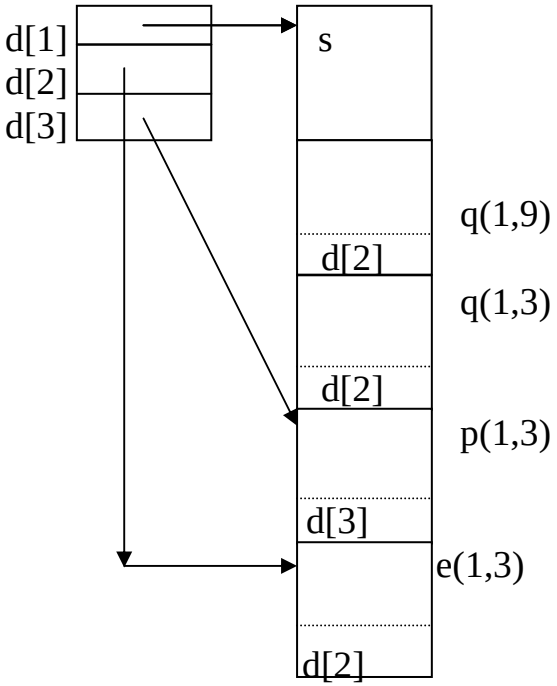
np. sterowanie jest w module p (poziom n_p=3, rekord aktywacji jest na szczycie stosu), odwołanie do zmiennej x zadeklarowanej w module s (poziom n_x=1); wykonaj $n_p - n_x$ przejść łańcuchem SL osiągając początek rekordu aktywacji modułu zawierającego deklarację zmiennej x; oblicz adres względny zmiennej względem początku rekordu aktywacji (*offset*);

- ***wektor display*** – tablica wskaźników rekordów aktywacji na stosie; dla każdego poziomu w drzewie deklaracji wektor zawiera jedno pole; liczba pól wektora obliczana jest na etapie kompilacji:




d[1]
d[2]
d[3]
s
q(1,9)
d[2]
q(1,3)
d[2]

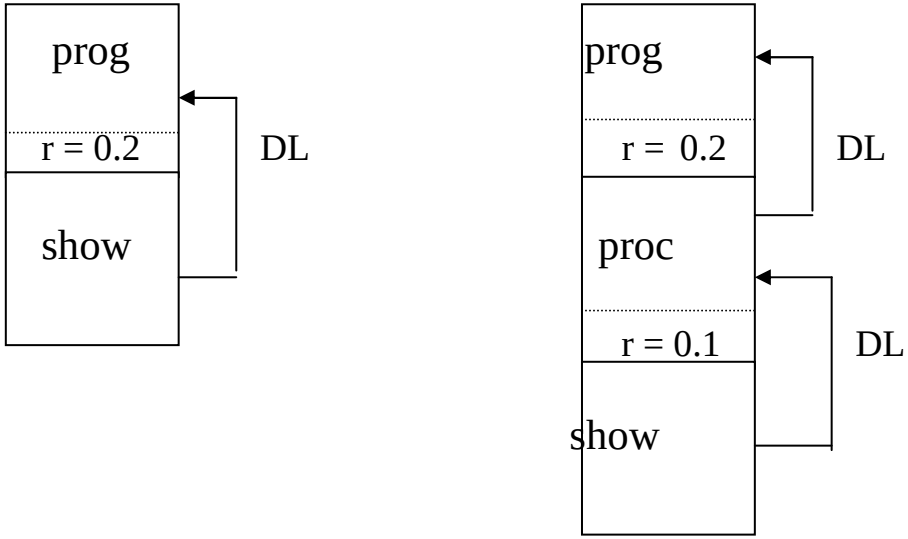

d[1]
d[2]
d[3]
s
q(1,9)
d[2]
q(1,3)
d[2]
p(1,3)
d[3]



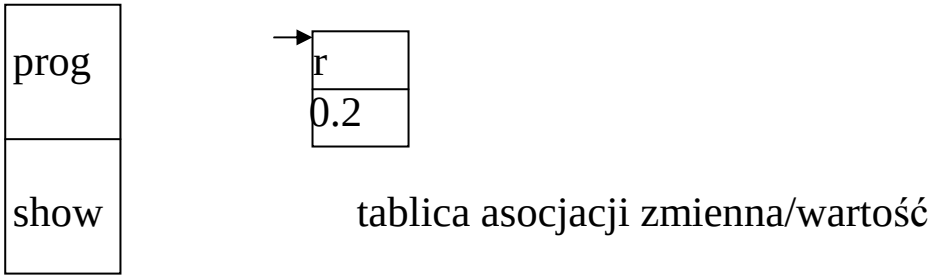
- dostęp dynamiczny: dostęp głęboki (ang. *deep access*) i dostęp płytki (ang. *shallow access*),
- przykład programu:

```
program prog;
  var r: real;
  procedure show;
    begin write(r:5:3) end;
  procedure proc;
    var r:real;
     begin r:=0.1; show end;
begin r:=0.2; show; proc; end.
```

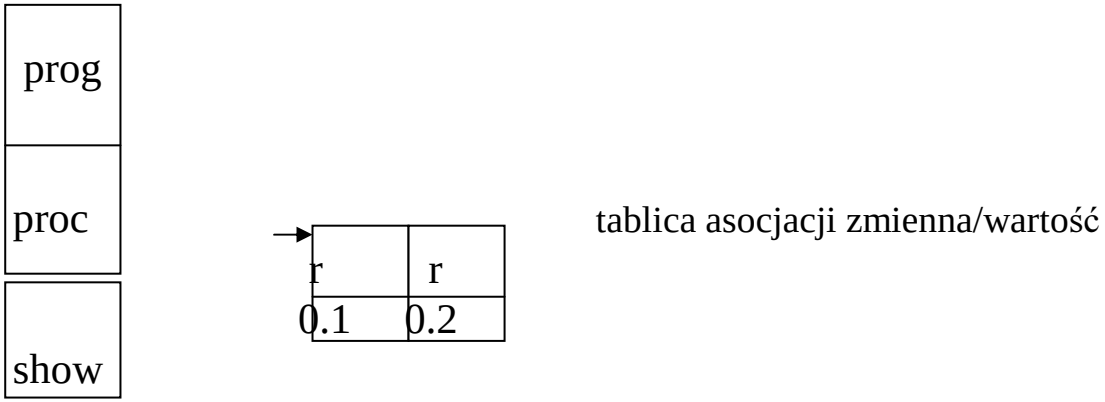
(zakres statyczny – `0.2 0.2`
zakres dynamiczny – `0.2 0.1`)



- realizacja dostępu głębokiego – wykorzystanie łączy dynamicznych (DL) na stosie do obliczania adresu zmiennej;



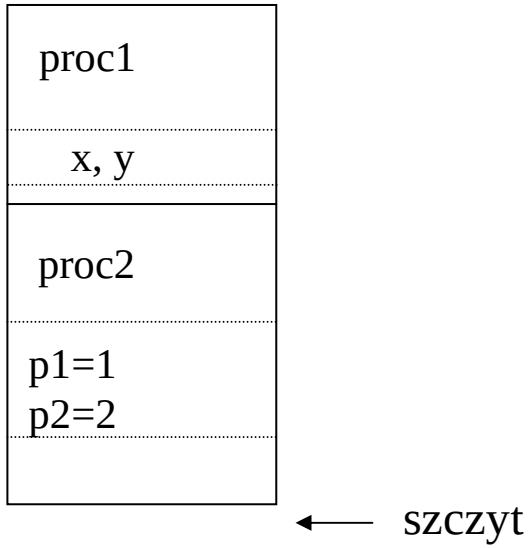
tablica asocjacji zmienna/wartość



14.2.6 Przekazywanie parametrów

- tryby przekazywania parametrów: przez wartość, przez referencję, copy-restore, przez nazwę (rzadko stosowane)
- przekazywanie parametrów przez wartość – wyznaczenie wartości parametrów przez moduł wywołujący, umieszczenie wyznaczonych wartości w rekordzie aktywacji modułu wywoływanego (po usunięciu ze stosu rekordu aktywacji modułu wywoływanego, wartości parametrów są tracone):

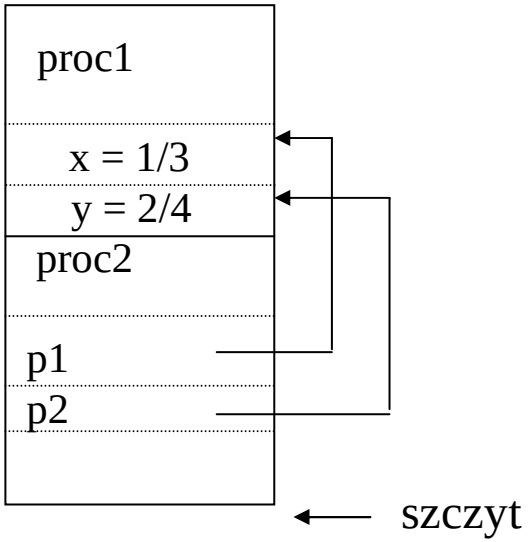
```
procedure proc1;
  var x, y:integer;
  begin x:=1; y:=2; proc2(x, y) end;
procedure proc2(p1, p2:integer);
  begin … :=p1+p2 end;
```



- przekazywanie parametrów przez referencję – wyznaczenie adresów parametrów aktualnych (także wyrażeń) przez moduł wywołujący i umieszczenie ich w rekordzie aktywacji modułu wywoływanego; moduł wywoływany działa na zmiennych dostępnych za pomocą tych adresów, zatem po usunięciu ze stosu rekordu aktywacji modułu wywoływanego, zmiany dokonane przez ten moduł są widoczne:

```
procedure proc1;
  var x, y:integer;
  begin x:=1; y:=2; proc2(x, y) end;
```

```
procedure proc2(var p1, p2:integer);
  begin p1:=3; p2:=4 end;
```



- przekazywanie parametrów typu copy-restore – wyznaczenie wartości parametrów aktualnych (z zachowaniem adresów zmiennych będących parametrami) przez moduł wywołujący i przekazanie ich do rekordu aktywacji modułu wywoływanego; moduł wywoływany działa na tych wartościach; w momencie powrotu do modułu wywołującego te wartości, które odpowiadają parametrom będącym zmiennymi są do tych zmiennych kopiowane