Wskaźniki

Wskaźniki to adresy pamięci, które pokazują komputerowi miejsce gdzie znajdują się jakieś dane. Możesz przykładowo przechowywać w pamięci wskaźnik na obrazek bitmapowy, podczas gdy przechowywanie obrazu w tablicy byłoby niezbyt praktyczne. Pamięć do której odnoszą się wskaźniki można w każdej chwili zwalniać, deklarować i zmieniać.

Po co są wskaźniki?
Bez wskaźników nie byłoby możliwe nieograniczone tworzenie nowych danych, np. w słowniku. Gdyby nie wskaźniki mielibyśmy z góry określoną liczbę słów, które słownik byłby w stanie zapamiętać. W grach komputerowych, liczba ludzików, zbudowanych elementów w strategii, zawsze byłaby z góry ograniczona. Nie moglibyśmy w systemie uruchomić większej liczby programów niż zaplanowano.

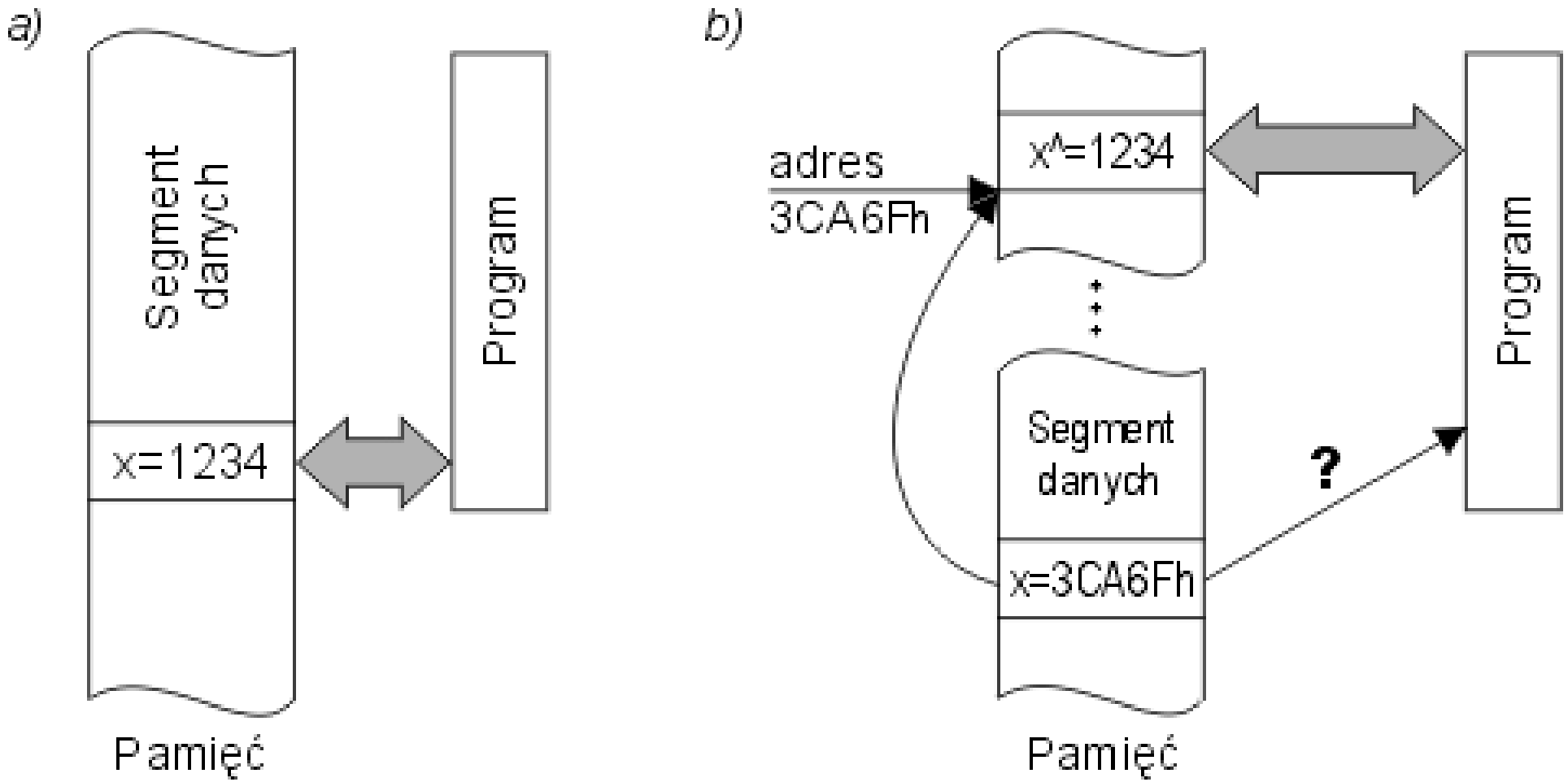
Budowa wskaźników*

Ze względu na ryzyko zawieszenia programu wskaźniki są traktowane jako typy, które należy umiejętnie stosować. Wskaźniki na ogół zbudowane są z 4 bajtów. Pierwsze 2 określają segment danych, kolejne 2 offset.

Pamięć konwencjonalna (pierwszy 1MB pamięci komputera) podzielony jest na segmenty po 16KB każdy. Tak więc 1 segment ma adres 0, drugi 1, trzeci 2, itd... Ofset to przemieszczenie względem segmentu. Dzięki ofsetowi możemy dojść do każdego jednego bajtu pamięci. Adres typu pointer składa się z dwóch bajtów segmentu, dwóch ofsetu i jest najczęściej zapisywany w kodzie szesnastkowym, dzięki czemu jest bardziej czytelny, np.

$a000:00, $b800:00

Wskaźnik do zmiennej jest po prostu jej adresem, czyli liczbą opisującą jej położenie w pamięci. Różnicę pomiędzy zmienną statyczną a wskazywaną ilustruje poniższy rysunek:



Typ pointer

Typ wskaźników nazywa się pointer. Gdy jakąś zmienną zadeklarujesz jako poiner, będzie ona wskaźnikiem. Wskaźniki mogą adresować różne typy zmiennych. Mogą adresować liczby, napisy, tablice, itd. Gdy adresują konkretny typ, nie deklaruje się ich jako pointer, tylko tworzy swój własny typ -o tym przeczytasz za moment.

Podstawianie wskaźników

Żeby wskaźnik wskazywał adres jakiejś zmiennej, możemy napisać:

```
var
  wskaznik : pointer; {lub podobny typ}

{...}

wskaznik := @nazwa_zmiennej;
```

znacznik @ oznacza pobranie adresu zmiennej. Gdybyśmy chcieli teraz coś zapisać pod wskazanym miejscem wystarczy wywołać wskazywane miejsce, za pomocą znaku ^

```
 wskaznik^ := nowa_wartosc;
```

W podanym przykładzie trzeba by wskaźnik był zadeklarowany jako wskaźnik na liczbę. Gdybyśmy bez rzutowania i bez określenia rodzaju wkaźnika próbowali mu podstawić wartość wystąpiłby błąd "illegal assigment".

Jak zadeklarować wskaźnik na liczbę?
np.

```
type p_liczba = ^integer; {utworzenie nowego typu - wskaźnika na liczbę}
var liczba : p_liczba;
  i : integer;

begin
  liczba := @i;
  i:=2;
  liczba^:=4;
end.

{teraz i = 4 a nie 2!}
```

Można ewentualnie potraktować typ pointer jako wskaźnik na liczbę co nazywa się rzutowaniem -dzięki temu unikniemy błędu a powiemy kompilatorowi, że wpisujemy w niego liczbę Integer

```
 var liczba : pointer;
    i : integer;

begin
  liczba := @i;
  i:=2;
  integer(liczba^):=4;
end.
```

Drugą **bardzo ważną** i pożyteczną ce**chą zmiennych wskazywany**ch jest **możliwość ich tworzenia i niszczenia w zależności od potrzeb**. Wymaga to co prawda użycia specjalnych procedur, pozwala jednak na znacznie **bardziej efektywną gospodarkę pamięcią**.

W odróżnieniu od statycznych zmiennych globalnych, istniejących przez cały czas wykonywania programu, zmienne wskazywane nal**eżą do klasy zmiennych *dynamicznych* (**czyli istnieją dokładnie wtedy, gdy życzy sobie tego programista).

Sam proces utworzenia zmiennej dynamicznej polega na zarezerwowaniu odpowiedniego obszaru pamięci i zapamiętaniu adresu tego obszaru we wskaźniku wskazującym na zmienną. Z kolei **usunięcie** zmiennej powoduje "**zwolnienie rezerwacji**" (zawartości zmiennej oraz wskaźnika nie są fizycznie niszczone, ale lepiej się już do nich nie odwoływać).

Ceną za możliwość swobodnego przydzielania i zwalniania pamięci jest konieczność bezwzględnego inicjalizowania wskaźników, które przed utworzeniem wskazywanej zmiennej mają wartość nieokreśloną (zwykle zero, co odpowiada wskaźnikowi **nil**, nie wskazującemu na żaden obiekt). Próba odczytu zmiennej wskazywanej przez taki wskaźnik dostarczy "tylko" bezsensownego wyniku, natomiast próba zapisu (w przypadkowe miejsce pamięci!!) może skończyć się zawieszeniem komputera lub zupełnie nieprzewidzianym jego zachowaniem.

PAMIETAJ O INICJALIZACJI WSKAZNIKÓW!

```
TYPE
        PS = ^STRING;
VAR
        PS_1 : PS;
BEGIN
        IF PS_1 = NIL THEN WRITLN('ZMIENNA PS_1 MA WARTOSC NIL');
        NEW(PS_1);
        IF PS_1 <> NIL THEN WRITELN('ZMIENNA PS_1 NIE MA JUŻ WARTOŚCI NIL');
END.
```

Turbo Pascal oferuje kilka metod tworzenia i usuwania zmiennych dynamicznych, z których najpopularniejszą realizuje para procedur new i dispose:

new(wskaźnik-do-zmiennej)
dispose(wskaźnik-do-zmiennej)

Procedura new wykonuje czynności związane z utworzeniem zmiennej wskazywanej, natomiast dispose - operacje związane z jej usunięciem.

Drugą parę zarządzającą dynamicznym przydziałem pamięci tworzą procedury GetMem i FreeMem:

GetMem(wskaźnik, rozmiar-bloku)
FreeMem(wskaźnik, rozmiar-bloku)

W odróżnieniu od pary new-dispose, procedury te wykorzystują wskaźniki amorficzne (typu pointer) i służą do bardziej "wewnętrznego" manipulowania pamięcią, tj. przydzielania i zwalniania bloków bajtów (a nie zmiennych wskazywanych jakiegoś konkretnego typu). Wielkość przydzielanego lub zwalnianego bloku (w bajtach) określa parametr rozmiar-bloku. Korzystając z obu grup procedur musisz pamiętać, że pamięć przydzielona przez GetMem nie może być zwolniona procedurą dispose i odwrotnie.

Ostatnią, chyba najrzadziej stosowaną parę tworzą procedury mark i release:

mark(wskaźnik)
release(wskaźnik)

Wykonanie procedury mark nie powoduje przydzielenia pamięci ani utworzenia zmiennej, a jedynie zapamiętanie bieżącej "wysokości" sterty w zmiennej wskaźnik. Zwolnienia całego obszaru sterty leżącego powyżej wskaźnika dokonuje się za pomocą procedury release. Obydwie procedury stosowane są - podobnie jak GetMem i FreeMem - głównie w programowaniu niskiego poziomu, do "masowego" zwalniania pamięci przydzielonej na stercie.

PRZYKŁAD:

```
program ZmienneDynamiczne;

type
    PString = ^string;          { wskaźnik do łańcucha }

var
    s : PString;                { zmienna typu wskaźnik do łańcucha }

begin
    writeln(s^);                { zmienna nie utworzona }
    new(s);                     { więc ją tworzymy }
    writeln(s^);                { utworzona, lecz nie zainicjalizowana }
    s^ := 'No wreszcie!';       { inicjalizujemy }
    writeln(s^);                { teraz jest OK }
    dispose(s);                 { usuwamy }
end.
```

Przykład 2:

```
PROGRAM RECORDY_DYNAMICZNIE;
TYPE OSOBA=RECORD
                IMIE:STRING[30];
                NAZWISKO:STRING[30];
                WIEK:BYTE;
        END;
VAR
        OSOBY : ARRAY[1..255] OF ^OSOBA;
        ILE, I : BYTE;
BEGIN
        WRITELN('ILE OSOB:');
        READLN(ILE);
        FOR I := 1 TO ILE DO
        BEGIN
                WRITELN('PRZYDZIELAM MIEJSCE W PAMIECI NA KOLEJNY REKORD');
                NEW(OSOBY[I]);
                WRITELN('POBIERAM DANE OSOBOWE);
                WRITELN('PODAJ IMIE');
                READLN(OSOBY[I]^.IMIE);
                WRITELN('PODAJ NAZWISKO');
                READLN(OSOBY[I]^.NAZWISKO);
                WRITELN('PODAJ WIEK');
                READLN(OSOBY[I]^.WIEK);
        END;
        WRITELN('WYPISUJE POBRANE OSOBY');
        FOR I := 1 TO ILE DO
                WRITELN('OSOBA NR.',I,' IMIE:',OSOBY[I]^.IMIE,' NAZWISKO:',OSOBY[I]^.NAZWISKO,' WIEK:',IMIE[I]^.WIEK);
        READLN;
END.
```