```
program Operatii_de_baza_la_alocarea_dinamica;

type
    { Definim un tip de date pointer la Integer. }
    PInteger = ^Integer;

    { Definim un tip de date pointer la Real. }
    PReal = ^Real;

    { Tip de date folosit pentru afisarea adreselor continute in pointeri.
      O variabila de tip pointer ocupa in memorie patru octeti si ea contine
      o adresa de memorie. Adresa de memorie se compune dintr-o adresa de
      segment si un deplasament (offset). Deplasamentul se memoreaza in
      primii din cei patru octeti, iar adresa de segment in ultimii doi
      octeti.
      Definim un tip record care are tot patru octeti si care contine
      doua campuri de cate doi octeti.
      Pentru a afisa o variabila de tip pointer, o convertim la acest
      tip record. Dupa conversie, prin accesarea celor doi membri ai
      structurii record putem afla valorile adresei de segment si a
      deplasamentului. }
    ConvPtr = record
            ofs : Word;
            seg : Word;
    end;

var
   { O variabila de tip Integer. }
   n : Integer;

   { O variabila de tip pointer la Integer. }
   pn : PInteger;

   { O variabila de tip Real. }
   r : Real;

   { O variabila de tip pointer la Real. }
   pr : PReal;

begin
     { Afisam dimensiunile ocupate de variabile in memorie. }
     WriteLn('Dimensiune Integer: ', SizeOf(n));
     WriteLn('Dimensiune pointer la Integer: ', SizeOf(pn));
     WriteLn('Dimensiune Real: ', SizeOf(r));
     WriteLn('Dimensiune pointer la Real: ', SizeOf(pr));

     { Atribuim valori variabilelor non-pointer. }
     n := 11;
     r := 11.11;

     { Afisam variabilele. Pentru variabilele non-pointer ne asteptam
       sa fie afisare valorile pe care le-am atribuit. Pentru variabilele
       pointer, neavand nici o valoare atribuita, se vor afisa valori
       aparent aleatoare.
       De fapt variabilele pointer sunt initializate cu valoare zero.
       Aceasta inseamna ca cele doua variabile pointer contin adresa zero,
       adica indica spre adresa zero din HEAP. Ca urmare la afisare vom
       vedea continutul memoriei HEAP de la adresa zero, privit o data sub
       forma de Integer (2 octeti) si apoi sub forma de Real (6 octeti). }
     WriteLn('n=', n);
     WriteLn('r=', r);
     WriteLn('pn^=', pn^);
     WriteLn('pr^=', pr^);
```

```
    { Ca sa ne convingem ca variabilele pointer sunt initializare cu zero,
      le afisam. Pentru a afisa adresa memorata intr-o variabila pointer,
      facem un artificiu: convertim variabila pointer la tipul "ConvPtr"
      si apoi afisam membrii "Seg" si "Ofs" (vezi declaratia tipului de
      date "ConvPtr"). E nevoie de acest artificiu pentru ca nu putem
      afisa direct variabilele pointer. }
    WriteLn('pn=', ConvPtr(pn).Seg, ':', ConvPtr(pn).Ofs);
    WriteLn('pr=', ConvPtr(pr).Seg, ':', ConvPtr(pr).Ofs);


    { Setam variabila pointer la Integer sa indice spre variabila "n".
      Cu alte cuvinte setam adresa memorata in "pn" sa fie chiar adresa
      la care se gaseste "n". Pentru a obtine adresa lui "n" folosim
      operatorul de adresa "@". }
    pn := @n;
    WriteLn('pn^=', pn^);
    WriteLn('pn=', ConvPtr(pn).Seg, ':', ConvPtr(pn).Ofs);

    { Setam acum variabila pointer la Real sa indice spre variabila "r".
      Cu alte cuvinte vom face ca adresa memorata in "pr" sa fie chiar
      adresa la care se gaseste "r". Folosim din nou operatorul de adresa
      "@" pentru a obtine adresa lui "r". }
    pr := @r;
    WriteLn('pr^=', pr^);
    WriteLn('pr=', ConvPtr(pr).Seg, ':', ConvPtr(pr).Ofs);


    { Vrem sa vedem ce se intampla daca atribuim variabilelor pointer
      valoarea "nil" (pointer spre nicaieri). Vom vedea ca de fapt
      se seteaza adresele memorate in variabilele pointer pe zero. }
    pn := nil;
    pr := nil;
    WriteLn('pn=', ConvPtr(pn).Seg, ':', ConvPtr(pn).Ofs);
    WriteLn('pr=', ConvPtr(pr).Seg, ':', ConvPtr(pr).Ofs);


    { Afisam cantitatea de memorie disponibila in HEAP. }
    WriteLn('Memorie disponibila in HEAP: ', MemAvail);

    { Alocam spatiu din HEAP pentru o variabila de tip Integer. Adresa
      alocata o vom memora in "pn". Vom vedea ca spatiul alocat in HEAP
      este umplut automat cu zero (cand afisam valoarea variabilei
      Integer alocate vom vedea ca este zero). Afisam si cate memorie mai
      ramane disponibila in HEAP. }
    New(pn);
    WriteLn('pn^=', pn^);
    WriteLn('pn=', ConvPtr(pn).Seg, ':', ConvPtr(pn).Ofs);
    WriteLn('Memorie HEAP disponibila dupa alocare Integer: ', MemAvail);

    { Mai alocam spatiu din HEAP si pentru o variabila de tip Real.
      Adresa alocata o vom memora in "pr". Vom vedea din nou cum
      spatiul alocat este umplut automat cu zero (si variabila Real
      proaspat alocata are valoarea zero). Urmarim din nou sa vedem
      cata memorie mai ramane disponibila in HEAP. }
    New(pr);
    WriteLn('pr^=', pr^);
    WriteLn('pr=', ConvPtr(pr).Seg, ':', ConvPtr(pr).Ofs);
    WriteLn('Memorie HEAP disponibila dupa alocare Real: ', MemAvail);

    { Eliberam memoria alocata din HEAP. Urmarim cum creste inapoi
      cantitatea de memorie disponibila. }
    Dispose(pn);
```

```
     WriteLn('Memorie HEAP disponibila dupa eliberare Integer: ', MemAvail);
     Dispose(pr);
     WriteLn('Memorie HEAP disponibila dupa eliberare Real: ', MemAvail);
end.
```