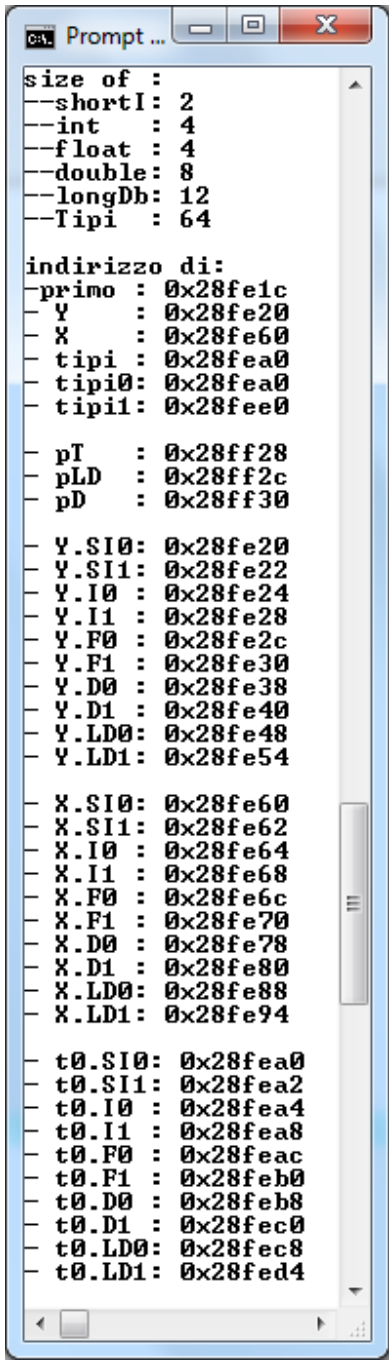
```
struct Tipi {      //----array di 2 elementi per ogni tipo
  short int    SI[2];    //--  2+2=   4 Byte
  int           I[2];    //--  4+4=   8 Byte
  float         F[2];    //--  4+4=   8 Byte
  double        D[2];    //--  8+8=  16 Byte
  long double LD[2];     //-- 12+12= 24 Byte
};                       //--totale= 60 Byte
int main(){

    short int*   pSI;        int*         pI;
    float*       pF;         double*      pD;
    long double* pLD;        Tipi*        pT;

    Tipi tipi[2], X, Y;
    int  primo;

cout << "size of :\n";
cout << "--shortI: "   << sizeof(short int)   <<endl;
cout << "--int   : "   << sizeof(int)         <<endl;
cout << "--float : "   << sizeof(float)       <<endl;
cout << "--double: "   << sizeof(double)      <<endl;
cout << "--longDb: "   << sizeof(long double) <<endl;
cout << "--Tipi  : "   << sizeof(Tipi)        <<endl;
cout << "\nindirizzo di:\n";
cout << "-primo : " << &primo    <<endl;
cout << "- Y    : "  << &Y       <<endl;
cout << "- X    : "  << &X       <<endl;
cout << "- tipi : "  << tipi     <<endl; //non occorre &
cout << "- tipi0: "  << &tipi[0] <<endl;
cout << "- tipi1: "  << &tipi[1] <<endl<<endl;
cout << "- pT   : "  << &pT      <<endl;
cout << "- pLD  : "  << &pLD     <<endl;
cout << "- pD   : "  << &pD      <<endl<<endl;

cout << "- Y.SI0: "  << &Y.SI[0]<<endl;
cout << "- Y.SI1: "  << &Y.SI[1] <<endl;
cout << "- Y.I0 : "  << &Y.I[0]  <<endl;
cout << "- Y.I1 : "  << &Y.I[1]  <<endl;
cout << "- Y.F0 : "  << &Y.F[0]  <<endl;
cout << "- Y.F1 : "  << &Y.F[1]  <<endl;
cout << "- Y.D0 : "  << &Y.D[0]  <<endl;
cout << "- Y.D1 : "  << &Y.D[1]  <<endl;
cout << "- Y.LD0: "  << &Y.LD[0] <<endl;
cout << "- Y.LD1: "  << &Y.LD[1] <<endl<<endl;
cout << "- X.SI0: "  << &X.SI[0]<<endl;
cout << "- X.SI1: "  << &X.SI[1] <<endl;
cout << "- X.I0 : "  << &X.I[0]  <<endl;
cout << "- X.I1 : "  << &X.I[1]  <<endl;
cout << "- X.F0 : "  << &X.F[0]  <<endl;
cout << "- X.F1 : "  << &X.F[1]  <<endl;
cout << "- X.D0 : "  << &X.D[0]  <<endl;
cout << "- X.D1 : "  << &X.D[1]  <<endl;
cout << "- X.LD0: "  << &X.LD[0] <<endl;
cout << "- X.LD1: "  << &X.LD[1] <<endl<<endl;
cout << "- t0.SI0: " << &tipi[0].SI[0] <<endl;
cout << "- t0.SI1: "  << &tipi[0].SI[1] <<endl;
cout << "- t0.I0 : "  << &tipi[0].I[0]  <<endl;
cout << "- t0.I1 : "  << &tipi[0].I[1]  <<endl;
cout << "- t0.F0 : "  << &tipi[0].F[0]  <<endl;
cout << "- t0.F1 : "  << &tipi[0].F[1]  <<endl;
cout << "- t0.D0 : "  << &tipi[0].D[0]  <<endl;
cout << "- t0.D1 : "  << &tipi[0].D[1]  <<endl;
cout << "- t0.LD0: "  << &tipi[0].LD[0] <<endl;
cout << "- t0.LD1: "  << &tipi[0].LD[1] <<endl<<endl;
```

programma: struct.cpp



Size of = Occupazione in Memoria Centrale
Viene esposta a video per i tipi standard e per la struttura **Tipi**

*NOTA: le variabili di tipo strutturato **Tipi** occupano **64** byte, 4 byte in più, perché vengono allineate alla parola (word)*

Indirizzi in M.C. (considerando le ultime 2 cifre esadecimali) a partire da **primo**:
- 28_{10} = $1C_{16}$ (**primo**)
- 32(=20_H,+4B di **primo**)
- 96 (=60_H, +64B di **Y**)
- 160(=$A0_H$, +64B di **X**)
- 224 (+64B di **tipi[0]**)

dopo **tipi[1]** (+4B) c'è:
- 296 =(1)28_H **pT**
- 300 =(1)$2C_H$(+4B) **pLD**

. . .

- 32 = 20_{16} (**Y.SI[0]**)
- 34 (=22_H +2Byte per SI)
- 36 (+2B per Short Int)
- 40 (+4B per Int)
- 44 (+4B per Int)
- 48 (+4B per Float)
- 56 (+4B per Fl. +4B per allineamento word)
- 64 (+8B per Double)
- 72 (+8B per Double)
- 84 (+12B per LD)
- 96 (+12B per LD)
- 98 (+2B per SI)
- 100 (+2B per SI)
- 104 (+4B per Int)

etc. etc.

per ogni variabile della struttura Tipi l'indirizzo del campo D[0] viene allineato alla parola (word di 8 byte) quindi il suo indirizzo non è a 4 byte da F[1] ma viene dopo 8 byte

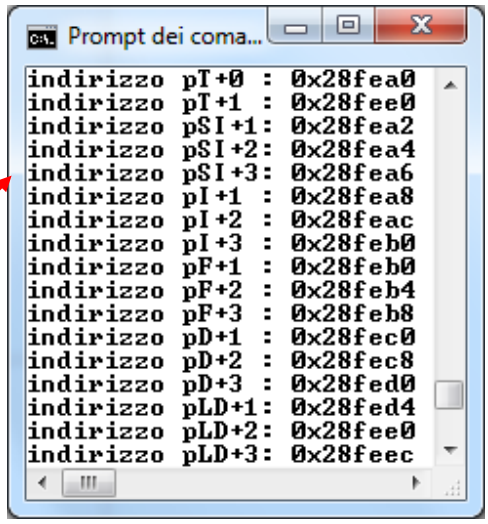
NOTE: le variabili vengono allocate in ordine inverso: le ultime variabili definite hanno un indirizzo più basso (in questo caso **primo** ha indirizzo più basso, poi **Y**, etc.), le prime hanno invece indirizzo più alto (i puntatori: **pSI**, **pI,** ...).

Un puntatore occupa 4byte.

L'indirizzo del primo elemento di **tipi** (**tipi[0]**) coincide con quello della tabella **tipi**. Indicando solo **tipi**, *cout* espone l'**indirizzo** dell'array **tipi**.

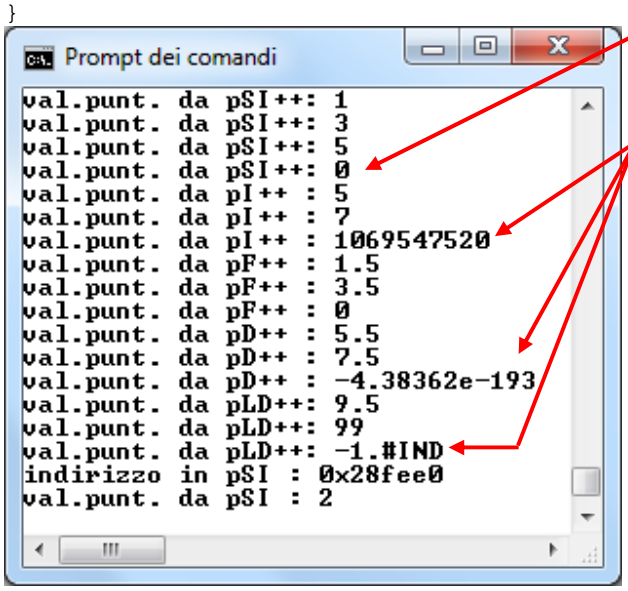
```
tipi[0].SI[0] = 1;      tipi[0].SI[1] = 3;      tipi[0].I[0] = 5;       tipi[0].I[1] = 7;
tipi[0].F[0]  = 1.5;    tipi[0].F[1]  = 3.5;    tipi[0].D[0] =5.5;      tipi[0].D[1] = 7.5;
tipi[0].LD[0] = 9.5;    tipi[0].LD[1] =99.0;
tipi[1].SI[0] = 2;      tipi[1].SI[1] = 4;      tipi[1].I[0] = 6;       tipi[1].I[1] = 8;
```

```
pT  = tipi;      //---l'array tipi indica l'indirizzo del primo elemento tipi[0]
for (int i=0; i<2; i++)
     cout << "indirizzo pT+"<< i <<" : " << pT+i  <<endl;
pSI = &tipi[0].SI[0];
for (int i=1; i<4; i++)
     cout << "indirizzo pSI+"<< i <<": " << pSI+i <<endl;
pI  = &tipi[0].I[0];
for (int i=1; i<4; i++)
     cout << "indirizzo pI+"<<i <<" : " << pI+i  <<endl;
pF  = &tipi[0].F[0];
for (int i=1; i<4; i++)
     cout << "indirizzo pF+"<<i <<" : " << pF+i  <<endl;
pD  = &tipi[0].D[0];
for (int i=1; i<4; i++)
     cout << "indirizzo pD+"<<i <<" : " << pD+i  <<endl;
pLD = &tipi[0].LD[0];
for (int i=1; i<4; i++)
     cout << "indirizzo pLD+"<<i <<": " << pLD+i <<endl;
for (int i=1; i<5; i++)
     cout<< "val.punt. da pSI++: "<<*pSI++ <<endl;
for (int i=1; i<4; i++)
     cout<< "val.punt. da pI++ : "<<*pI++   <<endl;
for (int i=1; i<4; i++)
     cout<< "val.punt. da pF++ : "<<*pF++   <<endl;
for (int i=1; i<4; i++)
     cout<< "val.punt. da pD++ : "<<*pD++   <<endl;
for (int i=1; i<4; i++)
     cout<< "val.punt. da pLD++: "<<*pLD++  <<endl;
pSI = &tipi[1].SI[0];
cout << "indirizzo in pSI : " <<  pSI <<endl;
cout << "val.punt. da pSI : " << *pSI <<endl;
}
```



Per i=3 i puntatori puntano, per errore, ad un byte posto all'interno della variabile del tipo successivo.

Per i=3 **pSI** punta all'indirizzo 0x28**fea6** (3^ byte di `tipi[0].I[0]`, dato di tipo **int** di 4byte).
Pur essendo un puntatore a **short int**, *cout* riesce ad interpretarne correttamente il contenuto perché considera i primi 2 byte dove sono rappresentate le cifre meno significative, cioè il valore 5 (i processori Intel, infatti, usano l'ordine di memorizzazione ***little endian***).
Per i=4, **pSI** considera i 2 byte successivi che contengono il valore 0 (cifre più significative dell'intero).



In generale i puntatori vanno incrementati ed utilizzati **senza superare la dimensione dell'array.**

Il programma, in generale, **NON riesce ad interpretare correttamente** il valore puntato dai puntatori **per i>=2**.

Per i=3 **pI** punta all'inizio di `tipi[0].F[0]` che contiene la rappresentazione *floating point* del numero 1.5; le cifre più significative della mantissa e dell'esponente sono nel terzo e quarto byte (per la memorizzazione ***little endian***) quindi i quattro byte del float vengono interpretati come un intero molto grande.

Per i=3 **pD** punta all'inizio di `tipi[0].LD[0]`

NOTE:

-1.#IND significa *Negative indefinite NaN (Not a Number)*

l'istruzione cout<<**pSI+i** non altera il contenuto di pSI

l'istruzione ***pSI++** invece:
- prima determina il contenuto puntato da pSI
- poi incrementa il puntatore **pSI** all'indirizzo successivo

(2 byte più avanti; **pI**++ va avanti di 4Byte … **pLD**++ va avanti di 12B)