Laboratorio di Programmazione Gr. 3 (N-Z)

Corso di Laurea in Informatica

Università degli Studi di Napoli Federico II

A.A. 2022/23

A. Apicella

Aritmetica dei puntatori

Sui (valori contenuti nelle variabili) puntatori sono definite le operazioni di *somma* e *differenza*, ma diverse da quelle canoniche.

dato un puntatore di nome p di tipo $T*$ ed un intero n l'operazione:

- `p +` n punta all'indirizzo $p + n \cdot sizeof(T)$
- `p -` n punta all'indirizzo $p - n \cdot sizeof(T)$

dati due puntatori rispettivaente con nomi `p` e `q` ed entrambi di tipo $T*$, l'operazione

- `p - q` punta all'indirizzo $\frac{p-q}{sizeof(T)}$

Queste operazioni possono essere utili nel caso di accesso ad aree di memoria continua (esempio, array)

In [1]:
```
#include <stdio.h>
int main()
{
    int v[4]   = {42, 101,106, 108};
    int* punt  = &v[0];
    int* punt2 = punt;
    printf("*punt:      %d\n",*punt);
    printf("incremento punt di 1...\n");
    punt = punt + 1;
    printf("*punt:      %d\n",*punt);
    printf("incremento punt di 1...\n");
    punt = punt + 1;
    printf("*punt:      %d\n",*punt);
    printf("decremento punt di 1...\n");
    punt--;
```



```
    printf("*punt:      %d\n",*punt);
    printf("incremento punt di 2...\n");
    punt += 2;
    printf("*punt:      %d\n",*punt);
    printf("decremento punt di 1 ed aggiungo 200 al valore a cui punt
    *(--punt) += 200;
    printf("*punt:      %d\n",*punt);
    printf("*punt+1:    %d\n",*punt+1);
    printf("*(punt+1):  %d\n",*(punt+1));



    printf("punt punta a %d, l'elemento iniziale del vettore è %d.\n"
    printf("punt: %llu (hex: %p), punt2: %llu (hex: %p)\n", punt, pur
    printf("tra %d e %d ci sono %d elementi di dimensione %d Byte\n",
           *punt, *punt2,punt-punt2, sizeof(*punt));


    return 0;
}
```

```
*punt:      42
incremento punt di 1...
*punt:      101
incremento punt di 1...
*punt:      106
decremento punt di 1...
*punt:      101
incremento punt di 2...
*punt:      108
decremento punt di 1 ed aggiungo 200 al valore a cui punta...
*punt:      306
*punt+1:    307
*(punt+1):  108
punt punta a 306, l'elemento iniziale del vettore è 42.
punt: 140721125449960 (hex: 0x7ffc30b1e0e8), punt2: 140721125449952
(hex: 0x7ffc30b1e0e0)
tra 306 e 42 ci sono 2 elementi di dimensione 4 Byte
```

L'operatore [ ]

dato un puntatore di nome `p` ed un intero contenuto in una variabile `n`, allora:

`p[n]` $\equiv$ `*(p+n)`

ad esempio, le istruzioni:

`*(p+1) = 5;`, `*p = 100;`, `*(p-3) = *(p+1)+3;`

possono essere riscritte come:

`p[1] = 5;`, `p[0] = 100;`, `p[-3] = p[1]+3;`

In [1]:
```
#include <stdio.h>
int main()
{
```

```
    int v[4]   = {42, 101,106, 108};
    int* punt  = &v[0];
    printf("punt[0]:      %d\n",punt[0]);
    punt += 3;
    printf("punt[0]:      %d\n",punt[0]);
    printf("punt[-2]:     %d\n",punt[-2]);

    return 0;
}
```

```
punt[0]:      42
punt[0]:      108
punt[-2]:     101
```

In [2]:
```
// Aritmetica dei puntatori
#include <stdio.h>
int main()
{
    int answer      = 42;
    int* punt       = &answer;
    printf("answer:        %d\n", answer);
    printf("punt:          %llu\n", punt);
    printf("*punt:         %d\n", *punt);

    punt = punt + 1;

    printf("====dopo incremento===\n");
    printf("punt:      %llu\n", punt);
    printf("*punt:     %d\n", *punt);

    return 0;

}
```

```
answer:        42
punt:          140731237675836
*punt:         42
====dopo incremento===
punt:      140731237675840
*punt:     -1955712192
```

In [2]:
```
#include <stdio.h>
int main()
{
    int answer = 42;
    int* punt  = &answer;
    printf("answer:       %d\n", answer);
    printf("&answer(hex): %p\n", &answer);
    printf("&answer(dec): %llu\n", &answer);
    printf("=======================\n");
    printf("punt:         %llu\n", punt);
    printf("*punt:        %d\n", *punt);
    printf("&punt(hex):   %p\n", &punt);
    printf("&punt(dec):   %llu\n", &punt);
    printf("===== ma anche... ======\n");
    printf("*(&answer):   %d\n", *(&answer));

    *punt = *punt + 10;
```



```
    printf("== modificando *punt ===\n");
    printf("answer:       %d\n", answer);
    printf("*punt:        %d\n", *punt);

    return 0;

}
```

```
answer:       42
&answer(hex): 0x7ffe546f9eec
&answer(dec): 140730315022060
=======================
punt:         140730315022060
*punt:        42
&punt(hex):   0x7ffe546f9ef0
&punt(dec):   140730315022064
===== ma anche... ======
*(&answer):   42
== modificando *punt ===
answer:       52
*punt:        52
```

In [1]:
```
#include <stdio.h>


void f(int f_v[]) // equivalente a void f(int* f_v)
{
    printf("sizeof(f_v): %d Byte\n", sizeof(f_v));
    printf("f_v+1: %p\n", f_v+1);

}

int main()
{
    int v[] = {3, 10, 42};
    int *p_v = v;
    printf("sizeof(v):   %d Byte\n", sizeof(v));
    printf("v:   %p;   v+1: %p\n", v, v+1);
    printf("sizeof(p_v): %d Byte\n", sizeof(p_v));
    printf("p_v: %p; p_v+1: %p\n", p_v, p_v+1);

    f(v);

    return 0;
}
```

```
/tmp/tmpzdnmvey4.c: In function ‘f’:
/tmp/tmpzdnmvey4.c:4:44: warning: ‘sizeof’ on array function paramet
er ‘f_v’ will return size of ‘int *’ [-Wsizeof-array-argument]
    4 |     printf("sizeof(f_v): %d Byte\n", sizeof(f_v));
      |                                            ^
/tmp/tmpzdnmvey4.c:2:12: note: declared here
    2 | void f(int f_v[]) // equivalente a void f(int* f_v)
      |        ~~~~^~~~
```

```
sizeof(v):   12 Byte
v:   0x7fff510af2ec;   v+1: 0x7fff510af2f0
sizeof(p_v): 8 Byte
p_v: 0x7fff510af2ec; p_v+1: 0x7fff510af2f0
sizeof(f_v): 8 Byte
f_v+1: 0x7fff510af2f0
```

Un esempio di utilizzo dell'aritmetica dei puntatori:

In [5]:
```
#include <stdio.h>


void stampa_vett(int v[], int n) // equivalente a void f(int* v, int
{
    printf("( ");
    for(int i = 0; i< n; i++)
        printf("%d, ", v[i]);
    printf("\b\b )");
}

int main()
{
    int v[] = {3, 10, 42, 8, 4 ,2};
    int n   = 6;

    //stampo l'array v a partire dalla terza posizione
    stampa_vett(v+2, n-2); // passo l'indirizzo della terza posizione

    return 0;
}
```

```
( 42, 8, 4, 2 )
```

Aritmetica dei puntatori su matrici allocate con operatore `[ ]`

In [6]:
```
#include <stdio.h>
#define MAX_COLS 10
void stampa_mat(int mat[][MAX_COLS], int n_r, int n_c)
{
    printf("(\n");
    for(int i = 0; i < n_r; i++)
    {
        for(int j = 0; j < n_c; j++)
            printf(" %2d, ", mat[i][j]);
        printf("\b\b\n");
    }
    printf(")\n");
}
int main()
{
    int M[][MAX_COLS] = {{3, 10, 42},
                         {8,7,64},
                         {11,22,33}
                        };

    stampa_mat(M, 3, 3);
    printf("M:    %p, *M:     %p, **M:          %d\n",M, *M, **M);
    /* NB: M non è da considerarsi  come un puntatore a puntatore ad
           ma è un puntatore ad un array di int, che è un tipo disti
    printf("M+1:  %p, *(M+1): %p, **(M+1):      %d\n",M+1, *(M+1), **
    printf("M+2:  %p, *(M+2): %p, **(M+2):      %d\n",M+2, *(M+2), **

    printf("M+2:  %p, *M+2:    %p, *(*M+2):     %d\n",M+2, *M+2, *(*M
    printf("M+2+1:%p, *(M+2)+1:%p, *(*(M+2)+1): %d\n",M+2+1, *(M+2)+1

    printf("M[2]: %p, *M[2]: %d\n",M[2], *M[2]);

    return 0;
}
```

```
(
  3,  10,  42
  8,   7,  64
 11,  22,  33
)
M:    0x7ffc06b36a50, *M:     0x7ffc06b36a50, **M:          3
M+1:  0x7ffc06b36a78, *(M+1): 0x7ffc06b36a78, **(M+1):      8
M+2:  0x7ffc06b36aa0, *(M+2): 0x7ffc06b36aa0, **(M+2):      11
M+2:  0x7ffc06b36aa0, *M+2:    0x7ffc06b36a58, *(*M+2):     42
M+2+1:0x7ffc06b36ac8, *(M+2)+1:0x7ffc06b36aa4, *(*(M+2)+1): 22
M[2]: 0x7ffc06b36aa0, *M[2]: 11
```

Una matrice `M` di tipo T è vista come un puntatore ad array di `MAX_COLS` elementi di tipo T, ossia:

`int (*) [MAX_COLS]` ← puntatore `(*)` ad array `[]` di tipo `int`.

Le parentesi tonde servono proprio a specificare che si tratta di un puntatore ad array, e non di un array di puntatori (ossia `int* [MAX_COLS]`).

Questa tipologia di puntatori non saranno trattati in questo corso, basti sapere che:

`int (*) [MAX_COLS]` $\not\equiv$ `int**`

Ad ogni modo, `M+n`, con `n` intero positivo, punta alla **riga** di indice `n`.

Si nota quindi che vale ancora:

`M[i]` $\equiv$ `*(M+i)`

Quindi `M[i][j]` $\equiv$ `*(*(M+i) + j)`

In quanto:

1. attraverso il primo livello di *dereferencing* (ossia `*(M+i)`) si accederà alla riga di indice `i`,

2. essendo tale riga un array di `[MAX_COLS]` elementi, posso accedere ai suoi elementi utilizzando l'aritmetica dei puntatori
3. In particolare, per accedere all'elemento di indice `j`, sfruttando sempre l'aritmentica dei puntatori è sapendo che `*(M+i)` è un array, basta sommarci `j`. `*(M+i)+j` sarà quindi l'indirizzo dell'elemento in riga di indice `i` e in colonna di indice `j`. Essendo un indirizzo, per avere il valore effettivo utilizzo un secondo livello di dereferencing, ossia `*(*(M+i) + j)`.

In [14]:
```
#include <stdio.h>
#define MAX_COLS 3

int main()
{
    //char v[] = {3, 8, 10};
    int M[][MAX_COLS] = {{3,10,20},
                         {8,4,2},
                         {10,14,16},
                         {1,2,3}
                        };
    //printf("sizeof(v): %lld\n", sizeof(v));
    long int M_piu_2   = (long int)(M + 2);
    long int M_piu_1   = (long int)(M + 1);
    long int M_0       = (long int) M;
    printf("M+1: %llu   M: %llu    diff: %llu\n", M_piu_1, M_0, M_piu
    printf("M+2: %llu   M: %llu    diff: %llu\n", M_piu_2, M_0, M_piu

    printf("sizeof(M):             %lld\n", sizeof(M));
    printf("sizeof(M[0]):          %lld\n", sizeof(M[0]));
    printf("sizeof(M[0][0]):       %lld\n", sizeof(M[0][0]));
    printf("n_c * sizeof(M[0][0]): %lld\n", MAX_COLS * sizeof(M[0][0]

    return 0;
}
```

```
M+1: 140721888639772   M: 140721888639760    diff: 12
M+2: 140721888639784   M: 140721888639760    diff: 24
sizeof(M):             48
sizeof(M[0]):          12
sizeof(M[0][0]):       4
n_c * sizeof(M[0][0]): 12
```

sia `M` definita come `int M[n_r][n_c]`, allora:

`M` punta alla *riga di indice 0* della matrice (quindi alla prima riga). `M+i` punterà quindi alla $i+1$-esima riga della matrice. L'indirizzo effettivo puntato da `M+i` sarà quindi dato da $M + i \cdot n_c \cdot sizeof(int)$.

In [ ]:
```

```