Aritmetica dei puntatori

Sui puntatori si possono anche effettuare operazioni aritmetiche, con opportune limitazioni

- somma o sottrazione di un intero
- sottrazione di un puntatore da un altro

Somma e sottrazione di un intero

Se `p` è un puntatore a `tipo` e il suo valore è un certo indirizzo `ind`, il significato di `p+1` è il primo indirizzo utile dopo `ind` per l'accesso e la corretta memorizzazione di una variabile di tipo `tipo`.

Esempio:

```
int *p, *q;
....
q = p+1;
```

Se il valore di `p` è l'indirizzo `100`, il valore di `q` dopo l'assegnamento è `104` (assumendo che un intero occupi 4 byte).

- Il valore calcolato in corrispondenza di un'operazione del tipo `p+i` dipende dal tipo `T` di `p` (analog. per `p-i`):
  Op. Logica: p = p+1 Op.Algebrica: p = p +`sizeof(T)`

  Esempio:

  ```
  int *pi;
  *pi = 15;
  pi=pi+1;
  ```
  $\Longrightarrow$ `pi` punta al prossimo `int` (4 byte dopo)

  Esempio:

  ```
  double *pd;
  *pd = 12.2;
  pd = pd+3;
  ```
  $\Longrightarrow$ `pd` punta a 3 `double` dopo (24 byte dopo)

  Esempio:

  ```
  char *pc;
  *pc = 'A';
  pc = pc - 5;
  ```
  $\Longrightarrow$ `pc` punta a 5 `char` prima (5 byte prima)
- Possiamo anche scrivere: `pi++;` `pd+=3;` `pc-=5;`

Puntatore a puntatore

- Le variabili di tipo puntatore sono variabili come tutte le altre: in particolare hanno un indirizzo che può costituire il valore di un'altra variabile di tipo puntatore a puntatore.

Esempio:

```
int *pi, **ppi, x=10;
pi = &x;
ppi = &pi;
printf("pi = %p ppi = %p *ppi = %p\n", pi, ppi, *ppi);
printf("*pi = %d **ppi = %d x = %d\n", *pi, **ppi, x);
```

```
pi = 0x22ef34   ppi = 0x22ef3c   *ppi = 0x22ef34
*pi = 10        **ppi = 10        x = 10
```


Esempi

```
int a, b, *p, *q;
a=10;
b=20;
p = &a;
q = &b;
*q = a + b;
a = a + *q;
q = p;
*q = a + b;
printf("a=%d b=%d *p=%d *q=%d", a,b,*p,*q);
```

Quali sono i valori stampati dal programma?

Esempi (contd.)

```
int *p, **q;
int a=10, b=20;
q = &p;
p = &a;
*p = 50;
**q = 100;
*q = &b;
*p = 50;
a = a+b;
printf("a=%d   b=%d  *p=%d  **q=%d\n", a, b, *p, **q);
```

Quali sono i valori stampati dal programma?

Relazione tra vettori e puntatori

- In generale non sappiamo cosa contengono le celle di memoria adiacenti ad una data cella.
- L'unico caso in cui sappiamo quali sono le locazioni di memoria successive e cosa contengono è quando utilizziamo dei vettori.
- In C il nome di un vettore è in realtà un puntatore, inizializzato all'inidirizzo dell'elemento di indice 0.

  int `vet[10];` $\Longrightarrow$ `vet` e `&vet[0]` hanno lo stesso valore (un indirizzo)

  $\Longrightarrow$ `printf("%p %p", vet, &vet[0]);` stampa 2 volte lo stesso indirizzo.
- Possiamo far puntare un puntatore al primo elemento di un vettore.

  ```
  int vet[5];
  int *pi;
  pi = vet;
  ```
  è equivalente a `pi = &vet[0];`

Accesso agli elementi di un vettore

Esempio:

```
int vet[5];
int *pi = vet;
*(pi + 3) = 28;
```

- `pi+3` punta all'elemento di indice 3 del vettore (il quarto elemento).
- 3 viene detto offset (o scostamento) del puntatore.
- N.B. Servono le `()` perchè `*` ha priorità maggiore di `+`. Che cosa denota `*pi + 3` ?
- Osservazione:

`&vet[3]`	equivale a	`pi+3`	equivale a	`vet+3`
`*&vet[3]`	equivale a	`*(pi+3)`	equivale a	`*(vet+3)`

- Inoltre, `*&vet[3]` equivale a `vet[3]`
  - In C, `vet[3]` è solo un modo alternativo di scrivere `*(vet+3)`.
- Notazioni per gli elementi di un vettore:
  - `vet[3]` $\Longrightarrow$ notazione con puntatore e indice
  - `*(vet+3)` $\Longrightarrow$ notazione con puntatore e offset

- Un esempio che riassume i modi in cui si può accedere agli elementi di un vettore.

```
int vet[5] = {11, 22, 33, 44, 55};
int *pi = vet;
int offset = 3;

/* assegnamenti equivalenti */

vet[offset] = 88;
*(vet + offset) = 88;
pi[offset] = 88;
*(pi + offset) = 88;
```

- **Attenzione**: a differenza di un normale puntatore, il nome di un vettore è un puntatore costante
  - il suo valore non può essere modificato!
- ```
  int vet[10];
  int *pi;
  ```
  `pi = vet;` corretto

  `pi++;` corretto

  `vet++;` scorretto: `vet` e' un puntatore costante!
- È questo il vero motivo per cui non è possibile assegnare un vettore ad un altro utilizzando i loro nomi

```
int a[3]={1,1,1}, b[3], i;
for (i=0; i<3; i++)
    b[i] = a[i];
```

ma non `b=a` (`b` è un puntatore costante!)

Modi alternativi per scandire un vettore

```
int a[LUNG]= {.......};
int i, *p=a;
```

- I seguenti sono tutti modi equivalenti per stampare i valori di `a`

```
for (i=0; i<LUNG; i++)
     printf("%d", a[i]);

for (i=0; i<LUNG; i++)
    printf("%d", *(a+i));

for (p=a; p<a+LUNG; p++)
    printf("%d", *p);
```

```
for (i=0; i<LUNG; i++)
      printf("%d", p[i]);

for (i=0; i<LUNG; i++)
     printf("%d", *(p+i));
```

- Non è invece lecito un ciclo del tipo

```
for ( ; a<p+LUNG; a++)
    printf("%d", *a);
```

perché? Perché a++ è un assegnamento sul puntatore costante a!.
```

Differenza tra puntatori

- Il parallelo tra vettori e puntatori ci consente di capire il senso di un'operazione del tipo `p-q` dove `p` e `q` sono puntatori allo stesso tipo.

```
int *p, *q;
int a[10]={0};
int x;
...
x=p-q;
```

- Il valore di `x` è il numero di interi compresi tra l'indirizzo `p` e l'indirizzo `q`.
- Quindi se nel codice precedente `...` sono le istruzioni:

```
q = a;
p = &a[5];
```

il valore di `x` dopo l'assegnamento è 5.

Esempio

```
double b[10] = {0.0};
double *fp, *fq;
char   *cp, *cq;

fp = b+5;
fq = b;

cp = (char *) (b+5);
cq = (char *) b;

printf("fp=%p  cp=%p  fq=%p  cq=%p\n", fp, cp, fq, cq);
printf("fp-fq= %d,   cp-cq=%d\n", fp-fq, cp-cq);
```

```
fp=0x22fe3c  cp=0x22fe3c  fq=0x22fe14  cq=0x22fe14
fp-fq=5   cp-cq=40
```

Le Stringhe

- Una **stringa**, in informatica, è una sequenza di caratteri.
- In C una stringa viene rappresentata come un array contenente i suoi caratteri, seguiti dal *carattere speciale di fine stringa*: `'\0'`.
- **Esempi:**

  ```
  char stringa1[10] = {'p', 'i', 'p', 'p', 'o', '\0'};
  char non_stringa1[2] = {'p', 'i'};
  ```

  Il secondo array di caratteri non è una stringa (non termina con `'\0'`).
- Si può denotare una *costante* di tipo stringa (o *stringa letterale*) scrivendone i caratteri tra doppi apici (senza carattere di fine stringa).
  Esempio: `"pippo"`, `"corso di Informatica"`.
- Costanti di tipo stringa sono trattate come puntatori a caratteri.

  ```
  char *p = "abc";
  printf("%c%c\n", *p, *(p+1));
  ```

  provoca la stampa sullo schermo di `a` e `b`, poiché il puntatore `p` viene inizializzato con l'indirizzo del primo elemento dell'array *costante* di *4* caratteri utilizzato per la memorizzazione di `"abc"`.

Attenzione:

- non confondere costanti di tipo stringa e costanti di tipo carattere.
  $\Longrightarrow$ `"a"` e `'a'` sono diversi. Il primo è un array di 2 elementi, i cui valori sono il carattere `'a'` e il carattere speciale `'\0'`.
- Le stringhe costanti non possono essere modificate. **Esempio:**

  ```
  char *p = "abc";
  *p='x';
  printf("p: %s\n",p);
  ```

  Stampa: `p: abc`, quindi la stringa non viene modificata.
- I tre comandi che seguono sono del tutto equivalenti:

  ```
  char stringa[] = "abc";
  char stringa[4] = "abc";
  char stringa[] = {'a', 'b', 'c', '\0'};
  ```

- La stringa `"abc"` viene usata per inizializzare il vettore `stringa`. Quest'ultimo non è un vettore costante e può essere modificato
  $\Longrightarrow$ `*stringa = 'x'` (equivalente a `stringa[0]='x'`) è lecito.

Ingresso/uscita di stringhe

Stampa di una stringa

- Si deve utilizzare la specifica di formato "\%s". Stampa tutti i caratteri fino al primo '\0' escluso.
- **Esempio:**

```
char stringa[] = "abc";
char stringa1[] = {'a', '\0', 'b', 'c', '\0'};
char non_stringa[] = {'a', 'b'};
printf("%s\n", "pippo");
printf("%s\n", stringa);
printf("%s\n", stringa1);
printf("%s\n", non_stringa);
```

```
pippo
abc
a
ab?????...
```

Dove ????... sta per una sequenza di caratteri non significativi.

Lettura di una stringa

- Si deve utilizzare la specifica di formato "\%s".
- **Esempio:**

```
char buffer[40];
scanf("%s", buffer);
```

1. Vengono letti da input i caratteri in sequenza fino a trovare il primo carattere di spaziatura (spazio, tabulazione, interlinea, ecc.).
2. I caratteri letti vengono messi dentro il vettore buffer.
3. Al posto del carattere di spaziatura, viene messo il carattere '\0'.

- Note:
  - il vettore *deve* essere sufficientemente grande da contenere tutti i caratteri letti
  - non si usa \&buffer ma direttamente buffer (questo perché buffer è di tipo char*, ovvero è già un indirizzo)

Esempio:

```
char buffer[40];
printf("Digita una stringa di caratteri: ");
scanf("%s", buffer);
printf("Ecco la stringa memorizzata in buffer:  %s\n",
      buffer);
```

```
Digita una stringa di caratteri: abcdefghi
Ecco la stringa memorizzata in buffer:  abcdefghi
```

```
Digita una stringa di caratteri: abcde fghi
Ecco la stringa memorizzata in buffer:  abcde
```


Manipolazione di stringhe

- Per manipolare una stringa bisogna accedere ai singoli caratteri singolarmente (è un vettore come tutti gli altri!).
- **Esempio:**

```
for (i = 0; buffer[i] != '\0'; i++) {
/* fai qualcosa con buffer[i], ad esempio: */
buffer[i]='*';
}
```

- **Esempio:** Uguaglianza tra due stringhe.

```
typedef char String20[20];
String20 s1, s2;
...
if (s1==s2) ...
```

- N.B. *Non* si può usare "==" perché questo confronta i puntatori e non le stringhe.
  $\Longrightarrow$ si devono necessariamente scandire le due stringhe.