Lezione 12: Allocazione Dinamica della Memoria

Laboratorio di Elementi di Architettura e Sistemi Operativi

17 Aprile 2013

Puntatori e vettori

Puntatori ed indirizzi

- Semplificando, la memoria di un computer può essere vista come un *vettore di celle* numerate in modo consecutivo
  - tipicamente, le celle hanno la dimensione di un byte
- ogni tipo di dato occupa un certo numero di celle di memoria:
  - `char`: un byte
  - `short int`: due byte
  - `long int`: quattro o otto byte
  - ...
- un *puntatore* è un tipo di variabile che contiene un *indirizzo di memoria*
- Dichiarazione di un puntatore: `tipo *nome`
  - ogni puntatore è vincolato a puntare un certo tipo di dato
  - *eccezione:* un puntatore a `void` punta ad un dato generico
  - un puntatore a `void` *non può essere dereferenziato*
- L'operatore unario `&` fornisce l'indirizzo di un oggetto in memoria:
  - `p = &c;`
- L'operatore unario $\star$ *dereferenzia* un puntatore, ossia permette di *accedere al contenuto* dell'oggetto puntato:
  - `*p = 'a';  c = *p;`

Puntatori e vettori

- In C c'è un legame molto stretto tra puntatori e vettori:
  - *nome del vettore* = *puntatore al primo elemento*

Esempio:

```
int a[10], *pa;

pa = a;
```

- Indicizzare un vettore corrisponde ad incrementare il puntatore corrispondente:

`int a[10]`		`int *pa`		
`a`	`&a[0]`	`pa`	`&pa[0]`	puntatore al primo elemento
`&a[i]`	`(a+i)`	`&pa[i]`	`(pa+i)`	puntatore all'`i`-esimo elemento
`a[i]`	`*(a+i)`	`pa[i]`	`*(pa+i)`	valore dell'`i`-esimo elemento

- Ci sono però delle differenze:
  - un puntatore è una *variabile* che può essere modificata:
    - sia `pa = a` che `pa++` sono operazioni permesse
  - il nome di un vettore non è una variabile, e non si può modificare:
    - le operazioni `a = pa` e `a++` *non sono permesse!*
  - la definizione di un puntatore *non riserva spazio di memoria* per il contenuto

Puntatori e costanti stringa

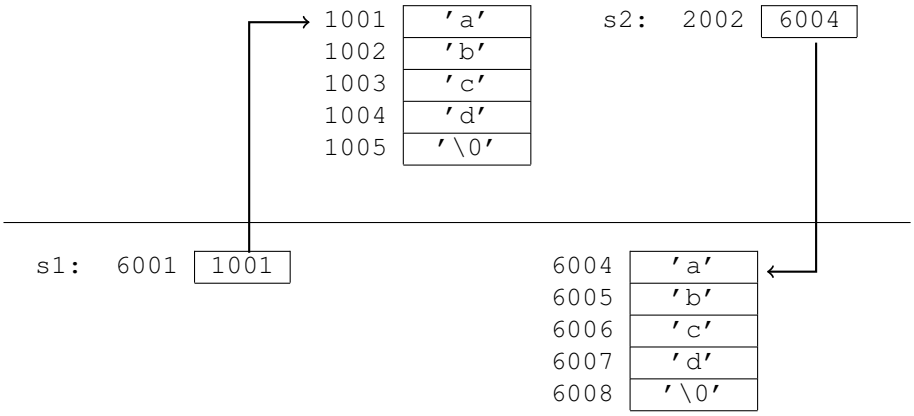
- Differenza sostanziale nel trattamento delle costanti stringa:

```
char s1[] = "abcd";
char *s2 = "abcd";
```

  - `s1` è un vettore:
    - i caratteri che lo compongono possono cambiare
    - si riferisce sempre alla stessa area di memoria
  - `s2` è un puntatore:
    - punta ad una stringa costante (`"abcd"`)
    - si può far puntare altrove (es. scrivendo `s2 = ...`) ma ...
    - *... la modifica del contenuto di `s2` ha risultato NON DEFINITO*

- *`s1` e `s2` puntano a "zone" di memoria diverse!!!*

Zona di memoria per le variabili



Zona di memoria per le costanti

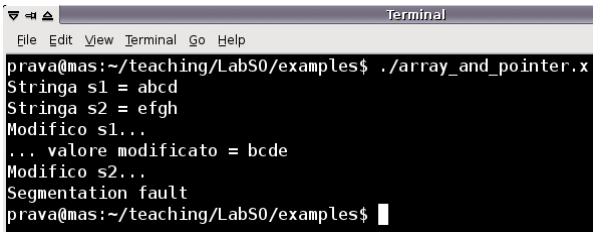
```
#include<stdio.h>

int main() {
  char s1[] = "abcd";
  char *s2 = "efgh";
  int i;

  printf("Stringa s1 = %s\n", s1);
  printf("Stringa s2 = %s\n", s2);

  printf("Modifico s1...\n");
  for(i=0;i<4;i++)
    s1[i] = s1[i]+1;
  printf("... valore modificato = %s\n", s1);

  printf("Modifico s2...\n");
  for(i=0;i<4;i++)
    s2[i] = s2[i]+1;
  printf("... valore modificato = %s\n", s2);
}
```



Vettori di puntatori e vettori multidimensionali

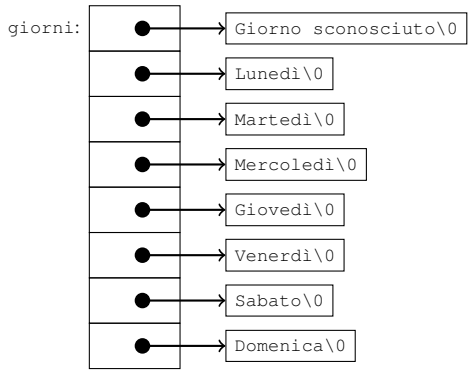
- I puntatori possono essere a loro volta memorizzati in vettori:
  - `char *line[MAXLINE];` crea un vettore di `MAXLINE` elementi, ognuno dei quali è un puntatore a carattere.
  - Se ogni `line[i]` punta al primo elemento di un ulteriore vettore di caratteri, il risultato complessivo sarà che il vettore `line` è un *vettore di righe di testo*.
- In C è possibile dichiarare vettori multidimensionali:
  - `int a[3][4];` dichiara una matrice di 3 righe e 4 colonne i cui elementi sono interi.
  - In C un vettore bidimensionale è trattato come un vettore (unidimensionale) di vettori.
  - Quindi l'espressione `a[i][j]` è corretta, mentre l'espressione `a[i,j]` non lo è.
- Le dichiarazioni

  ```
  int a[10][20];
  int *b[10];
  ```

  differiscono per i motivi seguenti:
  - nella prima dichiarazione vengono allocate le celle di memoria necessarie per contenere *200 interi* (10 x 20)
  - nella seconda vengono allocate le locazioni necessarie per contenere *10 puntatori ad interi*;
  - nel caso di `a` ogni elemento `a[i]` è un vettore di *lunghezza fissa* (20 elementi);
  - nel caso di `b` ogni elemento `b[i]` può puntare ad un vettore di *lunghezza arbitraria* (non necessariamente di 20 elementi);
    - l'inizializzazione degli elementi di `b` deve essere fatta esplicitamente!

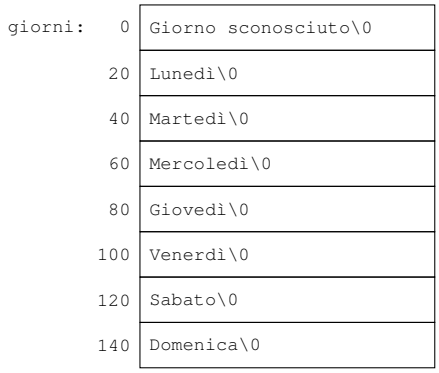
Esempio di vettore di puntatori

```
char *giorni[] = { "Giorno sconosciuto", "Lunedì",
                   "Martedì", "Mercoledì", "Giovedì",
                   "Venerdì", "Sabato", "Domenica"};
```



Esempio di vettore bidimensionale

```
char giorni[][20] = { "Giorno sconosciuto", "Lunedì",
                   "Martedì", "Mercoledì", "Giovedì",
                   "Venerdì", "Sabato", "Domenica"};
```



Allocazione dinamica della memoria

Memoria dinamica

- È possibile creare strutture dati allocate nella memoria dinamica del processo (*heap*)
- Allocazione al tempo di esecuzione tramite la funzione

  `void* malloc(int numero_di_byte)`

  - Ritorna `NULL` in caso di errore
  - Per allocare il tipo desiderato si usa l'operatore di cast `(tipo)`
- Richiede `#include <stdlib.h>`
- Due usi:
  - Allocazione dinamica di vettori e matrici

  - Allocazione dinamica di strutture

- È buona regola *liberare* la memoria allocata:
  - `free(<puntatore allocato con malloc>);`

Allocazione dinamica di vettori

- Bisogna dichiarare la dimensione del vettore
- Bisogna sapere quanto spazio (byte) occupa un elemento del vettore:
  - `sizeof(tipo)` ritorna il numero di byte occupati da `tipo`
- Esempi:
  - per allocare un vettore di 10 caratteri:

```
char* s;
s = (char*) malloc(10);
```

  - per allocare un vettore di n interi:

```
int* v;
int n = atoi(argv[1]);
v = (int*) malloc(n*sizeof(int));
```

Allocazione dinamica di matrici

- Allocazione statica

```
float matrice[15]   // 1 dimensione
float matrice[10][10]   // 2 dim
float matrice[12][16][19]    // 3 dim
```

- Allocazione dinamica

```
float *matrice;   // indip. dalle dimensioni
matrice=(float*)malloc(dim1*...*dimN*sizeof(float));
if (matrice == NULL)
{
  fprintf(stderr, "Out of memory\n");
   return -1;  //oppure exit(-1);
}
...
free(matrice);
```

- Accesso ad una matrice bidimensionale allocata staticamente

```
float matrice[10][10]   // 2 dim
...
matrice[i][j] = 3.14;
```

- Accesso ad una matrice bidimensionale allocata dinamicamente

```
float *matrice;
...
matrice = (float*) malloc(10*10*sizeof(float));
matrice[i*num_colonne+j] = 3.14;
```

Esempio: vettori dinamici

```
#include <stdlib.h>

int main() {
// 1: Dichiarazione del puntatore
  int *a;
  int i, n;
  printf("Inserire il numero di elementi:");
  scanf("%d", &n);

// 2: Allocazione della memoria
  a = (int *) malloc(n * sizeof(int));

// 3: controllo
  if(a == NULL)
    printerror("impossibile allocare la memoria", 1);

// 4: uso del vettore dinamico
  for(i = 1; i < n; i++) {
    printf("Inserire l'elemento numero %d:", i);
    scanf("%d", &a[i]);
  }
  ...
// 5: liberazione della memoria
  free(a); a = NULL;
  return 0;
}
```