# Laboratorio di Programmazione: Linguaggio C
# Lezione 21 del 19 maggio 2014

Damiano Macedonio

### Esercizio 1

Scrivere una funzione che riceve in ingresso un puntatore ad un array di interi e modifica tale array in modo che ogni posizione dispari `k` contenga la somma degli elementi che si trovano nelle posizioni dispari da `1` a `k`, e ogni posizione pari `j` contenga la somma degli elementi che si trovano nelle posizioni pari da `0` a `j`.

```
void transform_array(int *array_ptr, int n)
```

Il secondo parametro rappresenta la lunghezza dell'array e permette di rendere la funzione generica. Tutte le operazioni di accesso all'array devono essere eseguite tramite puntatori!

*Suggerimento.* Alla `i`-esima iterazione viene modificato l'elemento in posizione `i`, gli elementi precedenti sono già stati modificati e contengono già la somma dei loro predecessori (nelle posizioni dispari/pari).

La funzione è richiamata all'interno di un programma `main()` che si occupa anche di inizializzare l'array con l'input dell'utente e stampare il risultato. Scegliere a piacere la dimensione dell'array.

### Esercizio 2 [Tratto dalla prova parziale del 12 giugno 2013]

Le parole italiane non hanno mai più di due caratteri uguali di seguito. Si scriva una funzione

```
bool al_massimo_due_di_seguito_(char *s)
```

che determina se la stringa `s` contiene al massimo due caratteri uguali di seguito. In tal caso deve restituire `true`, altrimenti `false`.

Scrivere inoltre una funzione `main()` che si occupa anche di inizializzare la stringa con l'input dell'utente, chiama la funzione `al_massimo_due_di_seguito_()` e ne stampa il risultato.

### Esercizio 3 [Dizionario Italiano/Inglese]

Considerate il seguente header file `dictionary.h` come riferimento.

```
#ifndef DICTIONARY_H
#define DICTIONARY_H

#define MAX_ITEMS 10

struct item{
  char *it;
  char *en;
};

struct dictionary{
  struct item *items[MAX_ITEMS];
};

char *search(struct dictionary d, char *word, char language);

#endif
```

Scrivete un programma C (nel file `dictionary.c`) che implementa un dizionario italiano-inglese. Come descritto in `dictionary.h`, un dizionario (`dictionary`) è un array di strutture `item` ciascuna delle quali contiene due stringhe. Sulla struttura è definita una funzione `search()` che riceve in ingresso il dizionario, una parola da cercare e la lingua di partenza (`I`/`E`), e restituisce la corrispondente parola nell'altra lingua se esiste, oppure `NULL` altrimenti. Il termine del dizionario (ultimo elemento dell'array) è indentificato dal puntatore nullo `NULL`.

Definire inoltre una funzione `main()` che popola il dizionario, richiede all'utente la parola da tradurre e la lingua di partenza, chiama la funzione `search()` e infine stampa il risultato.

## Esercizio 4 [Tratto dalla prova parziale del 15 giugno 2012]

Si scrivano i file `polinomio.h` e `polinomio.c` che definiscono un polinomio a coefficienti interi in una sola variabile, cioè una cosa del tipo $-3x^4 + 9x^3 - 11x^2 - 2$. Si devono definire e implementare le seguenti funzioni:

- `struct polinomio *construct_polinomio(int coefficienti[], int grado)`, che costruisce un nuovo polinomio del grado indicato con i coefficienti indicati. Il grado è l'esponente del monomio più significativo. Si noti che il grado è sempre la lunghezza dell'array dei coefficienti meno 1. I coefficienti iniziano con quello del monomio più significativo. Per esempio, per costruire $-3x^4 + 9x^3 - 11x^2 - 2$ si deve poter scrivere `construct_polinomio(coefficienti, 4)` dove `int coefficienti[] = { -3, 9, -11, 0, -2 }`;
- `void destruct_polinomio(struct polinomio *this)`, che dealloca il polinomio indicato;
- `int grado(struct polinomio *this)`, che restituisce il grado del polinomio indicato;
- `struct polinomio *add(struct polinomio *this, struct polinomio *other)`, che restituisce un nuovo polinomio che è la somma dei due polinomi indicati;
- `int evaluate(struct polinomio *this, int x)`, che restituisce il valore del polinomio indicato valutato nel punto indicato;
- `char *toString(struct polinomio *this)`, che restituisce una nuova stringa che descrive il polinomio indicato, una cosa del tipo `- 3x^4 + 9x^3 - 11x^2 - 2x^0` (i monomi di coefficiente 0 non devono essere riportati).

*Suggerimento*: per realizzare la `toString`, può essere comodo usare la funzione

```
int sprintf(char *buffer, char *format, valori...)
```

della libreria standard `stdio.h`. Tale funzione si comporta come `void printf(char *format, valori...)` ma stampa dentro la stringa `buffer` invece che sul video. Inoltre `sprintf()` ritorna un `int` che è il numero di caratteri stampati. Si veda anche l'Esercizio 5 svolto alla fine del testo.

Se `polinomio.h` e `polinomio.c` sono definiti correttamente,il seguente programma:

```c
#include <stdio.h>
#include <stdlib.h>
#include "polinomio.h"

int main() {
  int coefficienti1[] = { -3, 4, 0, 0, -2 };
  int coefficienti2[] = { 5, -11, 0, 0 };

  struct polinomio *poly1 = construct_polinomio(coefficienti1, 4);
  struct polinomio *poly2 = construct_polinomio(coefficienti2, 3);
  struct polinomio *somma = add(poly1, poly2);

  char *s;

  printf("primo polinomio: %s di grado %d\n", s = toString(poly1),
      grado(poly1));
  free(s);

  printf("secondo polinomio: %s di grado %d\n", s = toString(poly2),
      grado(poly2));
  free(s);

  printf("somma: %s di grado %d\n", s = toString(somma), grado(somma));
  free(s);

  printf("la somma valutata in x = 7 vale %d\n", evaluate(somma, 7));

  destruct_polinomio(somma);
  destruct_polinomio(poly2);
  destruct_polinomio(poly1);

  return 0;
}
```

stamperà:

```
primo polinomio: - 3x^4 + 4x^3 - 2x^0 di grado 4
secondo polinomio: + 5x^3 - 11x^2 di grado 3
somma: - 3x^4 + 9x^3 - 11x^2 - 2x^0 di grado 4
la somma valutata in x = 7 vale -4657
```

### Esercizio 5 [Esercizio svolto: uso di `sprintf()`]

Costruiamo una funzione `char *intervallo_caratteri(char start, char end)` che restituisce una stringa formata da tutti i caratteri compresi tra il carattere `start` ed il carattere `end` con la corrispondente valore della codifica ASCII.

Ecco il codice

```
1  #include <stdio.h>  // per printf() e sprintf()
2  #include <stdlib.h> // per malloc() e free()
3
4  char *intervallo_caratteri(char start, char end) {
5          char c;
6          char *result = malloc(1000 * sizeof(char));
7          char *cursor = result;
8
9          for (c = start; c <= end; c++)
10                 // stampiamo la stringa che rappresenta il carattere corrente
11                 // e avanziamo cursor alla prossima posizione libera
12                 cursor += sprintf(cursor, "%3d : %c\n", c, c);
13
14         return result;
15 }
16
17 int main() {
18         char *s;
19         printf("%s", s = intervallo_caratteri('c', 'f'));
20         free(s);
21
22         return 0;
23 }
```

La sua esecuzione produce il seguente output:

```
 99 : c
100 : d
101 : e
102 : f
```

In particolare, dopo l'assegnamento di riga 19, la variabile `s` punta alla stringa

```
" 99 : c\n100 : d\n101 : e\n102 : f\n"
```