# Compito di Programmazione I - Bioinformatica

13 settembre 2024 (tempo disponibile: 2 ore)

## Esercizio 1 (15 punti)

**(si consegni `myatoi.c` e `myatoi.h`)**

Si completi `myatoi.c`, completando la funzione `myatoip(p,s)` in modo che restituisca l'intero `long` ottenuto facendo seguire al `long p` le cifre della stringa `s`, che si assume essere composta solo da caratteri tra il carattere dello `0` e il carattere del `9`, inclusi. La funzione `myatoip` deve essere ricorsiva. Per esempio, si dovrà avere:

```
myatoip(134, "3904") = 1343904
myatoip(1340, "3904") = 13403904
myatoip(22311, "0345895") = 223110345895
myatoip(0, "3458") = 3458
myatoip(13, "0") = 130
myatoip(13, "") = 13
```

Si completi quindi la funzione `myatoi(s)` in modo che restituisca l'intero `long` corrispondente alla stringa `s`, che si assume essere composta solo da caratteri tra il carattere dello `0` e il carattere del `9`, inclusi. La funzione `myatoi` deve essere implementata usando la funzione ausiliaria `myatoip`.

Si scriva il file `myatoi.h` che dichiara solo la funzione `myatoi`.

Se tutto è corretto, compilando insieme a `main_myatoi.c` (già scritto, da non modificare) ed eseguendo il risultato, verrà stampato:

```
myatoi("12345") = 12345
myatoi("012345") = 12345
myatoi("54321") = 54321
myatoi("543210") = 543210
myatoi("192837465") = 192837465
myatoi("23344556678026") = 23344556678026
myatoi("") = 0
```

**Suggerimento:** per intuire il meccanismo ricorsivo di `myatoip`, si noti per esempio che le seguenti uguaglianze sono vere:

```
myatoip(1340, "3904") = myatoip(13403, "904") = myatoip(134039, "04")
  = myatoip(1340390, "4") = myatoip(13403904, "") = 13403904
```

## Esercizio 2 (16 punti)

**(si consegni `tokenizer.c`)**

Un tokenizer è uno strumento di analisi del linguaggio naturale che divide una stringa in pezzetti (tokens) sulla base di un separatore o in base ad altri criteri.

Si modifichi opportunamente il file `tokenizer.c` in modo da costruire e stampare una lista di token a partire da una stringa che contiene i token separati da punto e virgola. In particolare, data in input una stringa del tipo *"Alessandra;Jessica;Kaur;Francesca;Vittoria"* si deve costruire una lista di 5 nodi, esemplificata in Figura 1. Si assuma che ogni token sia lungo al più `MAX_LENGTH-1` caratteri. Si noti che nell'esempio i token son inseriti sempre in testa alla lista; nell'implementazione si può scegliere liberamente se inserirli in testa o in coda.

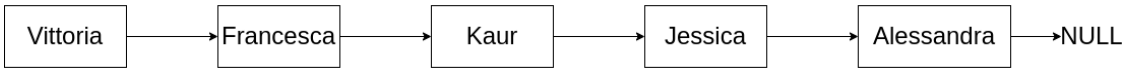


Figure 1: Esempio

**La funzione main è già scritta e completa, non va modificata. Si possono aggiungere funzioni ausiliare in `tokenizer.c`. Attenzione, NON si può usare la funzione di libreria `strtok` di `string.h`.**

```c
#include <stdio.h>
#define MAX_LENGTH 50
#define MAX_STRING_LENGTH 300
struct Nodo{
    //definizione di struttura da completare
};

struct Nodo * mystrtok(char *s){
//da completare
}

void print_list(struct Nodo *l) {
//da completare
}

int caratteri_validi(char * elenco){
//da completare
}

int main(void){
  char elenco_nomi[MAX_STRING_LENGTH];//ESEMPIO "Maria;Jessica;Kaur;Michela;
     Marco;Angelica;"
  int len = -1;

  do{
    printf("Inserisci un elenco di nomi, separati dal punto e virgola (;).
       La lunghezza complessiva deve essere inferiore a %d caratteri: ",
       MAX_STRING_LENGTH);
    scanf("%s",elenco_nomi);

    len = strlen(elenco_nomi);

    if(!caratteri_validi(elenco_nomi) || len >= MAX_STRING_LENGTH) {
      printf("Errore nell'input, riprovare\n");
    }
  } while(!caratteri_validi(elenco_nomi) || len >= MAX_STRING_LENGTH);

  struct Nodo* L=mystrtok(elenco_nomi);
  print_list(L);
  return 0;
}
```

In particolare si completino le funzioni:

- `mystrtok` che riceve in input una stringa formata da parole separate da punto e virgola e crea una lista dove ogni nodo contiene una di quelle parole. Quando si implementa la funzione `mystrtok`, SI DEVONO utilizzare i puntatori per scorrere la stringa ed estrarre i token. *BONUS: Se un token supera la lunghezza massima* ***`MAX_LENGTH`****, i caratteri in eccesso devono essere ignorati.*
- `print_list` che stampa una lista di token, stampando un token per ogni riga.
- `caratteri_validi` che riceve in input una stringa e ritorna 1 se la stringa è formata da caratteri alfabetici ([A-Z] e [a-z]) e/o dal punto e virgola (';'), 0 altrimenti.

Se tutto è corretto, un esempio di esecuzione di `tokenizer.c`, specificando in input la stringa *Elena;Vittoria;Maria* è:

```
Maria
Vittoria
Elena
```

Si noti che anche ; è un input valido.