# Fondamenti di Informatica T-1 (A.A. 2012/2013)
# Ingegneria Informatica

## Prof.ssa Mello

Prova parziale d'esame di Giovedì 31 Gennaio 2013 – durata 1h
Totale 12 punti, sufficienza con 7

**Compito A**

## Esercizio 1 (6 punti)

Data una lista di reali `numbers` e due interi `min` e `max` tali che `0` $\leq$ `min` $<$ `max`, si realizzi una funzione ***iterativa***

```
list choose(list numbers, int min, int max);
```

che restituisca una nuova lista contenente gli elementi presenti in `numbers` a partire dalla posizione `min` fino alla posizione `max` incluse. L'ordine degli elementi in questa nuova lista è ininfluente. Se `min` o `max` eccedono la dimensione di `numbers`, la funzione considera solo i numeri che riesce a leggere.

Ad esempio, se `numbers = [1.0, 2.7182, 3.1415, 9.8066]`, `min = 2` e `max = 10`, `choose(...)` restituisce la lista `[9.8066, 3.1415]`. Se invece `min = 5` e `max = 10`, restituisce una lista vuota.

Per implementare `choose(...)` si usino le sole primitive dell'ADT `list`. Ogni altra funzione che il candidato desideri utilizzare dovrà essere opportunamente specificata. Si realizzi infine un semplice `main()` di prova che invochi correttamente la funzione `choose(...)` appena creata.

## Esercizio 2 (2 punti)

Si consideri la seguente grammatica $G = \langle V_T, V_N, P, S \rangle$ con:

- *simboli terminali* $V_T = \{$'1','2','a','b','–','+'$\}$
- *simboli non-terminali* $V_N = \{S, A, B, X, Y, Z\}$
- *produzioni* $P = \{$
  - $S ::= X \mid Y\ T \mid Z\ S$
  - $T ::= Y\ S \mid Y$
  - $X ::=$ '1' | '2'
  - $Y ::=$ 'a' | 'b'
  - $Z ::=$ '–' | '+'

  $\}$
- *scopo* $S$

La stringa 'ab+2' appartiene al linguaggio generato da tale grammatica? In caso affermativo se ne mostri la *derivazione left-most*. Inoltre si disegni il *diagramma sintattico* per la produzione $T$.

# Esercizio 3 (3 punti)

Si consideri il seguente programma C: compila correttamente? In caso affermativo, si dica quali valori sono stampati a tempo d'esecuzione motivando opportunamente la risposta.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SEP '+'

char* analyse(char* s, int* l) {
  int i = 0, j = strlen(s), k = j;
  char* res = (char*) malloc(sizeof(char) * k);
  *l = 0;
  do {
    while (i < k && s[i] == SEP)   i++;
    while (j > 0 && s[j-1] == SEP)   j--;
    if (j > 0) {
      res[*l] = (s[i] == s[j-1]) ? s[i] : '*';
      i++; j--; (*l)++;
    }
  } while(i <= j);
  res[*l] = '\0';
  return res;
}

int main() {
  int i, dim; char* temp;
  char sentence[] = "+++Parlo+col+RAP";
  temp = analyse(sentence, &dim);
  for (i = 0; i < dim; i++)
    printf("%c", temp[i]);
  free(temp);
  return 0;
}
```

# Esercizio 4 (1 punto)

Si descrivano brevemente le differenze tra *allocazione statica* ed *allocazione dinamica della memoria*, avendo cura di fornire un breve esempio in cui tali tecniche sono messe a confronto.

# Soluzioni

## Compito A

# Esercizio 1

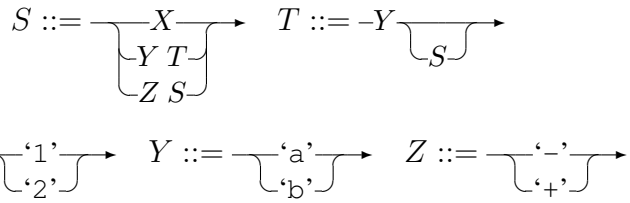
```
1  // Funzione iterativa
2  list choose(list numbers, int min, int max) {
3    int i;
4    list result = emptylist();
5    for (i = 0; !empty(numbers) && i <= max; i++) {
6      if (i >= min)
7        result = cons(head(numbers), result);
8      numbers = tail(numbers);
9    }
10   return result;
11 }
12
13 // Funzione main di prova
14 int main() {
15   list result, numbers = emptylist();
16   numbers = cons(3.1415, cons(9.8066, numbers));
17   numbers = cons(1.0, cons(2.7182, numbers));
18   result = choose(numbers, 2, 10);
19   while (! empty(result)) {
20     printf("%f ", head(res));
21     result = tail(result);
22   }
23   return 0;
24 }
```

# Esercizio 2

La frase ‘ab+2’ appartiene al linguaggio generato dalla grammatica $G$. In particolare, è possibile ottenere tale frase mediante la seguente derivazione left-most:

$S ::= YT \rightarrow$ ‘a’ $T \rightarrow$ ‘a’ $Y\ S \rightarrow$ ‘ab’ $S \rightarrow$ ‘ab’ $Z\ S \rightarrow$ ‘ab+’ $S \rightarrow$ ‘ab+’ $X \rightarrow$ ‘ab+2’

Di seguito sono riportati i diagrammi sintattici di tutte le produzioni della grammatica $G$:



# Esercizio 3

Il programma è sintatticamente corretto e produce la stampa:

```
P**loc
```

La funzione `main()` introduce le variabili intere `i` e `dim`, le stringhe `temp` e `sentence`, inizializzando quest'ultima a `"+++Parlo+col+RAP"`. Quindi chiama la funzione `analyse(...)` che accetta due parametri: la stringa `sentence` e `dim` (per riferimento).

La funzione `analyse(...)` introduce a sua volta quattro variabili locali: gli interi `i`, `j` e `k` (rispettivamente inizializzati a `0`, `strlen(s)` e `strlen(s)`), e la stringa `res` (per cui viene allocata una quantità di memoria pari a quella occupata da `s`). Si noti che `i` è l'indice per scandire la stringa `s` in avanti, `j` è l'indice per scandire la stringa `s` all'indietro e `k` memorizza la dimensione di `s`. Il parametro `*l` viene impostato a `0`.

`analyse(...)` consiste sostanzialmente in un ciclo **`do while`** che viene ripetuto fintantoché l'indice `i` è minore o uguale a `j`. Al suo interno si trovano due cicli **`while`** simili tra di loro. Il primo incrementa l'indice `i` fino al primo carattere che non sia il separatore `SEP` (`'+'`) o al termine della stringa, mentre il secondo decrementa `j` fino al primo carattere che non sia `SEP` (`'+'`) o all'inizio della stringa.

Se al termine di queste operazioni gli indici puntano ancora a legittime posizioni di `s`, `analyse(...)` scrive alla posizione `*l` di `res` il carattere '`*`' se i caratteri puntati da `i` e `j` sono diversi tra di loro, o il carattere stesso se invece sono uguali. Quindi l'indice `i` viene incrementato, `j` decrementato e `*l` aggiornato (incrementato).

Al termine del ciclo **`do while`**, prima di restituire la stringa `res` al chiamante, la funzione `analyse(...)` aggiunge il terminatore di fine stringa `'\0'` per garantire che la stessa stringa `res` sia ben formata.

La funzione `main()` provvede con un apposito ciclo **`for`** a stampare il contenuto della stringa `res` assegnata alla variabile `temp` un carattere alla volta. Il `main()` quindi libera la memoria allocata per `temp` da `analyse(...)` e termina la propria esecuzione restituendo `0`.

# Esercizio 4

Vedi slides del corso.