Corso di Informatica@Matematica
Prova scritta del 27 Giugno 2019

Tempo a disposizione: ore 2.

In tutti gli esercizi è possibile definire funzioni ausiliarie. Fare gli esercizi **(1, 2) e (3, 4, 5)** su due fogli differenti. Il voto massimo è **24** la sufficienza è **13** . Il **voto finale** dell'esame si ottiene sommando i due voti sufficienti di scritto e laboratorio (studenti di matematica).

1. (5 punti) Si consideri il seguente programma

```
def f(S):
    a=[]
    for x in S:
        a = a+x
    return a
```

   (i) Si tratta di un programma sintatticamente corretto? (ii) Cosa avviene chiamando la funzione `f` come `f([[1],[3]])`? (iii) Cosa avviene chiamando la funzione `f` come `f([2,4])`? (iii) Cosa avviene chiamando la funzione `f` come `f([[a],[a]])`?

2. (5 punti) La studentessa Alice dice di aver scoperto il linguaggio di programmazione A1 che permette di fare molte cose, fra cui la moltiplicazione di matrici di dimensione arbitraria, e che fornisce anche un un tool che, dato in input un generico programma P scritto in A1, decide se P termina oppure no. Lo studente Bob dice di aver scoperto un linguaggio B1 che permette di scrivere programmi che non saranno mai traducibili in programmi equivalenti scritti in A1. Il computer Charlie dice che l'affermazione di Bob è falsa perché, dati due generici linguaggi X e Y, è sempre possibile realizzare un compilatore da X a Y. Chi dei tre dice la verità e chi no? Motivare la risposta.

3. (4 punti) Quanti e quale insieme di interi distinti si possono rappresentare in modulo e segno su un byte? Ed in complemento a 2?
   Si consideri la seguente configurazione di bit `10011001`.
   (i) Quale numero rappresenta in complemento a due su 8 bit?
   (ii) Quale numero rappresenta in modulo e segno su 8 bit?

4. (5 punti) Cosa stampa il seguente programma Python? Si riporti anche l'esecuzione.

```
class A:
    def __init__(self, xx, yy):
        self._x = xx
        self._y = yy

    def g(self):
        self._y = self._x-10
        return self._y


class B(A):
    def __init__(self, xx, yy, zz):
        super().__init__(xx, yy)
        self.__x = self._x*2
        self.__y = self._y + zz

    def f(self,delta):
        self.__y = self.g() ** delta

    def __str__(self):
        return str(CC.__x) + " " + str(CC.__y)

CC = B(7, 5, 3)
CC.f(2)
print(CC)
```

5. (5 punti) Si supponga di avere a disposizione la definizione della classe Python `tree` (vista a lezione) che introduce gli alberi binari etichettati mediante i seguenti costruttori e metodi (`t` è un albero):

```
tree()              # Restituisce l'albero vuoto
tree(a)             # Restituisce l'albero con una sola foglia/radice etichettata a
tree(a,l,r)         # Restituisce l'albero con radice etichettata a,
                    # figlio sinistro l e figlio destro r
t.label()           # Restituisce l'etichetta sulla radice dell'albero t
t.is_empty()        # Restituisce True sse t \'e l'albero vuoto
t.first_child():    # Restituisce il figlio sinistro dell'albero t
t.second_child():   # Restituisce il figlio destro dell'albero t
t.is_leaf():        # Restituisce True sse t \'e una foglia
```

Si scriva una funzione Python `swaptree(t)` che restituisce un nuovo albero nel quale dato un nodo il figlio sinistro ha sempre valore inferiore al figlio destro. Per semplicità, si assuma che l'albero può contenere solo numeri naturali.