# Esame di Laboratorio di Fondamenti di Informatica II e Lab. del 10/02/2020

## Note importanti:

- È considerato errore qualsiasi output non richiesto dagli esercizi.
- È consentito utilizzare funzioni ausiliarie per risolvere gli esercizi.
- Quando caricate il codice sul sistema assicuratevi che siano presenti tutte le direttive di include necessarie, comprese quelle per l'utilizzo delle primitive. Non dovete caricare l'implementazione delle primitive.
- È importante sviluppare il codice in Visual Studio, prima del caricamento sul sistema, per poter effettuare il debug delle funzioni realizzate!

## Esercizio 1 (6 punti)

Dato un numero $n$ positivo la sequenza di *Hailstone*, definita dal matematico tedesco Lothar Collatz, si ottiene a partire da $n$ procedendo come segue:

1. Se `n` è pari il prossimo elemento della sequenza si ottiene facendo `n/2`
2. Se `n` è dispari l'elemento successivo è dato da `3*n + 1`

Il procedimento si ripete per il nuovo `n`. Ad esempio, la sequenza ottenuta a partire da `n = 5` è la seguente: `5, 16, 8, 4, 2, 1, 4, 2, 1,...`. O ancora, se `n = 11` abbiamo: `11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1,...`. Come si può notare, una volta arrivati ad `1` la sequenza si ripete all'infinito.

Nel file `hailstone.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern size_t Hailstone(int n);
```

La funzione prende in input un numero `n` e deve stampare a video (`stdout`) la sequenza di *Hailstone* costruita **ricorsivamente** a partire da `n`. La funzione deve interrompersi quando viene incontrato il primo `1` della sequenza e deve ritornare il numero di elementi che compongono la sequenza stessa.

Il formato dell'output deve essere il seguente: ogni elemento fatta eccezione per l'ultimo, deve essere seguito dai caratteri `<virgola>` e `<spazio>`.

Se il valore `n` passato alla funzione è minore o uguale di zero la funzione non deve stampare nulla e deve ritornare 0.

## Esercizio 2 (7 punti)

Nel file `sortlist.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern list SortList(list l);
```

La funzione prende in input una lista di `int` e deve ordinarne i suoi elementi per valore crescente e ritornare la testa della lista ordinata. Se la lista `l` è vuota deve essere ritornata una lista vuota. Non ci sono vincoli sull'algoritmo di ordinamento, ovvero potete scegliere l'algoritmo che preferite. Suggerimento: per scambiare due elementi della lista è sufficiente scambiarne i valori, senza dover aggiornare i puntatori.

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```
typedef int element;
typedef struct list_element {
```

```
  element value;
  struct list_element *next;
} item;
typedef item* list;
```

e le seguenti primitive:

```
list EmptyList();
list Cons(const element *e, list l);

bool IsEmpty(list l);
element Head(list l);
list Tail(list l);
element Copy(const element *e);
void FreeList(list l);
list InsertBack(list l, const element *e);
```

trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `list_int.h` e `list_int.c`. La documentazione delle primitive sopraelencate è disponibile al link:

http://imagelab.ing.unimore.it/OLJ2/esami/materiale/20201002_esame/list_int/list__int_8h.html

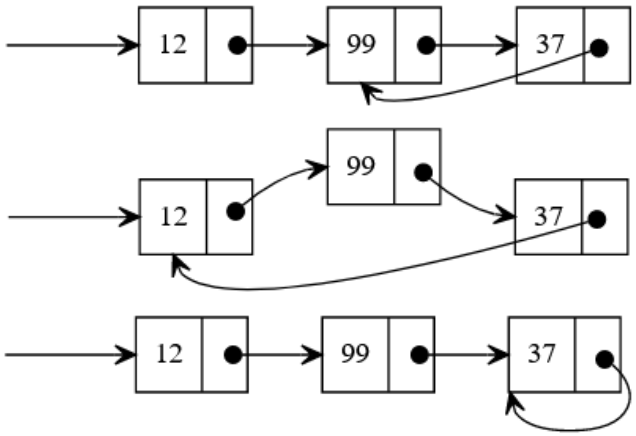
## Esercizio 3 (8 punti)

Nel file `any_loop.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern bool AnyLoop(list l);
```

La funzione prende in input una lista di `int` e deve verificare se questa contiene o meno un ciclo. La funzione deve ritornare `true` se nella lista è presente un ciclo, `false` altrimenti (compreso il caso in cui la lista è vuota).

Una lista contiene un ciclo se almeno uno dei suoi elementi è raggiungibile da due percorsi differenti. Di seguito alcuni esempi di liste contenenti cicli:




Per la risoluzione di questo esercizio avete a disposizione le stesse definizioni e primitive dell'esercizio precedente.

## Esercizio 4 (5 punti)

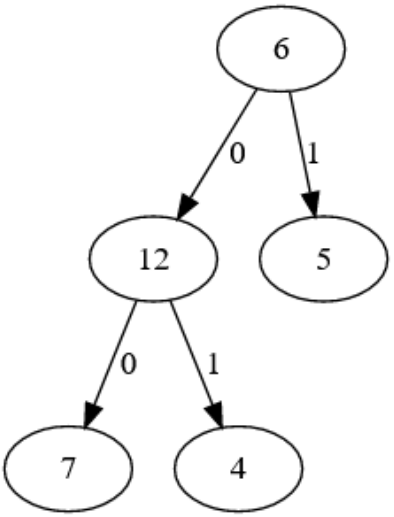
Nel file `preorder.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern int PreOrder(tree t);
```

La funzione prende in input un albero di `int` che deve essere attraversato in pre-ordine e stampato a video (`stdout`). La funzione deve ritornare la somma delle chiavi contenute nell'albero. Il formato dell'output deve essere il seguente:

```
key1 <spazio> key2 <spazio> key3 <spazio> ...
```

Sia dato ad esempio l'albero seguente:




La funzione dovrebbe produrre l'output `6 12 7 4 5` e ritornare 34.

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```
typedef int element;
typedef struct tree_element {
element value;
    struct tree_element *left, *right;
} node;
typedef node* tree;
```

e le seguenti primitive:

```
tree EmptyTree();
tree ConsTree(const element *e, tree l, tree r);
bool IsEmpty(tree t);
element *GetRoot(tree t);
tree Left(tree t);
tree Right(tree t);
bool IsLeaf(tree t);
tree InsertBinOrd(const element *e, tree t);
```

trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `tree_int.h` e `tree_int.c`. La documentazione delle primitive sopraelencate è disponibile al link:

http://imagelab.ing.unimore.it/olj2/esami/materiale/20201002_esame/tree_int/tree__int_8h.html

## Esercizio 5 (7 punti)

Nel file `stringhe.c` definire la procedura corrispondente alla seguente dichiarazione:

```
extern void BacktrackStr(int n);
```

la procedura prende in input un `int` e deve stampare a video (`stdout`), utilizzando un approccio di **backtracking**, tutte le possibili stringhe di lunghezza `n` che è possibile costruire utilizzando le prime `n` lettere minuscole dell'alfabeto inglese. L'output dovrà avere il seguente formato:

```
{ xx…x }, { xx…x }, { xx…x }, … { xx…x },
```

Se `n <= 0` o `n > 26` la funzione non deve stampare nulla. Le stringhe devono essere stampate in ordine alfabetico.

Si consiglia di utilizzare una funzione ausiliaria per la risoluzione dell'esercizio. N.B. Non sarà considerata valida nessuna soluzione che non faccia uso del backtracking.

Esempio:

```
 input: n = 2
output: { aa }, { ab }, { ba }, { bb },
```