

	Politecnico di Milano Facoltà di Ingegneria Industriale INFORMATICA B Appello 29 giugno 2015		COGNOME E NOME				
	RIGA	COLONNA	MATRICOLA				
			Spazio riservato ai docenti <table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td style="width: 20px; height: 20px;"></td> <td style="width: 20px; height: 20px;"></td> <td style="width: 20px; height: 20px;"></td> <td style="width: 20px; height: 20px;"></td> </tr> </table>				

- Il presente plico contiene 3 esercizi e **deve essere debitamente compilato con cognome e nome, numero di matricola.**
- Il tempo a disposizione è di 1 ora e 30 minuti.
- Non separate questi fogli. Scrivete la soluzione solo sui fogli distribuiti, utilizzando il retro delle pagine in caso di necessità. Cancellate le parti di brutta (o ripudiate) con un tratto di penna.
- Ogni parte non cancellata a penna sarà considerata parte integrante della soluzione.
- È possibile scrivere a matita (e non occorre ricalcare al momento della consegna!).
- **È vietato utilizzare calcolatrici, telefoni o pc.** Chi tenti di farlo vedrà annullata la sua prova.
- È ammessa la consultazione di libri e appunti, purché con pacata discrezione e senza disturbare.
- **Qualsiasi tentativo di comunicare con altri studenti comporta l'espulsione dall'aula.**
- È possibile ritirarsi senza penalità.
- **Non è possibile lasciare l'aula conservando il tema della prova in corso.**
- L'esame orale è parte integrante dell'esame e deve essere realizzato almeno sufficientemente per il superamento dell'esame complessivo.
- Per accedere all'esame orale è necessario svolgere almeno sufficientemente gli esercizi 1 e 2.

Esercizio 1 (10 punti)

Si considerino i seguenti tipi di dato in linguaggio C. Essi servono a contenere le informazioni relative a una carta di credito, inclusa la registrazione delle transazioni eseguite.

```
#define N 100
typedef char Stringa[N];
typedef enum {falso, vero} bool;

typedef struct // descrizione della transazione
{
    float importo;
    Stringa nazione;
    int timestamp; // momento dell'acquisto espresso in secondi dalle ore 00:00:00 del 1/1/1970.
    // Es. le ore 9:00:00 del 2015.06.29 corrispondono al 1435561200 secondo
    //     le ore 9:01:00 del 2015.06.29 corrispondono al 1435561260 secondo
    bool usato_pin; // determina se la transazione è avvenuta richiedendo il PIN all'utente
} Acquisto;

typedef struct
{
    int card_number; // numero della carta di credito
    Acquisto trans[N];
    int n_trans; // numero delle transazioni eseguite
} Carta;
```

- A)** Si scriva un frammento di codice in linguaggio C per rilevare le carte di credito che possono aver subito una frode. Una carta di credito può aver subito una frode se:
- registra due transazioni in meno di 1 minuto,
- oppure
- registra due transazioni che richiedono il PIN, in nazioni diverse, in meno di un'ora.

In particolare:

- Si dichiari una variabile *cards* in grado di contenere 1 milione di carte di credito.
 - Si assuma che la variabile *cards* sia stata interamente popolata. Si scriva un frammento di codice in linguaggio C in cui si scorre *cards* per identificare i numeri delle carte di credito che possono aver subito una frode. Dopo aver eseguito tali controlli, si stampino a schermo i numeri delle carte di credito che possono aver subito una frode. Ogni numero deve essere stampato una sola volta. Si assuma che le transazioni all'interno di *trans* siano state registrate in ordine cronologico, dalla prima all'ultima. Si dichiarino tutte le variabili necessarie per lo svolgimento dell'esercizio.
- B)** Si definisca un nuovo tipo di variabile *Persona*, atta a contenere le informazioni relative al proprietario della carta di credito, e la si utilizzi per modificare la definizione del tipo *Carta* in modo che contenga anche le informazioni relative al proprietario. Si indichi inoltre quali modifiche apportare al frammento di codice sviluppato al punto 2 per fare in modo che le informazioni di tutte le persone che hanno subito una frode vengano copiate (senza lasciare buchi) in un array *persone_frodate*.

SOLUZIONE

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

typedef struct
{
    Stringa nome;
    Stringa cognome;
} Persona; // MODIFICA PER punto B

typedef struct
{
    int card_number; // numero della carta di credito
    Acquisto trans[N];
    Persona proprietario; // MODIFICA PER punto B
    int n_trans; // numero delle transazioni eseguite
} Carta;

#define M 1000000
int main()
{
    Carta cards[M];
    int i, j, frodata;
    Persona persone_frodate[M]; // MODIFICA PER punto B

    for(i = 0; i < M; i++)
    {
        frodata = 0;
        k = 0; // MODIFICA PER punto B
        for(j = 0; (j < cards[i].n_trans - 1) && frodata == 0; j++)
        {
            if(cards[i].trans[j+1].timestamp - cards[i].trans[j].timestamp < 60)
            {
                frodata = 1;
            }
            else
            {
                if(cards[i].trans[j+1].usato_pin == vero &&
                    cards[i].trans[j].usato_pin == vero &&
                    cards[i].trans[j+1].timestamp - cards[i].trans[j].timestamp < 60 * 60 &&
                    strcmp(cards[i].trans[j+1].nazione, cards[i].trans[j].nazione) != 0)
                {
                    frodata = 1;
                }
            }
        }

        if (frodata)
        {
            printf("\nLa carta di credito %d è stata frodata ", cards[i].card_number);
            // MODIFICHE PER punto B
            persone_frodate[k] = cards[i].proprietario;
            k++;
        }
    }
}
```

Esercizio 2 (10 punti)

Si sviluppino in Matlab le seguenti funzioni, sfruttando le proprietà di Matlab per la gestione degli array.

Funzione n. 1

Si definisca la funzione *rimuovi(A,v)* che prenda in ingresso un vettore riga *A* e un valore *v*, e restituisca il vettore *A* privato di *v* (qualora *A* contenga *v*), altrimenti restituisca *A*.

Es: `rimuovi([5, 6, 7], 5)` deve dare --> `[6, 7]`
`rimuovi([5, 6, 7], 9)` deve dare --> `[5, 6, 7]`

SOLUZIONE

```
function B = rimuovi(A, v)
    B = A(find(A ~= v));
end
```

Funzione n. 2

Si definisca la funzione *aggiungi(Q,v)* che prenda in ingresso una matrice *Q* e un valore *v*, e restituisca una matrice corrispondente a *Q* a cui è stato aggiunto *v* come primo elemento di ogni riga.

Es: `aggiungi([5, 4; 6, 8; 7, 9], 3)` deve dare --> `[3, 5, 4; 3, 6, 8; 3, 7, 9]`

SOLUZIONE

```
function T = aggiungi(Q, v)
    [r, c] = size(Q);
    temp = v .* ones(r, 1);
    T = [temp, Q];
end
```

Funzione n. 3

Si supponga di avere un vettore riga *S* contenente *n* numeri diversi:

Es: `S = [5, 8, 1]`

Si rappresenti una **disposizione semplice** di lunghezza *k* (con $k \leq n$) con una matrice di *k* colonne che contenga, in ogni riga, un sottoinsieme di *k* elementi di *S* tale che: a) le righe della matrice siano tutte diverse tra loro, b) all'interno di ogni riga non vi siano ripetizioni di uno stesso elemento, c) due righe possano contenere gli stessi elementi purché in ordine diverso.

Ad esempio, la disposizione semplice di *S* con $k = 2$ è la matrice seguente¹:

`[5, 8; 5, 1; 8, 5; 8, 1; 1, 5; 1, 8]`

Si scriva in linguaggio Matlab/Octave una funzione **ricorsiva** *disposizioni(S, k)* che implementi il calcolo delle disposizioni semplici di *k* elementi di un insieme *S*, ad esempio:

```
octave> disposizioni([5,8,1], 2)
```

```
ans =
```

¹ Ricordate che in una matrice Matlab/Octave, il punto e virgola separa le righe.

5 8
 5 1
 8 5
 8 1
 1 5
 1 8

Osservazioni utili per l'implementazione

Quando $k = 1$, le disposizioni semplici di S corrispondono a S trasposto, ad esempio:

>> disposizioni([5,8,1], 1)

ans =

5
 8
 1

Quando $k > 1$, le disposizioni semplici si possono calcolare usando le funzioni *rimuovi(A,v)* e *aggiungi(Q,v)*, definite ai punti n. 1 e n. 2 precedenti, nel modo seguente:

1. si rimuove un elemento v di S
2. si aggiunge v come primo elemento di ogni riga di ognuna delle disposizioni di $k - 1$ elementi di $S - v$ (ovvero del vettore S a cui è stato rimosso v)

Tale procedura deve essere ripetuta per tutti gli elementi v di S .

Applicando la procedura descritta al nostro esempio, per $k = 2$ si devono prima di tutto calcolare, per ogni v , le disposizioni di $k = 1$ elementi di $S - v$:

S	k	v	S - v	disposizioni(S-v, 1)
[5,8,1]	2	5	[8, 1]	[8; 1]
[5,8,1]	2	8	[5, 1]	[5; 1]
[5,8,1]	2	1	[5, 8]	[5; 8]

e poi aggiungere come primo elemento di ogni riga, a ciascuna delle disposizioni ottenute, l'elemento v tolto in precedenza:

v	disposizioni(S-v, 1)	disposizioni risultanti
5	[8;1]	[5, 8; 5, 1]
8	[5;1]	[8, 5; 8, 1]
1	[5;8]	[1, 5; 1, 8]

Il risultato finale è la fusione delle disposizioni così trovate in un'unica matrice:

[5, 8; 5, 1; 8, 5; 8, 1; 1, 5; 1, 8]

SOLUZIONE

```
function P = disposizioni(s, k)
    n = length(s);
    if k == 1
        P = s';
    else
        P = [];
        for x = 1 : n
            v = s(x);
            t = rimuovi(s, v);
            Q = disposizioni(t, k-1);
            T = aggiungi(Q, v);
            P = [P; T];
        end
    end
end
```

Esercizio 3 (6 punti)

Sia dato il seguente frammento di codice in linguaggio C:

```
if(!a) {  
    if(b) {  
        if((a && c) || (!b && !c)) {  
            printf("Nooo!");  
        } else {  
            printf("Yes!");  
        }  
    }  
}
```

dove a, b e c sono variabili intere inizializzate in precedenza a un valore che può essere 0 oppure 1.

1) Si compili la seguente tabella per ogni combinazione dei valori delle tre variabili a, b e c:

a	b	c	Messaggio stampato
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

2) Si riscriva il frammento di codice in linguaggio C utilizzando un solo if, ove la condizione deve essere la più ridotta possibile in termini di operatori e variabili utilizzate.

SOLUZIONE

1)

a	b	c	Messaggio stampato
0	0	0	
0	0	1	
0	1	0	Yes!
0	1	1	Yes!
1	0	0	
1	0	1	
1	1	0	
1	1	1	

2)

```
if(!a && b) {  
    printf("Yes!");  
}
```