Informatica A – Seconda Prova in Itinere

**Cognome Matricola**

**Nome**

Istruzioni

1. 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**.
2. Ogni parte non cancellata a penna sarà considerata parte integrante della soluzione.
3. **È possibile scrivere a matita** (e non ricalcare al momento della consegna!).
4. È **vietato** utilizzare **calcolatrici** o **telefoni**. Chi tenti di farlo vedrà **annullata** la sua prova.
5. È ammessa la consultazione di **libri** e **appunti**, purché con pacata discrezione e senza disturbare.
6. Qualsiasi **tentativo** di comunicare con altri studenti comporta **l’espulsione** dall’aula.
7. È possibile **ritirarsi senza penalità**.
8. Non è possibile lasciare l’aula conservando il tema della prova in corso.
9. Tempo a disposizione:
10. **1h:00m per recupero prima prova**
11. **2h:15m per seconda prova**

**RECUPERO PRIMA PROVA**

**Esercizio 1.1 ( 7 punti )**

**Esercizio 1.2 ( 7 punti )**

**SECONDA PROVA**

**Esercizio 2.1 ( 4 punti )**

**Esercizio 2.2 ( 4 punti )**

**Esercizio 2.3 ( 6 punti )**

**Esercizio 2.4 ( 4 punti )**

**Voto finale:**

Esercizio 1.1 ( 7 punti ) – RECUPERO PRIMA PROVA

In un festival della canzone si votano le canzoni da casa in più giornate. Le strutture dati utilizzate sono le seguenti:

#define N 100

typedef struct { int giorno, mese, anno; } Data;

typedef struct { char canzone[N];

Data d;

int voti;

int valido; } VotiGiornalieriCanzone;

typedef VotiGiornalieriCanzone voti[N];

Il campo “valido” serve a dire se la casella dell’array voti contiene contenuto valido (nel caso l’attributo ha valore 1) o è da considerarsi vuota (nel caso l’attributo ha valore 0). Non necessariamente le caselle valide dell’array voti sono tutte contigue.

Si codifichi in C la seguente funzione:

void f(voti V, char vincitore[])

che riceve un array *V* di voti giornalieri per le canzoni e inserisce nella stringa vincitore il titolo della canzone che complessivamente ha raccolto più voti*.*

Esercizio 1.2 ( 7 punti ) – RECUPERO PRIMA PROVA

Una matrice si dice sparsa se la maggior parte dei suoi elementi (più di metà) ha uno stesso unico valore, detto valore dominante. Ad esempio la seguente matrice di dimensione 6x4 è sparsa con valore dominante 3

3 3 3 2

3 0 3 3

5 3 3 4

1 0 3 3

3 3 3 3

Si codifichi una funzione ***int f(int M[][N], int R, int C)*** che riceve in ingresso una matrice M, che contiene solo numeri positivi, e due interi R e C che rappresentano, rispettivamente, il numero di righe e di colonne della matrice. La funzione restituisce 1 se la matrice è sparsa, 0 altrimenti. Si noti che non è noto a priori il possibile valore dominante.

Si codifichi una funzione ***int f(int M[][N], int R, int C)*** che riceve in ingresso una matrice M, che contiene solo numeri positivi, e due interi R e C che rappresentano, rispettivamente, il numero di righe e di colonne della matrice. La funzione restituisce il valore dominante se la matrice è sparsa, -1 se non lo è. Si noti che non è noto a priori il possibile valore dominante.

Esercizio 2.1 ( 4 punti ) SECONDA PROVA

Si dica cosa stampa il seguente codice

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int f( char \* x, char \* y, int k );

int g(int i, char \* a, char \* b);

int main() {

int i = 3;

char x[] = "P", y[] = "LA";

i++;

printf("%c", \*x);

i = f( x, y, i-1 );

return 0;

}

int f( char \* x, char \* y, int k ) {

if( k ) {

k=g(k,x,y);

}

return k+1;

}

int g(int i, char \* a, char \* b){

int j;

if(i%2==0)

j=0;

else

j=1;

printf("%c", b[j]);

i = f( a, b, i-1 );

return i;

}

Esercizio 2.2 ( 4 punti ) SECONDA PROVA

Il seguente database è utilizzato da un sito Web che consente di prenotare l’uso di furgoni per traslochi.

UTENTE (CODUTENTE, COGNOME, NOME, DATANASCITA, INDIRIZZO, DATAISCRIZIONE)

FURGONE (TARGA, CILINDRATA, CAPIENZA, COSTOORARIO)

PRENOTAZIONE (TARGA, CODUTENTE, DATA, ORA, DURATA)

Estrarre in SQL i codici degli utenti che hanno sempre prenotato furgoni con capienza maggiore di 2000 kg.

Estrarre in SQL i codici degli utenti che hanno sempre prenotato furgoni con capienza maggiore di 2500 kg.

UTENTE (CODUTENTE, COGNOME, NOME, DATANASCITA, INDIRIZZO, DATAISCRIZIONE)

FURGONE (TARGA, CILINDRATA, CAPIENZA, COSTOORARIO)

PRENOTAZIONE (TARGA, CODUTENTE, DATA, ORA, DURATA)

Estrarre in SQL gli utenti che nel gennaio 2017 hanno prenotato più volte che nel gennaio 2016.

Estrarre in SQL gli utenti che nel dicembre 2016 hanno prenotato più volte che nel dicembre 2015.

Esercizio 2.3 ( 6 punti ) SECONDA PROVA

Si consideri la seguente struttura dati relativa alle prenotazioni di furgoni:

#define N 100

typedef struct { int giorno, mese, anno; } Data;

typedef struct FN { char targa[8];

int capienza, cilindrata, costoOrario;

struct FN \* next; } Furgone;

typedef Furgone \* Furgoni;

typedef struct PN { char targa[8];

Data d;

int durata; // in ore

struct PN \* next; } Prenotazione;

typedef Prenotazione \* Prenotazioni;

Si codifichi in C una funzione che riceve una lista *P* di prenotazioni e una lista F di furgoni. La funzione restituisce il furgone che ha reso più soldi nel 2016.

Si codifichi in C una funzione che riceve una lista *P* di prenotazioni e una lista F di furgoni. La funzione restituisce il furgone che ha reso meno soldi nel 2016.

Furgone f1(Prenotazioni P, Furgoni F)

Si codifichi in C una funzione che riceve una lista *P* di prenotazioni e una lista F di furgoni. La funzione restituisce una lista contenente i dati dei 10 furgoni che hanno reso più soldi nel 2016.

Si codifichi in C una funzione che riceve una lista *P* di prenotazioni e una lista F di furgoni. La funzione restituisce una lista contenente i dati dei 10 furgoni che hanno reso meno soldi nel 2016.

Furgoni f2(Prenotazioni P, Furgoni F)

Esercizio **2.4** ( 4 punti ) – SECONDA PROVA

Si considerino la seguente definizione di lista

typedef struct NO { int dato;

struct NO \* next; } nodo;

typedef nodo \* lista;

e la seguente di albero binario:

typedef struct EL { int dato;

struct EL \* left, \* right; } node;

typedef node \* tree;

Implementare una funzione che riceve in ingresso una lista e un albero e restituisce 1 se la lista corrisponde a uno dei cammini dalla radice alle foglie, 0 altrimenti.

void f(lista L, tree T)