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 ( 7 punti )**

**Esercizio 2.4 ( 4 punti )**

**Voto finale:**

Esercizio 1.1 ( 7 punti ) – RECUPERO PRIMA PROVA

In un festival della birra si susseguono degustazioni di birre artigianali. Le strutture dati utilizzate sono le seguenti:

#define N 100

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

typedef struct { char nome[N]; //si suppone essere diverso per ogni birra

int gradoAlcolico;

char descrizione[N];

int valido; } Birra;

typedef struct { char nome[N];

Data data;

Birra Degustate[N];

int valido; } Degustazione;

typedef Degustazione Degustazioni[N];

Il campo “valido” serve a dire se la casella dell’array 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 sono tutte contigue.

Si codifichi in C la seguente funzione:

void f(Degustazioni D, Degustazioni ECCESSIVE, int limite)

che riceve un array *D* di degustazioni, un array *ECCESSIVE* di degustazioni inizialmente vuoto e un intero *limite* che rappresenta un tasso alcolico limite. La funzione deve copiare le degustazioni del mese di gennaio 2016 che prevedevano (almeno) una birra con tasso alcolico superiore al limite ammesso nell’array *ECCESSIVE* (riempiendo l’array dall’inizio e senza lasciare buchi) e deve eliminarle da *D.*

Esercizio 1.2 ( 8 punti ) – RECUPERO PRIMA PROVA

Operando con le matrici spesso è necessario “comprimere” matrici grandi in matrici più piccole che contengono in ogni casella un valore ottenuto come media aritmetica dei valori di più caselle della matrice di partenza.

Si scriva una funzione f che riceve in input una matrice quadrata *M1* con N righe e N colonne e una matrice quadrata *M2* con N/2 righe e N/2 colonne.

void f(int M1[][N], int M2[][N/2])

La matrice deve comprimere M1 in M2 inserendo in ogni casella di M2 la media (troncata) di 4 caselle adiacenti di M1 come nell’esempio:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 7 | 3 | 1 | 2 |
| 6 | 5 | 1 | 9 | 3 | 5 |
| 7 | 6 | 4 | 7 | 1 | 1 |
| 7 | 9 | 4 | 1 | 1 | 1 |
| 9 | 5 | 2 | 4 | 6 | 7 |
| 5 | 8 | 7 | 6 | 1 | 4 |

diventa

|  |  |  |
| --- | --- | --- |
| (2+3+6+5)/4 | (7+3+1+9)/4 | (1+2+3+5)/4 |
| (7+6+7+9)/4 | (4+7+4+1)/4 | (1+1+1+1)/4 |
| (9+5+5+8)/4 | (2+4+7+6)/4 | (6+7+1+4)/4 |

quindi

|  |  |  |
| --- | --- | --- |
| 4 | 5 | 2 |
| 7 | 4 | 1 |
| 6 | 4 | 4 |

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 ) {

if( !k )

printf(" & ");

else {

printf("%c", y[k%2]);

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

printf("%c", y[k%2]);

k++;

}

return k;

}

int main() {

int i = 3;

char inutile;

char x[] = "CI", y[] = "MO";

i++;

printf("%c", x[i%2]);

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

fflush(stdin);

scanf("%c",&inutile);

return 0;

}

Esercizio 2.2 ( 4 punti ) SECONDA PROVA

Il seguente schema descrive i dati di degustazioni di birre

Birra ( Nome, Descrizione, GradoAlcolico )

Degustazione ( Nome, Data, Luogo )

Assaggi (NomeDegustazione, NomeBirra, Quantità )

Estrarre in SQL le degustazioni in cui si assaggiano più di 5 birre.

SELECT NomeDegustazione

FROM Assaggi

GROUP BY NomeDegustazione

HAVING count(\*)>5

Birra ( Nome, Descrizione, GradoAlcolico )

Degustazione ( Nome, Data, Luogo )

Assaggi (NomeDegustazione, NomeBirra, Quantità )

Estrarre in SQL la degustazione col più alto tasso alcolico medio delle birre degustate.

SELECT NomeDegustazione, avg(GradoAlcolico)

FROM Assaggi JOIN Birra ON NomeBirra=Nome

GROUP BY NomeDegustazione

HAVING avg(GradoAlcolico) >= ALL SELECT avg(GradoAlcolico)

FROM Assaggi JOIN Birra ON NomeBirra=Nome

GROUP BY NomeDegustazione

Esercizio 2.3 ( 7 punti ) SECONDA PROVA

#define N 100

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

typedef struct BEL { char nome[N]; //si suppone essere diverso per ogni birra

int gradoAlcolico;

char descrizione[N];

struct BEL \* next; } Birra;

typedef struct DEL { char nome[N];

Data data;

Birra \* Birre;

struct DEL \* next; } Degustazione;

typedef Degustazione \* Degustazioni;

Si codifichi in C la seguente funzione:

Degustazioni f(Degustazioni \* D, int limite)

che riceve una lista *D* di degustazioni (si noti che è passato il puntatore al puntatore al primo elemento) e un intero *limite* che rappresenta un tasso alcolico limite. La funzione restituisce la lista delle degustazioni del mese di gennaio 2016 che prevedevano (almeno) una birra con tasso alcolico superiore al limite ammesso e elimina queste degustazioni da *D.*

Degustazioni f(Degustazioni \* D, int limite) {

Degustazioni P, temp=\*D,temp2;

while(temp!=NULL){

if(verificaPericolo(\*temp,limite)){

temp2= temp->next;

\*D=cancella(\*D,\*temp);

P=insInTesta(P,\*temp);

temp=temp2;

}

else

temp=temp->next;

}

return P;

}

int verificaPericolo(Degustazione d,int limite){

Birra \* temp=d.Birre;

while(temp!=NULL){

if(temp->gradoAlcolico > limite)

return 1

//scorri

temp=temp->next;

}

return 0;

}

Degustazioni **InsInTesta** ( Degustazioni **lista**, Degustazioni d) {

Degustazioni punt;

punt = d;

punt–>next = lista;

return punt;

}

Degustazioni **Cancella**( Degustazioni lista, Degustazione elem ) {

Degustazioni puntTemp;

if( lista!=NULL)

if( strcmp(lista–>info.Nome,elem.Nome) == 0 ) {

puntTemp = lista–>prox;

return puntTemp;

}

else

lista–>prox = **Cancella**( lista–>prox, elem );

return lista;

}

{}

{}

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

Si consideri la seguente definizione di un albero binario:

typedef struct EL { int dato;

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

typedef node \* tree;

Implementare una funzione

void f(tree TA, tree TB, tree TRES)

che riceve in ingresso tre alberi binari con la stessa profondità e di cui uno (*TRES*) completo (cioè ogni nodo fino alle foglie escluse ha esattamente due figli e tutte le foglie hanno la stessa profondità. La funzione inserisce in ogni nodo di TRES la somma degli elementi corrispondenti (nella stessa posizione) di TA e TB se esistono entrambi, il valore del nodo corrispondente di TA se esiste solo il nodo corrispondente di TA, il valore del nodo corrispondente di TB se esiste solo il nodo corrispondente di TB, 0 se non esiste nessuno dei due.

void f(tree TA, tree TB, tree TRES) {

if(TRES==NULL)

return;

if(TA!=NULL && TB!=NULL)

TRES->dato=TA->dato + TB->dato;

f(TA->left,TB->left,TRES->left); f(TA->right,TB->right,TRES->right);

else if(TA!=NULL)

TRES->dato=TA->dato;

f(TA->left,NULL,TRES->left); f(TA->right,NULL,TRES->right);

else if(TB!=NULL)

TRES->dato=TB->dato;

f(NULL, TB->left,TRES->left); f( NULL, TB->right,TRES->right);

else

TRES->dato=0;

f(NULL,NULL,TRES->left); f(NULL,NULL,TRES->right);

}

{}{}{}{}{}