Informatica A

**Cognome Matricola**

**Nome Firma**

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 occorre ricalcare al momento della consegna!).
4. È **vietato** utilizzare **calcolatrici,** **telefoni o pc**. 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: **2h 30min**

Valore indicativo degli esercizi, voti parziali e voto finale:

**Esercizio 1 ( 2 punti )**

**Esercizio 2 ( 4 punti )**

**Esercizio 3 ( 4 punti )**

**Esercizio 4 ( 4 punti )**

**Totale: ( 14 punti )**

Esercizio 1 (2 punti)

Si costruisca la tabella di verità della seguente espressione booleana.

**not ( (not A) or ( B and C ) ) or ( A and (C or B) )**

0 0 0 🡪 0

0 0 1 🡪 0

0 1 0 🡪 0

0 1 1 🡪 0

1 0 0 🡪 1

1 0 1 🡪 1

1 1 0 🡪 1

1 1 1 🡪 1

Si stabilisca il minimo numero di bit sufficiente a rappresentare in complemento a due i numeri A = 77dec e B = –123dec, li si converta, se ne calcolino la somma (A+B) e la differenza (A–B) in complemento a due e si indichi se si genera riporto sulla colonna dei bit più significativi e se si verifica overflow.

Servono 8 bit

77=01001101

123=01111011

-123=10000101

A+B=11010010 (no riporto, no overflow)

A-B=11001000 (no riporto, sì overflow)

Esercizio 2 (4 punti)

Si dica cosa stampa il seguente codice:

#include<stdio.h>

#define N 8

void t(int n, int p, int\*i, int\*f) {

if( i>=f )

printf(" NON TROVATO \n");

else if( n == \*i )

printf(" : dispari[%d] \n",p);

else

t( n, p+1, i+1, f );

}

void d( int n, int \*v ) {

if( n%2 )

t( n, 0, v, v+N );

else {

printf(" -> %d", n/2);

d(n/2, v);

}

}

int main(){

int dispari[4\*N]={1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47},

x[N]={7,18,24,17,88}, i=0;

while( x[i] > 0 ) {

printf("[%d]", x[i]);

d( x[i++], dispari );

}

return 0;

}

[7] : dispari[3]

[18] -> 9 : dispari[4]

[24] -> 12 -> 6 -> 3 : dispari[1]

[17] NON TROVATO

[88] -> 44 -> 22 -> 11 : dispari[5]

Esercizio 3 (4 punti)

Si considerino le seguenti dichiarazioni di tipi che definiscono le strutture dati per gestire le visite di un dentista.

typedef struct {

int giorno, mese, anno;

} data;

typedef struct {

int ora, minuto;

} orario;

typedef struct {

char codiceFiscale[100], cognome[100], nome[100];

data DataNascita;

} persona;

typedef struct {

persona pazienti[1000];

int numPazienti; //dice quante caselle dell’array contengono dati validi

} listaPazienti;

typedef struct {

char CFPaziente[100];

data d;

orario o;

} visita

typedef struct {

visita visite[1000];

int numVisite; //dice quante caselle dell’array contengono dati validi

} listaVisite;

Si definisca una funzione di prototipo

listaPazienti f(listaPazienti P, listaVisite V);

che restituisce la lista dei pazienti che hanno avuto una visita medica il giorno del loro compleanno.

listaPazienti f(listaPazienti P, listaVisite V) {

int i,j;

listaPazienti Pcomp;

Pcomp.numPazienti=0;

for(i=0;i<P.numPazienti;i++){

for(j=0;j<P.numVisite;j++){

if(strcmp(P. pazienti[i]. codiceFiscale,V.visite[j]. CFPaziente)==0

&& P. pazienti[i].data.giorno= V.visite[j].d.giorno

&& P. pazienti[i].data.mese= V.visite[j].d.mese)){

PComp.pazienti[Pcomp.numPazienti]=P. pazienti[i];

Pcomp.numPazienti++;

break;

}

}

}

return Pcomp;

}

int esisteVisitaStessoGiorno(persona p,listaVisite V){

int i;

for(i=0;i<V.numVisite;i++)

if(strcmp(p.codiceFiscale,V.visite[i].CFPaziente)==0

&& confDate(p.DataNascita, V.visite[i].d))

return 1;

return 0;

}

listaPazienti f(listaPazienti P, listaVisite V){

int i;

listaPazienti ris;

ris.numPazienti=0;

for(i=0;i<P.numPazienti;i++)

if(esisteVisitaStessoGiorno(P.pazienti[i],V)){

ris.pazienti[ris.numPazienti]=P.pazienti[i];

ris.numPazienti++;

}

return ris;

}

int confDate(Data d1, data d2){

if(d1.giorno==d2.giorno && d1.mese==d2.mese)

return 1;

else

return 0;

}

Esercizio 4 **(4 punti)**

Una matrice si definisce “riempita a scacchiera” se in essa sono presenti zeri e numeri diversi da zero di modo da formare una scacchiera.

Delle tre matrici sotto

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 5 | 0 | 21 | 0 | 7 |
| 6 | 0 | 4 | 0 | 2 | 0 |
| 0 | 4 | 0 | 11 | 0 | 8 |
| 55 | 0 | 33 | 0 | 55 | 0 |
| 0 | 5 | 0 | 21 | 0 | 9 |
| 22 | 0 | 1 | 0 | 32 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 22 | 0 | 1 | 0 | 32 | 0 |
| 0 | 5 | 0 | 21 | 0 | 7 |
| 6 | 0 | 4 | 0 | 2 | 0 |
| 0 | 4 | 0 | 11 | 0 | 8 |
| 55 | 0 | 33 | 0 | 55 | 0 |
| 0 | 5 | 0 | 21 | 0 | 9 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 5 | 0 | 21 | 0 | 7 |
| 0 | 4 | 0 | 2 | 0 | 4 |
| 0 | 4 | 0 | 11 | 0 | 8 |
| 55 | 0 | 33 | 0 | 55 | 0 |
| 0 | 5 | 0 | 21 | 0 | 9 |
| 22 | 0 | 1 | 0 | 32 | 0 |

Le prime due sono a scacchiera mentre la terza non lo è.

Definire una funzione

int f(int M[][N])

che riceve in input una matrice di dimensione NxN (con N costante definita globalmente con una define) e restituisce 1 se la matrice è a scacchiera, 0 altrimenti.

int f(int M[][N]) {

if(M[0][0]==0) {

for(i=0;i<N;i++){

for(j=0;j<N;j++){

if((i+j)%2==0 && M[0][0]!=0)

return 0;

if((i+j)%2!=0 && M[0][0]==0)

return 0;

}

}

} else {

for(i=0;i<N;i++){

for(j=0;j<N;j++) {

if((i+j)%2!=0 && M[0][0]!=0)

return 0;

if((i+j)%2==0 && M[0][0]==0)

return 0;

}

}

}

}