Informatica A – 18-2-2020

Cognome (IN STAMPATELLO)

Nome

Matricola o Codice Persona _ _

Istruzioni

Non separate questi fogli. **Scrivete la soluzione solo sui fogli distribuiti**, utilizzando il retro delle pagine in caso di necessità.

È possibile scrivere a matita (e non ricalcare al momento della consegna).

Ogni parte non cancellata sarà considerata parte integrante della soluzione.

È **vietato** utilizzare **calcolatrici** o **telefoni**. Chi tenti di farlo vedrà **annullata** la sua prova.

Non è ammesso consultare **libri** o **appunti** o altro.

Qualsiasi **tentativo** di comunicare con altri studenti comporta **l'espulsione** dall'aula.

È possibile **ritirarsi senza penalità** lasciando il tema d'esame con nome e cognome.

Non è possibile lasciare l'aula conservando il tema della prova in corso.

Tempo a disposizione: 2 ore e 30 minuti

Esercizio 1	**(2 punti)**	_____
Esercizio 2	**(4 punti)**	_____
Esercizio 3	**(4 punti)**	_____
Esercizio 4	**(10 punti)**	_____
Esercizio 5	**(8 punti)**	_____
Esercizio 6	**(4 punti)**	_____
Voto finale:		_____

Esercizio 1 (2 punti)

Si costruisca la tabella di verità della seguente espressione booleana.

(C or (not B)) or ((not A) or B)

É una tautologia perché le parentesi sono tutte superflue e abbiamo B or not B

Si stabilisca il minimo numero di bit sufficiente a rappresentare in complemento a due i numeri A = -101 e B = 118, li si converta, se ne calcoli 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.

Col metodo dei resti 101 -> 01100101
Quindi -101 –> 10011010+1=10011011
Col metodo dei resti 118 -> 01110110
Servono quindi 8 bit

Per effettuare A-B mi serve calcolare -118: 10001001+1=10001010

10011011 +
10001010 =
---
(1)00100101 c’è riporto e c’è overflow

Esercizio 2 (4 punti)

Lo schema sottostante descrive un portale per la pubblicazione e fruizione di video.

UTENTE (USERNAME, NOME, COGNOME, DATANASCITA, EMAIL)

VIDEO (IDVIDEO, TITOLO, USERNAMECREATOR, GENERE, DURATA, DATAPUBBLICAZIONE)

VISIONE (IDVIDEO, USERNAME, DATAORAVISIONE, MINUTIGUARDATI, VALUTAZIONE)

Scrivere una query che estrae nome e cognome degli utenti che non hanno mai visto un video.

```
Select USERNAME, NOME, COGNOME
From UTENTE
Where USERNAME not in (Select USERNAME
                       From VISIONE)
```

La parte azzurra non è necessaria per la correttezza dell'esercizio, ma mostra eventuali omonimi

Aggiungere condizioni nella seconda query tipo where MINUTIGUARDATI>0 o <QualcheCampo> is not NULL non era necessario (anzi, è proprio inutile), ma dato che non inficia i risultati non ha comportato penalità.
Non è corretto neanche

```
Select USERNAME, NOME, COGNOME
From UTENTE, VISIONE
Group by username
Having count(*)=0 o Having sum(MINUTIGUARDATI)=0
```

Perché NON esiste proprio il gruppo
Invece da soluzione corretta

```
Select USERNAME, NOME, COGNOME
From UTENTE U1
Where 0 = (Select count(*)=0                    o sum(MINUTIGUARDATI)
           From Visione V
           Where V.USERNAME=U1.USERNAME)
```

Qualunque soluzione con where count o where sum è certamente sbagliata

Lo schema sottostante descrive un portale per la pubblicazione e fruizione di video.

UTENTE (USERNAME, NOME, COGNOME, DATANASCITA, EMAIL)

VIDEO (IDVIDEO, TITOLO, USERNAMECREATOR, GENERE, DURATA, DATAPUBBLICAZIONE)

VISIONE (IDVIDEO, USERNAME, DATAORAVISIONE, MINUTIGUARDATI, VALUTAZIONE)

Scrivere una query che estrae, per ogni genere, il titolo del video col maggior numero di visualizzazioni.

```
Select IDVIDEO, TITOLO, GENERE
From VIDEO V1 JOIN VISIONE VS1 ON V1.IDVIDEO=VS1.IDVIDEO
Group by IDVIDEO, GENERE
Having count(*)>= ALL (Select count(*)
                       From VIDEO V2 JOIN VISIONE VS2 ON V2.IDVIDEO=VS2.IDVIDEO
                       Where V1.GENERE=V2.GENERE
                       Group by V2.IDVIDEO)
```

La condizione in verde serve a legare i conteggi di modo confrontare a parità di genere, perché la query chiede “per ogni genere”.

Esercizio 3 (4 punti)

Si consideri il seguente programma, completando l'inizializzazione del vettore V con la propria matricola (composta da 6 cifre) e si scriva nel riquadro sottostante l'output stampato a schermo.

```
void f(char * str);

int main() {
  char V[7] = "ABCDEF"; // METTERE QUI LA VOSTRA MATRICOLA
  f(V+3);
  return 0;
}

void f(char * str) {
    printf("%c", *str);
    str=str+1;
    if(strlen(str) == 0)
      return;
    else {
      f(str);
      printf("%c", *str);
      return;
    }
}
```

```
>>>

DEFFE
```

Ho messo sei lettere diverse per far vedere che importano solo le posizioni

Esercizio 4 (10 punti)

Sia f una funzione analitica di due variabili e a valori reali. Si assuma che f sia stata campionata in una matrice float M[100][100] tale per cui ogni posizione i,j della matrice contiene il valore di f nelle coordinate (i,j):
M[i][j] = $f(i,j)$, $i,j = 0,\dots,99$
Si scriva una funzione `estraiMassimiLocali` che prende in ingresso la matrice M e restituisce una lista contenente tutti i massimi locali di M. Si definisca preventivamente un opportuno tipo di dato Nodo che deve contenere, oltre alle coordinate del massimo locale, anche il valore della funzione.
Si scriva un frammento di codice per invocare `estraiMassimiLocali` sulla matrice M, assumendo che questa sia già stata popolata.
Un massimo locale è una coordinata (i,j) dove il valore di M[i][j] è strettamente maggiore di tutte le 8 coordinate (i,j) adiacenti.
Si suggerisce di scrivere una funzione ausiliaria `maxLoc` che restituisce il valore massimo tra tutti i valori adiacenti ad una posizione passata in ingresso.
Ricordatevi di gestire i bordi della matrice (in questi casi le caselle adiacenti sono meno di 8).

```
typedef struct NodeL { int val,i,j; struct NodeL * next } Node;
typedef Node * Lista;

int main(){
        float M[N][N];
        int i,j;
        Lista l = NULL;
        for(i = 0; i < N; i++)
                for(j=0; j < N; j++)
                        scanf("%f",&M[i][j]);

        l = estraiMassimiLocali(M);
        VisualizzaLista(l);

        return 0;
}
Lista InsInFondo( Lista lista,int v,int i,int j ) {
```

```
   Lista punt;
   if( lista==NULL ) {
        punt = malloc( sizeof(Node) );
        punt–>next = NULL;
        punt–>val = v;  punt–>i = i;  punt–>j = j;
        return  punt;
   }   else {   lista–>next = InsInFondo( lista–>next, elem );
                return lista;   }
}
Lista estraiMassimiLocali(float M[][100]){
          int i,j;
          Lista lis=NULL;
          for(i=0;i<100;i++)
                    for(j=0;j<100;j++)
                              if(maxLoc(M,i,j)==1)
                                        lis=insInFondo(lis,M[i][j],i,j);
          return lis;
}

int maxLoc(float M[][100],int i,int j){
          int k,t,cont=0;
          //conto le otto posizioni possibili (nove, ma per una gli if sono falsi per forza)
          for(k=i-1;k<=i+1;k++)
                    for(t=j-1;t<=j+1;t++)
                              if(k<0 || k>=100 || t<0 || t>=100)//se fuori dalla matrice
                                        cont++;
                              else if(M[i][j]> M[k][t])
                                        cont++;
          if(cont==8)
                    return 1;
          else return 0;
}

OPPURE
```

```
Lista estraiMassimiLocali(float M[N][N]){
    int i,j;
    Lista l = NULL;

    for(i = 0; i < N; i++)
        for(j = 0; j < N; j++) {
                if(M[i][j] > maxLoc(M,i,j)) {
                    l = InsInTesta(l, i, j, M[i][j]);
                    //printf("\n%d, %d, %f", i, j, M[i][j]);
                 }
            }
    return l;
}

float maxLoc(float M[N][N], int r, int c){
    int i,j, initialized = 0;
    float m;
    for(i = -1; i <=1; i++)
        for(j = -1; j <=1; j++)
            if(r + i >= 0 && r + i < N && c + j > 0 && c + j < N && !(i == 0 && j ==0))
                if(initialized == 0)
                        {
                                m =  M[r + i][c + j];
                                initialized = 1;
                        }
                        if(m < M[r + i][c + j])
                                m =  M[r + i][c + j];
    return m;
}

Lista InsInTesta (Lista l, int r, int c, float val) {
    Lista punt;
    punt = (Lista) malloc(sizeof(Nodo));
    punt->i = r;    punt->j = c;    punt->val = val;
```

```
    punt->next = l;
    return punt;
}
```

Esercizio 5 (8 punti)

Si consideri la seguente definizione di lista:

```
typedef struct EL {
     int dato;
     struct EL * next;
} nodo;
typedef nodo * lista;
```

Scrivere una funzione

Lista Merge(Lista lista1, Lista lista2)

che prese in input due liste ordinate per valori crescenti, fornisce in output una nuova lista ordinata contenente tutti gli elementi delle due liste (duplicati inclusi).

```
Lista Merge(Lista lista1, Lista lista2){
      Lista ris=NULL;
      while(lista1!=NULL){
            ris=InsInOrd(ris,lista1->dato);
            lista1=lista1->next;
      }
      while(lista2!=NULL){
            ris=InsInOrd(ris,lista2->dato);
            lista2=lista2->next;
      }
      return ris;
}
Lista InsInOrd( Lista lista, int elem ) {
      Lista punt, puntCor = lista, puntPrec = NULL;
      while ( puntCor != NULL  &&  elem > puntCor->info ) {
            puntPrec = puntCor;
            puntCor = puntCor->next;
      }
      punt = (Lista) malloc(sizeof(nodo));
```

```
  punt->dato = elem; punt->next = puntCor;
  if ( puntPrec != NULL ) {
        puntPrec->next = punt;
        return lista;
  } else
        return punt;
}
```

Esercizio 6 (4 punti)

Si consideri la seguente definizione di albero binario che contiene i dati relativi alla gerarchia aziendale di un'azienda in cui ogni persona è a capo di uno o due sottoposti:

```
typedef struct node_s {
char nome[100],cognome[100];
struct node_s * left, *right;
} node_t;
typedef node_t * tree;
```

Scrivere una funzione
void cognomeDiffuso(tree T, char str[])
che preso in ingresso un albero e una stringa, inserisca nella stringa il cognome più diffuso nell'azienda (si supponga sia unico).

```
typedef struct Elemento { char parola[100];
                          int occorrenze;
                          struct Elemento * next;  } Nodo;
typedef Nodo * Lista;

Lista inserisci ( Lista lis, char * p ) {
  if( lis == NULL ) {
    lis = (Nodo) malloc(sizeof(Nodo));
    lis->next = NULL;
    lis->occorrenze = 1;
    strcpy(lis->parola, p);
  }
  else if ( strcmp(p, lis->parola)==0 )
    lis->occorrenze = lis->occorrenze + 1;
  else lis->next = inserisci ( lis->next, p );
  return lis;
}

Lista creaLista ( Lista lis, tree T ){
        if(T==NULL)
```

```
                return lis;
        lis = inserisci ( lis, T->cognome );
        lis = creaLista( lis, T->left );
        lis = creaLista( lis, T->right );
        return lis;
}

void scegliParola(Lista lis, char str[]) {
        int max=0;
        if(lis==NULL) // controllo inutile, è impossibile accada
                strcpy(str,"");

        while(lis!=NULL){
                if( lis->occorrenze > max ){
                        max = lis->occorrenze;
                        strcpy(str,lis->parola);
                }
        }
        return;
}

void cognomeDiffuso(tree T, char str[]) {
        Lista lis=NULL;
        if(t==NULL)
                strcpy(str,"");
        lis=creaLista(lis,T);
        scegliParola(lis,str);
        return;
}
```

```
int conta(tree T, char str[]){
	if(t==NULL)
		return 0;
	if(strcmp(T->cognomen,str)==0)
		return 1+conta(T->left,str)+conta(T->right,str);
	else
		return conta(T->left,str)+conta(T->right,str);
}
```