Quesito 1

Si definisca un tipo "Boolean" e un tipo atto a rappresentare una Matrice di N x N interi, con N predefinita
tramite una direttiva define.
Si scriva una funzione ricorsiva che, ricevendo in ingresso una Matrice e un eventuale altro parametro utile,
restituisca True se la Matrice ricevuta in ingresso è simmetrica, False in caso contrario.

```
typedef enum {False, True} Boolean;
typedef int Matrice[N][N];


Boolean Simmetrica(Matrice M, int Dim)
{       int i;
        if(Dim==1) return True;
        for (i=0; i<Dim-1;i++)
                if (M[i][Dim-1] != M[Dim-1][i]) return False;
        return Simmetrica(M, Dim-1)
}
```

Quesito 2

Dato il tipo Matrice, definita come matrice quadrata di N x N interi, con N definita tramite una direttiva define, scrivere un sottoprogramma ricorsivo che calcoli la differenza tra la somma degli elementi della diagonale principale e la somma degli elementi della diagonale secondaria.

```
Soluzione:
int DiffDiag(Matrice M, int Iniz,int Fin)
{
      if (Fin-Iniz <=1) return (0);
      return (M[Iniz,Iniz]+M[Fin,Fin]-M[Iniz,Fin]-M[Fin,Iniz]+
                  DiffDiag(M,Iniz+1,Fin-1);
}
```

Quesito 3

Scrivere un sottoprogramma ricorsivo che ricevuto in ingresso un numero intero restituisca la cifra più alta della sua rappresentazione decimale.
Ad esempio, se il valore ricevuto in ingresso è milleduecentotrentadue il sottoprogramma restituisce 3.

Soluzione:

```
int CifraMassima(int N)
{
  int temp;
  if (N  /10 == 0) return N;
  temp = CifraMassima(N/10);
  if (N % 10 > temp) return (N % 10);
  else return temp;
}
```

Quesito 4. Si scriva una funzione **ricorsiva** Converti che, ricevendo in ingresso un intero positivo N (inizialmente non nullo) e una base intera b > 1, restituisce una lista sequenziale che contiene la rappresentazione di N in base b. Si supponga che il campo array della lista sequenziale abbia dimensione sufficiente a contenere tutte le cifre che costituiscono la rappresentazione di N in base b.
NOTA: una lista sequenziale è a una struct a due campi: un campo contiene un array mentre l'altro campo contiene un intero che indica il numero di elementi dell'array effettivamente occupati.

Soluzione:

```
typedef struct {int Nel: int Arr[100];} SeqCifre;
SeqCifre Converti (int N, int b)
{
 SeqCifre L;
 int quoziente, resto;

 if (N == 0) {L.Nel=0; return (L);}
 quoz = N / b;
 resto = N – b * quoz;
 L = Converti (quoz, b);
 L.Arr[L.Nel] = resto;
 L.Nel++;
 return (L);
}
```

Quesito 5

Una Parola è rappresentata come una stringa di al più 20 caratteri compreso il carattere di terminazione. La ListaOccorrenze, associata a una Parola, è una lista sequenziale -di al più 19 elementi- ognuno dei quali contiene un carattere facente parte della parola e il numero di volte in cui questo carattere compare nella parola.
Si definiscano il tipo Parola e il tipo ListaOccorrenze. Si scriva poi una procedura ricorsiva che, ricevendo in ingresso una Parola, costruisca la ListaOccorenze associate alla parola ricevuta in ingresso. L'ordine con cui i vari caratteri devono essere posizionati nella ListaOccorrenze deve essere lo stesso con cui essi compaiono per la prima volta nella Parola.
Ad esempio:
la ListaOccorrenze associata alla parola `ordini` è |5, ['o',1] ['r',1] ['d',1] ['i',2] ['n',1] [*,*] [*,*] … [*,*]|
mentre la ListaOccorrenze associata alla parola `amaca` è |3, ['a',3] ['m',1] ['c',1] [*,*] [*,*] … [*,*] |.

Soluzione:

```
typedef enum {false, true} Boolean;
typedef struct {char car;
                int num;} Occorrenza;
typedef struct {int N;
                Occorrenza Seq[19};} ListaOccorrenze;

void CostruisciListaOccorrenze(char Parola[], ListaOccorrenze *PL)
{
 int j;
 Boolean trovato = false;
 if (*Parola == '\0') return;
 for (j=0; j < PL->N && !trovato; j++)
      if (PL->Seq[j].car == *Parola)
      { (PL->Seq)[j].num++;
        trovato = true;}
 if (!trovato)
 { (PL->Seq)[PL->N].car = *Parola;
   (PL->Seq)[PL->N].num = 1;
   (PL->N)++;}
 CostruisciListaOccorrenze(Parola+1, PL)
}
```

Chiamata:

```
/* Acquisizione della parola in Stringa*/
L.N=0;
CostruisciListaOccorrenze(Stringa,  &L);
```

Quesito 6

Si scriva una funzione ricorsiva Calcola che, ricevendo in ingresso un'espressione algebrica, ne restituisca il valore. Un'espressione algebrica semplice consiste in un valore float, un'espressione algebrica (complessa) è costituita da una parentesi aperta seguita da un primo operando (che è a sua volta un'espressione semplice o complessa), da un operatore (rappresentato da un carattere '+', '-', '*', o '/'), da un secondo operando (un'espressione) e infine da una parentesi chiusa. Un espressione algebrica sia rappresentata tramite un array di elementi. Ciascun elemento può consistere in un valore float oppure da un operatore oppure da una parentesi aperta oppure una parentesi chiusa.

Soluzione:

```
typedef enum {Valore, Operando, ParentesiAperta, ParentesiChiusa} Tipoelemento;
typedef struct {Tipoelemento Tipo;
                char Qualeoperando;
                float QualeValore;} Elemento;

float Calcola (Elemento *E)
{
 Elemento *PuntI, *PuntII, *Punt;
 char op;
 int diff = 0; /* differenza tra numero di parentei aperte e numero di parentesi
                  chiuse durante il calcolo di un'espressione */
 if (E -> Tipo == Valore) return (E -> Qualevalore);
 if (E -> Tipo == ParentesiAperta)
 {
        Punt = E + 1;         /* inizio del primo operando */
        Punt = PuntI + 1      /* punto di partenza per ricerca operatore */
        if (PuntI -> Tipo == ParentesiAperta) diff++;
        while (diff > 0)
        {
            if (Punt -> Tipo == ParentesiAperta) diff++;
            if (Punt -> Tipo == ParentesiChiusa) diff--;
            Punt = Punt +1;
        }
        op = Punt -> QualeOperatore;
        PuntII = Punt +1;       /* inizio del secondo operando */
        if (op == '+') return (Calcola(PuntI) + Calcola(PuntII));
        if (op == '-') return (Calcola(PuntI) - Calcola(PuntII));
        if (op == '*') return (Calcola(PuntI) * Calcola(PuntII));
        if (op == '/') return (Calcola(PuntI) / Calcola(PuntII));
 }
}
```