Archivio Appelli

Archivio degli Appelli per il corso “Introduzione alla Programmazione’’

Claudio Lucchese, Alvise Spanò, Francesco Busolin, Federico Marcuzzi, Alberto Veneri

Università Ca’Foscari Venezia

September 8, 2023

Contents

Prova Intermedia del 04/11/2020

Istruzioni

Il testo prevede 4 domande a risposta multipla obbligatorie per tutti. Dopo lo studente potrà scegliere tra completare la Parte A o completare la parte B. Completare correttamente tutta la Parte A consentirà di avere un voto massimo di 25, mentre completare correttamente tutta la Parte B consentirà di avere un voto massimo pari a 30L. Non è quindi obbligatorio completare entrambe le parti, ad esempio chi completa la Parte B non deve completare la Parte A.

In ogni caso é obbligatorio attenersi allo standard ANSI C.

Domande a risposta multipla

1. Scope. (Variante A - punti 4)

Considerare il seguente codice:

```c
int x = 1;

int g(int y){
  return ++x + y;
}

int f(int x) {
  int z = x;
  {
    int x = g(z);
    return x + 1;
  }
}
```

Qual è il valore restituito dalla chiamata di funzione `f(0)`?

- ☐ Il risultato è 2
- ☐ Il risultato è 3
- ☐ Il codice non compila perché `x` non è definito nella funzione `g`
- ☐ Il codice non compila perché la sintassi della funzione `f` non è corretta per via delle doppie parentesi graffe

1. Scope. (Variante B - punti 4)

Considerare il seguente codice:

```c
int z = 1;

int r(int y){
  return ++z * y;
}

int s(int z) {
```

```
  int w = z;
  {
    int z = r(w);
    return z / 2;
  }
}
```

Qual è il valore restituito dalla chiamata di funzione `s(2)`?

- ☐ Il codice non compila perché `z` non è definito nella funzione `r`
- ☐ Il codice non compila perché la sintassi della funzione `s` non è corretta per via delle doppie parentesi graffe
- ☐ Il risultato è `1`
- ☐ Il risultato è `2`

2. Puntatori. (Variante A - punti 4)

Si consideri il seguente codice:

```
int a = 2;
int *b = &a;
int c[3] = { 0, 1, 2 };
int *d = c;

if (*(d + a) == c[a])
    *b = *c;
c[*b] = *(c + *d + 1);
```

Quanto valgono gli elementi dell'array `c` dopo aver eseguito il codice di cui sopra?

- ☐ Il codice non compila perché l'espressione `c + *d + 1` all'ultima riga si riferisce ad una locazione di memoria fuori dall'array `c`
- ☐ Il codice non compila perché l'espressione `d + a` nella guardia dell'if non ha tipo `int` ma `int*`
- ☐ L'array `c` è cambiato in { 1, 1, 2 }
- ☐ L'array `c` è immutato: { 0, 1, 2 }

2. Puntatori. (Variante B - punti 4)

Si consideri il seguente codice:

```
int a = 2;
int *b = &a;
int c[3] = { 0, 1, 2 };
int *d = c;

if (*(d + a) == c[a])
    *b = *(c + a) / 2;
c[*b] = *(c + *b + *d);
```

Quanto valgono gli elementi dell'array `c` dopo aver eseguito il codice di cui sopra?

- ☐ Il codice non compila perché l'espressione `c + *b + *d` all'ultima riga si riferisce ad una locazione di memoria fuori dall'array `c`
- ☐ Il codice non compila perché l'espressione `d + a` nella guardia dell'if non ha tipo `int` ma `int*`
- ☐ L'array `c` è cambiato: { 1, 1, 2 }
- ☐ L'array `c` è immutato: { 0, 1, 2 }

3. Array e Matrici. (Variante A - punti 4)

Si prenda in considerazione il seguente programma sorgente:

```c
long mult_2nd_diag(int (*a)[10]) {
    int i;
    long r = 1L;
    for (i = 0; i < 10; ++i)
        r *= a[i][9 - i];
    return r;
}

int main() {
    int a[10][10];
    int i, j;
    long res;
    for (i = 0; i < 10; ++i)
        for (j = 0; j < 10; ++j)
            a[i][j] = i + j + 1;
    res = mult_2nd_diag(a);
    return 0;
}
```

Quale valore si trova nella variabile `res` di tipo **`long`** dopo l'invocazione della funzione `mult_2nd_diag` appena prima della fine del `main`?

- ☐ Il codice compila ma il programma fallisce a runtime perché non si può scorrere una matrice quadrata con un solo ciclo **`for`**
- ☐ 1000
- ☐ 1001
- ☐ 10,000,000,000 (10 miliardi)

3. Array e Matrici. (Variante B - punti 4)

Si prenda in considerazione il seguente programma sorgente:

```c
long sub_2nd_diag(int (*a)[10]) {
    int i;
    long r = 1000L;
    for(i = 0; i < 10; ++i)
        r -= a[i][9 - i];
    return r;
}
int main() {
    int a[10][10];
    int i, j;
    long res;
    for(i = 0; i < 10; ++i)
        for(j = 0; j < 10; ++j)
            a[i][j] = i + j + 1;
    res = sub_2nd_diag(a);
    return 0;
}
```

Quale valore si trova nella variabile `res` di tipo **`long`** dopo l'invocazione della funzione `sub_2nd_diag` appena prima della fine del `main`?

- ☐ Il codice compila ma il programma fallisce a runtime perché non si può scorrere una matrice quadrata con un solo ciclo **`for`**
- ☐ 900
- ☐ 991
- ☐ 890

4. Loops. (Variante A - punti 4)

Si prenda in considerazione il seguente ciclo `while`:

```c
int i = 0;
while (i++ < 12) {
    printf("%d ", ++i);
}
```

Ed il seguente ciclo `for`:

```c
int i;
for (i = 2; i <= 12; ++i) {
    printf("%d ", i++);
}
```

Cosa stampano in standard output?

- ☐ Stampano entrambi i numeri: 2 4 6 8 10 12
- ☐ La prima stampa i numeri da 1 a 11; la seconda i numeri da 2 a 12
- ☐ Stampano entrambi i numeri da 2 a 11
- ☐ La prima stampa i numeri: 1 3 5 7 9 11; la seconda i numeri: 3 5 7 9 11

4. Loops. (Variante B - punti 4)

Si prenda in considerazione il seguente ciclo `while`:

```c
int i = 1;
while (++i < 12) {
    printf("%d ", ++i);
}
```

Ed il seguente ciclo `for`:

```c
int i;
for (i = 12; i > 2; --i) {
    printf("%d ", --i);
}
```

Cosa stampano in standard output?

- ☐ Stampano entrambi i numeri: 2 4 6 8 10 12
- ☐ La prima stampa i numeri da 1 a 11; la seconda i numeri da 12 a 1
- ☐ La prima stampa i numeri: 2 4 6 8 10; la seconda i numeri: 10 8 6 4 2
- ☐ La prima stampa i numeri: 3 5 7 9 11; la seconda i numeri: 11 9 7 5 3

5. Tipi

Si prendano in considerazione le seguenti dichiarazioni con inizializzazione:

```c
int k[2] = { 0, 1 };
int* p1 = k;
int** v[3] = { &p1, &p1, &p1 };
int* a[2][3] = { { k, k, k }, { **v + 0, **v + 1, **v } };
int** x = a[*(*v[*k] + 1)];
```

Che tipo ha l'espressione `x[2][1]` e quanto vale?

- ☐ Ha tipo `int*` ed è l'indirizzo del sotto-array che rappresenta la riga 1 dell'array bidimensionale `a`
- ☐ Ha tipo `int` e vale `1`
- ☐ Ha tipo `int**` ed è l'indirizzo della variabile `p1`
- ☐ Ha tipo `int(*)[3]` ed è l'indirizzo dell'array bidimensionale `a`

Soluzioni Domande

1. Scope variante A

☒ Il risultato è `3`

1. Scope variante B

☒ Il risultato è `2`

2. Puntatori variante A

☒ L'array `c` è cambiato in { 1, 1, 2 }

2. Puntatori variante B

☒ L'array `c` è immutato: { 0, 1, 2 }

3. Array e Matrici variante A

☒ `10,000,000,000` (10 miliardi)

3. Array e Matrici variante B

☒ `900`

4. Loops variante A

☒ Stampano entrambi i numeri: 2 4 6 8 10 12

4. Loops variante B

☒ La prima stampa i numeri: 3 5 7 9 11; la seconda i numeri: 11 9 7 5 3

5. Tipi

☒ Ha tipo `int` e vale `1`

Esercizi Parte A (voto 18-25)

1. Trova l'intruso. (Variante A - punti 3)

Scrivere una funzione `find` che dato un array `a` trovi l'elemento *minimo* il cui quadrato è maggiore di 66. Se non c'è nessun elemento che soddisfa il criterio, la funzione restituisce -1.

Esempi:

```
int a[5] = { 7, 8, 10, 9, 12 };
int result = find(a, 5); /* 5 è la lunghezza dell'array */
```

`result` vale 9, perché i quadrati di 7 e 8 sono minori di 66, mentre 10 e 12 sono di 9.

```
int a[5] = { 1, 2, 3, 4, 5 };
int result = find(a, 5); /* 5 è la lunghezza dell'array */
```

`result` vale -1, perché non c'è nessun elemento il cui quadrato sia maggiore di 66.

```
int a[5] = { 7, 8, 10, 9, -100 };
int result = find(a, 5); /* 5 è la lunghezza dell'array */
```

`result` vale -100, perché il suo quadrato è maggiore di 66, ed è chiaramente il minimo dell'array.

1. Trova l'intruso. (Variante B - punti 3)

Scrivere una funzione `find` che dato un array `a` trovi l'elemento *massimo* il cui quadrato è pari. Se non c'è nessun elemento che soddisfa il criterio, la funzione restituisce -1.

Esempi:

```
int a[5] = { 1, 7, 3, 9, 11 };
int result = find(a, 5); /* 5 è la lunghezza dell'array */
```

`result` vale -1, perché non c'è nessun elemento il cui quadrato sia pari.

```
int a[5] = { 1, 7, 3, -80, -100};
int result = find(a, 5); /* 5 è la lunghezza dell'array */
```

`result` vale -80, perché è maggiore di -100 ed il suo quadrato è pari.

2. Massima differenza (Variante A - punti 3)

Scrivere una funzione `max_diff` che dati due array `a` e `b` della stessa lunghezza, calcoli la massima differenza tra gli elementi di `a` e `b` nella stessa posizione, definita come segue (assumiamo gli array abbiano almeno un elemento):

$$\text{max\_diff}(a, b) = \max_i a[i] - b[i]$$

Esempi:

```
int a[5] = { 1, 2, 5, 8, 10 };
int b[5] = { 3, 4, 5, 6, 5 };
int result = max_diff(a, b, 5); /* 5 è la lunghezza degli array */
```

`result` vale 5, perché è la massima differenza è quella tra 10 e 5.

```
int a[5] = { 1, 2, -5, 8 };
int b[5] = { 3, 4, -9, -6 };
int result = max_diff(a,b,4); /* 4 è la lunghezza degli array */
```

`result` vale 14, perché è la massima differenza è quella tra 8 e -6.

2. Minimo prodotto. (Variante B - punti 3)

Scrivere una funzione `min_prod` che dati due array `a` e `b` della stessa lunghezza, calcoli il minimo prodotto tra gli elementi di `a` e `b` nella stessa posizione, definito come segue (assumiamo gli array abbiano almeno un elemento):

$$\text{min\_prod}(a, b) = \min_i a[i] * b[i]$$

Esempi:

```
int a[5] = { 1, 2, 5, 8, 1 };
int b[5] = { 3, 4, 5, 6, 5 };
int result = min_prod(a, b, 5); /* 5 è la lunghezza degli array */
```

`result` vale 3, perché il minimo prodotto è quello tra 1 e 3.

```
int a[5] = { 0, 2, 5, 8, 2 };
int b[5] = { 3, 4, -5, 6, 5 };
int result = min_prod(a, b, 5); /* 5 è la lunghezza degli array */
```

`result` vale -25, perché il minimo prodotto è quello tra 5 e -5.

3. Sequenza. (Variante A - punti 3)

Scrivere una funzione `p` che dato un intero `n`, dove `n` > 0, calcoli con un ciclo iterativo l'elemento `n`-esimo della sequenza così definita matematicamente:

$$p(n) = \begin{cases} 1 & \text{se } 0 < n \leq 2 \\ 2 & \text{se } 2 < n \leq 4 \\ p(n-1) + 3p(n-2) - 2p(n-4) & \text{altrimenti} \end{cases}$$

I primi numeri della sequenza sono:

```
1 1 2 2 6 10 24 ...
```

3. Sequenza. (Variante B - punti 3)

Scrivere una funzione `q` che dato un intero `n`, dove `n` > 0, calcoli con un ciclo iterativo l'elemento `n`-esimo della sequenza così definita matematicamente:

$$q(n) = \begin{cases} 1 & \text{se } 0 < n \leq 3 \\ 2 & \text{se } n = 4 \\ 3q(n-1) - 2q(n-2)q(n-4) & \text{altrimenti} \end{cases}$$

I primi numeri della sequenza sono:

```
1 1 1 2 4 8 16 ...
```

Soluzioni Esercizi Parte A

Trova l'intruso variante A

```c
int find_a(int *a, int a_size) {
  int found = 0;
  int smallest,i;
  for (i=0; i<a_size; i++) {
    int q = a[i]*a[i];
    if (q>66) {
      if (!found || a[i]<smallest) {
        found = 1;
        smallest = a[i];
      }
    }
  }
  if (found)
    return smallest;
  else
    return -1;
}
```

Trova l'intruso variante B

```c
int find_b(int *a, int a_size) {
  int found = 0;
  int largest,i;
  for (i=0; i<a_size; i++) {
    int q = a[i]*a[i];
    if (q%2==0) {
      if (!found || a[i]>largest) {
        found = 1;
        largest = a[i];
      }
    }
  }
  if (found)
    return largest;
  else
    return -1;
}
```

Massima differenza variante A

```c
#include <stdlib.h>

int max_diff(int *a, int *b, int size) {
  int i;
  int m_diff = a[0]-b[0];
  for (i=1; i<size; i++) {
    int diff = a[i]-b[i];
    if (diff>m_diff) {
      m_diff = diff;
    }
  }
  return m_diff;
}
```

Minimo prodotto variante B

```c
#include <stdlib.h>

int min_prod(int *a, int *b, int size) {
  int i;
  int m_prod = a[0]*b[0];
  for (i=1; i<size; i++) {
    int prod = a[i]*b[i];
    if (prod<m_prod) {
      m_prod = prod;
    }
  }
  return m_prod;
}
```

Sequenza variante A

```c
int p(int n) {
  int p, p_1, p_2, p_3,p_4;
  int i;

  if (n<=2)
    return 1;
  if (n<=4)
    return 2;

  /* n==4 */
  p_3 = 1;
  p_2 = 1;
  p_1 = 2;
  p   = 2;
  for (i=5; i<=n; i++) {
    /* prepare */
    p_4 = p_3;
    p_3 = p_2;
    p_2 = p_1;
    p_1 = p;
    /* compute */
    p = p_1 + 3*p_2 - 2*p_4;
  }
  return p;
}
```

Sequenza variante B

```c
int q(int n) {
  int q, q_1, q_2, q_3,q_4;
  int i;

  if (n<=3)
    return 1;
  if (n==4)
    return 2;

  /* n==4 */
```

```
  q_3 = 1;
  q_2 = 1;
  q_1 = 1;
  q   = 2;
  for (i=5; i<=n; i++) {
    /* prepare */
    q_4 = q_3;
    q_3 = q_2;
    q_2 = q_1;
    q_1 = q;
    /* compute */
    q = 3*q_1 - 2*q_2*q_4;
  }
  return q;
}
```

Esercizi Parte B (voto 18-30L)

1. Riempi la matrice. (Variante A - punti 5)

Scrivere una funzione `void fill(int* m, int n, int k)`, dove `m` è un puntatore ad una matrice quadrata di interi (disposta in memoria come un array *flattened* / *appiattito*) di `n` righe e `n` colonne, che riempia la matrice `m` con valori `1` e `0` formando una diagonale principale (da sinistra a destra) con uno spessore pari a `k` come negli esempi che seguono.

Esempi:

```
int m[5][5];
fill(&m[0][0], 5, 2);
```

Il codice sopra riempie la matrice `m` come segue:

```
1 1 0 0 0   /* nota: due volte 1 sulla prima riga e sulla prima colonna */
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 0 1 1
```

```
int m[6][6];
fill(&m[0][0], 6, 4);
```

Il codice sopra riempie la matrice `m` come segue:

```
1 1 1 1 0 0  /* nota: quattro volte 1 sulla prima riga e sulla prima colonna */
1 1 1 1 1 0
1 1 1 1 1 1
1 1 1 1 1 1
0 1 1 1 1 1
0 0 1 1 1 1
```

1. Riempi la matrice. (Variante B - punti 5)

Scrivere una funzione `void fill(int* m, int n, int k)`, dove `m` è un puntatore ad una matrice quadrata di interi (disposta in memoria come un array *flattened* / *appiattito*) di `n` righe e `n` colonne, che riempia la matrice `m` con valori `1` e `0` formando una diagonale secondaria (da destra a sinistra) con uno spessore pari a `k` come negli esempi che seguono.

Esempio:

```c
int m[5][5];
fill(&m[0][0], 5, 2);
```

Il codice sopra riempie la matrice `m` come segue:

```
0 0 0 1 1    /* nota: due volte 1 sulla prima riga e sull'ultima colonna */
0 0 1 1 1
0 1 1 1 0
1 1 1 0 0
1 1 0 0 0
```

```c
int m[6][6];
fill(&m[0][0], 6, 4);
```

Il codice sopra riempie la matrice `m` come segue:

```
0 0 1 1 1 1  /* nota: quattro volte 1 sulla prima riga e sull'ultima colonna */
0 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 0
1 1 1 1 0 0
```

2. Pattern match (Variante A - punti 5)

Dati due array di interi positivi `a` e `pattern`, vogliamo sapere se esistono dei sotto-array di `a` esattamente uguali a `pattern`; valgono anche le sovrapposizioni parziali, come si vede nell'esempio. Scrivere una funzione `count` che restituisce il numero di occorrenze di `pattern` in `a`.

Esempio:

```c
int pattern[3] = { 1, 0, 1 };
int a[8] = { 5, 0, 1, 0, 1, 0, 1, 5 };
int result = count(pattern, 3, a, 8); /* 3 e 8 sono le lunghezze degli array */
```

`result` vale 2 perché il pattern `101` è presente, con sovrapposizione, a partire dall'indice 2 e di nuovo dall'indice 4.

2. Pattern match (Variante B - punti 5)

Dato un array `a` di lunghezza `size`, chiamiamo k-rotazione di `a` l'array che si ottiene "spostando" gli ultimi k, con $0 \leq k <$`size` elementi dell'array nelle prime posizioni e spostando gli altri elementi alla fine dell'array. Scrivere una funzione `rotated` che dati due array di interi positivi `a` e `b` della stessa lunghezza, restituisca il valore k per il quale `b` si può ottenere tramite una k-rotazione di `a`, e restituisca `-1` se invece non esiste un k per cui `b` è una k-rotazione di `a`.

Esempio:

```c
int a[5] = { 1, 2, 3, 4, 5 };
int b[5] = { 4, 5, 1, 2, 3 };
int result = rotated(a, b, 5); /* 5 è la lunghezza degli array */
```

`result` vale 2 perché `b` è una 2-rotazione di `a`, ovvero `b` si ottiene spostando gli ultimi due elementi di `a` (`4` e `5`) all'inizio dell'array.

```c
int a[5] = { 1, 2, 3, 4, 5 };
int b[5] = { 3, 4, 5, 2, 1 };
int result = rotated(a, b, 5); /* 5 è la lunghezza degli array */
```

`result` vale -1 perché `a` non esiste una rotazione di `a` che generi `b`.

3. Jump! (Variante A - punti 5)

Dato un vettore di interi `jump`, chiamiamo visita a salti il seguente procedimento. Seleziono una posizione `i` e se `jump[i]` è uguale a `-1` mi fermo, altrimenti "salto" all'elemento in posizione `jump[i]` e ripeto il procedimento. Scrivere una funzione `max_jump` che restituisca la posizione iniziale che genera il massimo numero di salti.

Esempi:

```c
int jump[6] = { 1, -1, 3, 4, 5, 0 };
int result = max_jump(jump, 6); /* 6 è la lunghezza dell'array */
```

`result` vale 2 perché partendo dalla posizione 2 posso eseguire 5 salti: 2 → `jump`[2]=3, 3 → `jump`[3]=4, 4 → `jump`[4]=5, 5 → `jump`[5]=0, 0 → `jump`[0]=1 e mi fermo perché `jump`[1] vale -1.

```c
int jump[6] = { 1, 2, 3, -1, 5, -1 };
int result = max_jump(jump, 6); /* 6 è la lunghezza dell'array */
```

`result` vale 0 perché partendo dalla posizione 0 posso eseguire 3 salti (0 → `jump`[1]=2, 2 → `jump`[2]=3, 3 → `jump`[3]=-1), mentre tutte le altre posizioni iniziale generano percorsi più brevi.

3. Jump! (Variante B - punti 5)

Dato un vettore di interi `jump`, chiamiamo visita a salti il seguente procedimento. Seleziono una posizione `i` e se `jump[i]` è uguale a `-1` mi fermo, altrimenti "salto" all'elemento in posizione `jump[i]` e ripeto il procedimento. Siamo in presenza di un cerchio se partendo dalla posizione `i` ad un certo punto torniamo alla posizione `i`. Scrivere una funzione `circles` che conta il numero di posizioni iniziali che generano dei cerchi.

Esempio:

```c
int jump[6] = { 1, 2, 0, 4, 5, -1 };
int result = circles(jump, 6); /* 6 è la lunghezza dell'array */
```

`result` vale 3 perché partendo dalla posizione 0 ho il primo cerchio (0 → `jump`[0]=1, 1 → `jump`[1]=2, 2 → `jump`[2]=0), partendo dalla posizione 1 ho il secondo cerchio (1 → `jump`[1]=2, 2 → `jump`[2]=0, 0 → `jump`[0]=1), e infine ho il terzo cerchio partendo dalla posizione 2 (2 → `jump`[2]=0, 0 → `jump`[0]=1, 1 → `jump`[1]=2). Le altre posizioni iniziali non generano cerchi.

```c
int jump[6] = { 1, -1, 3, 4, 5, 0 };
int result = circles(jump, 6); /* 6 è la lunghezza dell'array */
```

`result` vale 0 perché non ci sono posizioni iniziali che generano dei cerchi. Tutti i percorsi possibili termineranno in posizione 1 senza tornare alla posizione iniziale.

Soluzioni Esercizi Parte B

Riempi la matrice variante A

```c
#include <stdlib.h>

void fill_a(int *m, int n, int k) {
  int row, col;
  int diff = k-1;
  for (row=0; row<n; row++)
    for (col=0; col<n; col++)
      if ( abs(col-row)<=diff )
        m[row*n + col] = 1;
      else
        m[row*n + col] = 0;
```

```
}
```

Riempi la matrice variante B

```c
#include <stdlib.h>

void fill_b(int *m, int n, int k) {
  int row, col;
  int diff = k-1;
  for (row=0; row<n; row++)
    for (col=0; col<n; col++)
      if ( abs(n-1-col-row)<=diff )
        m[row*n + col] = 1;
      else
        m[row*n + col] = 0;
}
```

Pattern match variante A

```c
int count(int *pattern, int p_size, int *a, int a_size) {
  int c = 0;
  int start;
  for (start=0; start+p_size<=a_size; start++) {
    int found = 1;
    int i;
    for (i=0; i<p_size && found; i++)
      found = found && a[start+i] == pattern[i];
    if (found)
      c++;
  }
  return c;
}
```

Pattern match variante B

```c
int rotated(int *a, int *b, int size) {
  int k;
  for (k=0; k<size; k++) {
    int found = 1;
    int i;
    for (i=0; i<size && found; i++)
      found = found && b[ (k+i)%size ] == a[i];
    if (found)
      return k;
  }
  return -1;
}
```

Jump! variante A

```c
int max_jump(int *jump, int jump_size) {
  int max_len=0, max_start;
  int i;
```

```
  for (i=0; i<jump_size; i++) {
    int j=i;
    int jumps = 0;
    while (jump[j]!=-1){
      j = jump[j];
      jumps++;
    }
    if (jumps>max_len) {
      max_len = jumps;
      max_start = i;
    }
  }
  return max_start;
}
```

Jump! variante B

```
int circles(int *jump, int jump_size) {
  int circles=0;
  int i;
  for (i=0; i<jump_size; i++) {
    int j=i;
    while (jump[j]!=-1 && jump[j]!=i){
      j = jump[j];
    }
    if (jump[j]==i) {
      circles++;
    }
  }
  return circles;
}
```

Appello del 13/01/2021

Istruzioni

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti è obbligatoria solo per chi non ha ricevuto un voto finale sufficiente nelle esercitazioni. La seconda metà è obbligatoria per tutti. Nella seconda parte, lo studente potrà scegliere tra completare la Sezione A o completare la Sezione B. Completare correttamente tutta la Sezione A consentirà di avere un voto massimo di 25, mentre completare correttamente tutta la Sezione B consentirà di avere un voto massimo pari a 30L. Non è quindi obbligatorio completare entrambe le sezioni, ad es. chi completa la Sezione B non deve completare la Sezione A.

In ogni caso é obbligatorio attenersi allo standard ANSI C.

Domande a Risposta Multipla - Per chi non ha un voto sufficiente nelle esercitazioni

Q1 Scope

Dato il seguente codice:

```c
int x = 10;
int q1(int z) {
  {
    int z = ++x + 1;
    int x = z + 1;
    return x;
  }
}
```

Qual è l'output della seguente istruzione ?

```c
  printf("%d %d \n", q1(0), q1(0));
```

- ☐ Il codice è errato per via delle doppie parentesi graffe
- ☐ Il codice è errato perché non posso creare una variabile `z` se esiste già un parametro `z`
- ☐ `13 13` perché al valore `10` della variabile globale `x` vengono applicati 3 incrementi unitari
- ☐ Nessuna delle precedenti risposte

Q2 Puntatori

Data la seguente funzione:

```c
void swap(int *x, int *y) {
  int *aux = x;
  x = y;
  y = aux;
}
```

Qual è l'output del seguente codice?

```
  int a = 10, b = 20;
  swap( &a, &b );
  swap( &b, &a );
  printf("%d %d \n", a, b);
```

- ☐ `10 20` perché la funzione `swap` scambia il contenuto delle variabili `a` e `b`, ma il doppio scambio ripristina i valori iniziali
- ☐ `10 20` perché la funzione `swap` non modifica affatto le variabili `a` e `b`
- ☐ `20 10` perché la prima invocazione di `swap` scambia il contenuto delle variabili `a` e `b` ma la seconda invocazione non ha effetto perché gli argomenti sono in ordine inverso
- ☐ il codice è errato perché la variabile `aux` dovrebbe essere di tipo intero

Q3 Array e Matrici

E' corretto dire che la seguente funzione calcola la somma degli elementi sulla prima colonna di una matrice bidimensionale?

```
int q3(int *m, int n_rows, int n_cols) {
  int res = 0;
  int *end = m + n_rows*n_cols;
  while ( m<end ) {
    res += *m;
    m += n_cols;
  }
  return res;
}
```

- ☐ Si
- ☐ No, andrebbe applicata la correzione `int *end = m + n_rows*n_cols*sizeof(int);`
- ☐ No, andrebbe applicata la correzione `while ( m<=end )`
- ☐ No, andrebbe applicata la correzione `m += n_rows;`

Q4 Loops

Cosa stampa in output la chiamata alla funzione seguente?

```
void q4() {
  int i = 100;
  while ( i-- >= 0 ) {
    printf ("%d ", --i);
  }
}
```

- ☐ I numeri pari in ordine decrescente da 98 a -2
- ☐ I numeri dispari in ordine decrescente da 99 a 1
- ☐ I numeri pari in ordine decrescente da 98 a 0
- ☐ I numeri dispari in ordine decrescente da 99 a -1

Domande a Risposta Multipla - Obbligatorie per tutti

Q5 Struct

```c
typedef struct player {
  int age;
  int* points;
  int num_matches;
} player_t;

typedef struct team {
  player_t *players;
  int num_players;
} team_t;
```

Considerando i tipi `team_t` e `player_t` definiti sopra, quale delle seguenti funzioni calcola correttamente la somma dei punteggi ottenuti da tutti i giocatori di una squadra in tutte le partite giocate?

```c
int A(team_t team) {
  int res = 0;
  int p, m;
  for (p=0; p<team->num_players; p++)
    for ( m=0; m < team->players[p].num_matches; m++)
      res += team->players[p].points[m];
  return res;
}
```

```c
int B(team_t team) {
  int res = 0;
  int p, m;
  for (p=0; p<team.num_players; p++)
    for ( m=0; m < team.players[p]->num_matches; m++)
      res += team.players[p]->points[m];
  return res;
}
```

```c
int C(team_t team) {
  int res = 0;
  int p, m;
  for (p=0; p<team.num_players; p++)
    for ( m=0; m < team.players[p].num_matches; m++)
      res += team.players[p].points[m];
  return res;
}
```

```c
int D(team_t team) {
  int res = 0;
  player_t *p;
  int *m;
  for ( p = team.players; p != team.players + team.num_players; p++)
    for ( m = p->points; m != p->points + p->num_matches; m++)
      res += *m;
  return res;
}
```

- ☐ A
- ☐ B
- ☐ C
- ☐ D

Q6 Iterazione per Ricorsione

Dire quali sono implementazioni corrette della seguente funzione ricorsiva.

$$f(n) = \begin{cases} 1 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ 2 * f(n-1) + f(n-2) & \text{altrimenti} \end{cases}$$

```c
int f1(int n) {
  if ( n==0 || n==1 ) return 1;
  return 2 * f1(n-1) + f1(n-2);
}

int f2(int n) {
  int f_, f__;
  int i, res;
  if ( n==0 || n==1 ) return 1;

  f_ = f__ = 1;
  for (i=2; i<=n; i++) {
    res = 2 * f_ + f__;
    f__ = f_;
    f_ = res;
  }

  return res;
}

int f3(int n) {
  int f_, f__;
  int i, res;
  if ( n==0 || n==1 ) return 1;

  res = f_ = 1;
  for (i=2; i<=n; i++) {
    f__ = f_;
    f_ = res;
    res = 2 * f_ + f__;
  }

  return res;
}

int f4(int n) {
  int f_, f__;
  int i, res;
  if ( n==0 || n==1 ) return 1;

  res = 3;
  f_  = 1;
  f__ = 1;
  for (i=2; i<n; i++) {
    f__ = f_;
    f_ = res;
    res = 2 * f_ + f__;
  }

  return res;
}
```

- ☐ f1
- ☐ f2

- ☐ f3
- ☐ f4

Q7 Stringhe

Claudio ha scritto una funzione per la concatenazione di due stringhe inserendo due asterischi tra le due. Ad esempio, chiamata con gli argomenti "`I Love`" e "`Coding`", la funzione dovrebbe restituire "`I Love**Coding`".

Quali errori ci sono nella funzione qui sotto?

```c
#include <string.h>
char* q7(const char *a, const char *b) {
  size_t i=0, j=0;
  char * c = (char*) malloc ( strlen(a) + strlen(b) + 3 );
  if (!c) return NULL;
  for (j=0; j<strlen(a); j++)
    c[i++] = a[j];
  c[i] = "**";
  i += 2;
  for (j=0; j<strlen(b); j++)
    c[i++] = b[j];
  return c;
}
```

- ☐ E' stata allocata più memoria del necessario
- ☐ `c[i] = "**";`
- ☐ La funzione restituisce sempre `NULL`
- ☐ La stringa non termina correttamente

Q8 Ricorsione

Assumendo che l'input `n` della funzione definita di seguito sia sempre maggiore o uguale a 0, indicare quali affermazioni sono vere:

```c
int q8(int n) {
  if (n==0)
    return 0;
  if ( n%2 == 0)
    return 1 + q8(n-1);
  else
    return n + q8(n-2);
  if ( n%3 == 0)
    return -1;
}
```

- ☐ Se `n` è un numero pari, la funzione calcola quanti sono i numeri pari maggiori di 0 fino a `n` incluso
- ☐ Se `n` è un numero dispari, la funzione calcola la somma dei numeri dispari maggiori di 0 fino a `n` incluso
- ☐ La funzione termina solo per `n` uguale a 0
- ☐ Se `n` è multiplo di 3, la funzione restituisce 1

Soluzioni Domande

1. Scope

- ☒ Nessuna delle precedenti risposte

2. Puntatori

☒ `10 20` perché la funzione `swap` non modifica affatto le variabili `a` e `b`

3. Array e Matrici

☒ Sì

4. Loops

☒ I numeri pari in ordine decrescente da 98 a -2

5. Struct

☒ C
☒ D

6. Iterazione per Ricorsione

☒ f1
☒ f2
☒ f3
☒ f4

7. Stringhe

☒ `c[i] = "**";`
☒ La stringa non termina correttamente

Versione Corretta:

```c
char* q7_ok(const char *a, const char *b) {
  size_t i=0, j=0;
  char * c = (char*) malloc ( strlen(a) + strlen(b) + 3 );
  if (!c) return NULL;
  for (j=0; j<strlen(a); j++)
    c[i++] = a[j];
  c[i++] = '*';
  c[i++] = '*';
  for (j=0; j<strlen(b); j++)
    c[i++] = b[j];
  c[i] = '\0';
  return c;
}
```

8. Ricorsione

☒ La funzione termina solo per `n` uguale a 0

Esercizi Appello

Parte A (voto 18-25)

A - Array Compare

Scrivere una funzione che, dati due array di interi `A` e `B` aventi la stessa lunghezza diversa da zero, restituisca `-1` se `A[i]<=B[i]` per tutte le posizioni `i`, restituisca `1` se `A[i]>B[i]` per tutte le posizioni `i`, e 0 altrimenti.

Esempio:

```c
int W[] = {2,4,7,6,5};
int X[] = {1,2,3,4,5};
int Y[] = {6,7,8,9,5};
int Z[] = {1,3,5,3,1};
printf("%d \n", es1( X, Y, 5) );
printf("%d \n", es1( Y, X, 5) );
printf("%d \n", es1( W, Z, 5) );
printf("%d \n", es1( X, Z, 5) );
```

Output:

```
-1
0
1
0
```

A - Marginal Sum

Scrivere una funzione che data una matrice rettangolare di interi `A` restituisca un nuovo vettore che in posizione `i`-esima contenga la somma degli elementi nell'`i`-esima colonna.

Esempio:

```c
int M1[][3] = { {1,2,3}, {4,5,6} };
int i=0;
int *v = es2(&M1[0][0], 2, 3);
for (i=0; i<3; i++) {
  printf("%d ", v[i]);
}
```

Output:

```
5 7 9
```

A - Minimal

Nell'alfabeto "minimal" al posto delle lettere maiscuole si raddoppia la corrispondente minuscola. Ad esempio "WoW" diventa "wwoww". Scrivere una funzione che data una stringa costituita solo da caratteri alfabetici e spazi restituisca la stringa corrispondente in alfabeto "minimal".

Esempio:

```c
printf("%s \n", es3("WoW") );
printf("%s \n", es3("eVViVa") );
```

Output:

```
wwoww
evvvvivva
```

A - list & list

Date due liste di elementi distinti, scrivere una funzione che restituisca `1` se tutti gli elementi della prima lista sono inclusi nella seconda lista in qualunque ordine e `0` altrimenti. La lista è definita come segue.

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```c
/* given that: l1 contains {1,2,3} and l2 contains {5,4,3,2,1} */
printf("%d \n", es4(l1, l2) );
/* given that: l1 contains {1,2,3} and l3 contains {1,2,7} */
printf("%d \n", es4(l1, l3) );
```

Output:

```
1
0
```

A - Consecutive X

Scrivere una funzione che dato un intero n, $n > 0$, ed una stringa s restituisca `1` se la stringa contiene n occorrenze consecutive del carattere "`x`", e 0 altrimenti.

Esempio:

```c
printf("%d \n", consec(2, "oxoxoxoxox") );
printf("%d \n", consec(4, "oxxxxxxxxx") );
```

Output:

```
0
1
```

Parte B (voto 18-30L)

B - Replica

Scrivere una funzione che data una matrice di interi, restituisca una nuova matrice ottenuta replicando 4 volte la matrice iniziale come nell'esempio seguente:

Input:

```
1 2 3
3 4 5
```

Output:

```
1 2 3 1 2 3
3 4 5 3 4 5
1 2 3 1 2 3
3 4 5 3 4 5
```

B - Zip

La compressione ZIP usa anche la seguente intuizione. Se posso costruire una lista di sotto-stringhe frequenti, allora posso comprimere un testo memorizzando solo una identificazione di queste sotto-stringhe. Implementare una funzione di decompressione ZIP con i seguenti parametri: stringa di codifica, array delle posizioni iniziali, array delle lunghezze. La decodifica è definita dal seguente esempio:

```c
char *enc_text = " over and";
int starts[] = {1,0};
int lens[]   = {8,5};
printf("%s \n", es6(enc_text, starts, lens, 2) );
```

Output:

```
over and over
```

La stringa decompressa si ottiene prendendo la sottostringa di `enc_text` che inizia in posizione 1 ed è lunga 8 ("`over and`""), e concatenandola con la sottostringa che inizia in posizione 0 ed è lunga 5 ("`over`""). L'ultima parametro indica il numero di elementi nei vettori `starts` e `lens`.

B - Lista Palindroma

Scrivere una funzione che, data una lista definita come segue, restituisca `1` se la lista è palindroma e 0 altrimenti.

Definizione di lista:

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

B - Fred the Frog

La rana Fred deve percorrere esattamente n metri ($n \geq 0$) e può moversi a salti di $1, 2$ o 3 metri con un vincolo: non può mai fare due salti consecutivi della stessa lunghezza. Scrivere una funzione ricorsiva che dato n calcoli il numero di modi possibili in cui è possibile percorrere esattamente gli n metri senza superarli.

Esempio:

- $n = 0$. Esiste un solo modo: stare fermo perché siamo già a destinazione.
- $n = 2$. Esiste un solo modo con un salto lungo 2. Non è possibile fare 2 salti consecutivi lunghi 1.
- $n = 4$. : Ci sono 3 modi: $(1, 2, 1)$, $(1, 3)$, $(3, 1)$

B - x-n-x

Chiamiamo "x-n-x" la stringa fatta da una "`x`", n volte "`o`" e di nuovo "`x`". Ad esempio "x-4-x" è la stringa "xoooox". Scrivere una funzione che data una stringa s restituisca il valore di n, $n > 0$, più grande per cui esiste in s la sottostringa "x-n-x". La funzione restituisce `0` se la sottostringa non viene mai trovata.

Esempio:

```c
printf("%d \n", xnx("oxoxoooxoxo") );
printf("%d \n", xnx("ooxxxxoo") );
```

Output:

```
3
0
```

Soluzioni Esercizi

A - Array Compare

```c
int es1(int *A, int *B, int size) {
  int A_is_lesser  = 1;
```

```
  int A_is_greater = 1;
  int i;
  for (i=0; i<size; i++) {
    A_is_lesser  = A_is_lesser  && A[i]<=B[i];
    A_is_greater = A_is_greater && A[i]>B[i];
    if (!A_is_lesser && !A_is_greater)
      return 0;
  }
  if (A_is_greater)
    return 1;
  else
    return -1;
}
```

A - Marginal Sum

```
int* es2(int *A, int n_rows, int n_cols) {
  int* v = (int*) malloc( n_cols*sizeof(n_cols) );
  int i, j;
  if (!v) return NULL;
  for (i=0; i<n_cols; i++) {
    v[i] = 0;
    for (j=0; j<n_rows; j++)
      v[i] += A[j*n_cols + i];
  }
  return v;
}
```

A - Minimal

```
char* es3(const char *s) {
  char* ss = (char*) malloc(strlen(s)*2 +1); /* max possible length */
  int i=0, len = 0;
  while (i<strlen(s)) {
    if ( s[i]>='A' && s[i]<='Z' ) { /* upper case */
      char new_letter = s[i] - 'A' + 'a';
      ss[len++] = new_letter;
      ss[len++] = new_letter;
    } else
      ss[len++] = s[i];
    i++;
  }
  ss[len++] = '\0';
  ss = realloc (ss, len);
  return ss;
}
```

A - List & List

```
int is_present(list_t *l, int x) {
  while (l!=NULL) {
    if (l->data==x)
      return 1;
    l = l->next;
  }
```

```
  return 0;
}
int es4(list_t *l1, list_t *l2) {
  while (l1!=NULL) {
    if ( !is_present(l2, l1->data) )
      return 0;
    l1 = l1->next;
  }
  return 1;
}
```

A - Consecutive X

```
int consec(int n, const char *s){
  int i, j;
  int l = strlen(s);
  int count = 0;
  for (i=0; i<l; i++) {
    if (s[i]=='x') {
      count += 1;
      if (count==n) return 1;
    } else
    count = 0;
  }
  return 0;
}
```

B - Replica

```
int* es5 (int* M, int n_rows, int n_cols) {
  int mi, mj;
  int* Q = (int*) malloc ( 4*n_rows*n_cols * sizeof(int) );
  if (!Q) return NULL;
  for (mi=0; mi<n_rows; mi++)
    for (mj=0; mj<n_cols; mj++) {
      Q[ mi*2*n_cols + mj ]        = Q[ (mi+n_rows)*2*n_cols + mj ] =
      Q[ mi*2*n_cols + mj+n_cols ] = Q[ (mi+n_rows)*2*n_cols + mj + n_cols ] =
      M[ mi*n_cols + mj];
    }
  return Q;
}
```

B - Zip

```
char* es6(char* enc, int* starts, int* lens, int size) {
  char *s = (char*) malloc(1);
  int i, j, l=0;
  for (i=0; i<size; i++) {
    s = realloc( s, l+lens[i]+1 );
    for (j=0; j<lens[i]; j++)
      s[l++] = enc[ starts[i]+j ];
  }
  s[l] = '\0';
  return s;
}
```

B - Lista palindroma

```c
int es7(list_t *l) {
  list_t* fwd_l = l;
  list_t* bwd_l = NULL;
  /* reverse the input list */
  while ( fwd_l!=NULL ) {
    list_t* n = (list_t*) malloc(sizeof(list_t));
    n->data = fwd_l->data;
    n->next = bwd_l;
    bwd_l = n;
    fwd_l = fwd_l->next;
  }
  /* check equality */
  fwd_l = l;
  while (fwd_l!=NULL) {
    if (bwd_l->data != fwd_l->data)
      return 0;
    fwd_l = fwd_l->next;
    bwd_l = bwd_l->next;
  }
  return 1;
}
```

B - Fred the Frog

```c
int frog(int n, int prev) {
  int count = 0;
  int jump;
  if (n<0)
    return 0;
  if (n==0)
    return 1;

  for (jump=1; jump<=3; jump++) {
    if (jump!=prev)
      count += frog(n-jump, jump);
  }

  return count;
}

int es8(int n) {
  return frog(n, 0);
}
```

B - x-n-x

```c
int is_xnx(int n, const char *s) {
  int i;
  if (strlen(s)<n+2) return 0;

  if ( s[0]!='x' ) return 0;
  for (i=1; i<=n; i++)
    if ( s[i] != 'o' ) return 0;
  if ( s[n+1]!='x' ) return 0;
```

```
  return 1;
}

int xnx(const char *s) {
  int l = strlen(s);
  int i, n;
  for (n=l; n>0; n--) {
    for (i=0; i<l; i++) {
      if (is_xnx(n,s+i))
        return n;
    }
  }
  return 0;
}
```

Appello del 28/01/2021

Istruzioni

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti è obbligatoria solo per chi non ha ricevuto un voto finale sufficiente nelle esercitazioni. La seconda metà è obbligatoria per tutti. Nella seconda parte, lo studente potrà scegliere tra completare la Sezione A o completare la Sezione B. Completare correttamente tutta la Sezione A consentirà di avere un voto massimo di 25, mentre completare correttamente tutta la Sezione B consentirà di avere un voto massimo pari a 30L. Non è quindi obbligatorio completare entrambe le sezioni, ad es. chi completa la Sezione B non deve completare la Sezione A.

In ogni caso é obbligatorio attenersi allo standard ANSI C.

Domande a Risposta Multipla - Per chi non ha un voto sufficiente nelle esercitazioni

Q1 Scope

Dato il seguente codice:

```c
int x = 10;
int f1(int z) {
  int res = z++;
  {
    int z = 7;
    res = z;
  }{
    int w = z;
    res = z * 2;
  }
  return z;
}
```

Qual è l'output della seguente istruzione ?

```c
  printf("%d \n", f1(++x) );
```

- ☐ 12
- ☐ 14
- ☐ 11
- ☐ Il codice non compila per via della scrittura `}{`

Q2 Puntatori

Data la seguente funzione:

```c
void f2(int *x, int *y, int *z) {
  x = y;
  y = z;
```

```
  z = x;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
  int a = 1, b = 2, c = 3;
  f2(&a, &b, &c);
  printf("%d %d %d \n", a, b, c);
```

- ☐ 2 3 1 perché la funzione `f2` copia b in a, c in b e a in c
- ☐ 1 2 3 perché vengono modificati i puntatori alle variabili `a`, `b`, `c` ma non i valori che questi riferiscono
- ☐ 2 3 2 perché quando `f2` copia a in c, la variabile a non contiene più il valore 1 ma il valore 2
- ☐ 1 2 3 perché le variabili `a`, `b`, `c` non compaiono mai nella funzione `f2`

Q3 Array e Matrici

E' corretto dire che la seguente funzione calcola la somma degli elementi sulla diagonale principale matrice quadrata? In figura marcati con un asterisco gli elementi della diagonale principale di una matrice 4x4.

```
+----+
|*   |
| *  |
|  * |
|   *|
+----+
```

```
int f3(int *m, int n_rows) {
  int res = 0;
  int i,j;
  for (i=0; i<n_rows; i++) {
    for (j=0; j<n_rows; j++) {
      if (i==j)
        res += m[i][j];
    }
  }
  return res;
}
```

- ☐ Si
- ☐ No, andrebbe applicata la correzione `res += m[i*n_rows + j];`
- ☐ No, nei cicli for andrebbe applicata la correzione `i<=n_rows` e `j<=n_rows`
- ☐ No, e in ogni caso sarebbe stato possibile utilizzare un solo ciclo for

Q4 Loops

Cosa stampa in output la chiamata alla funzione seguente?

```
void f4() {
  int i = 0, j=10;
  while ( i <= 100 )
    for (j=i; j<=100; j++)
      if ( (i+j) % 51 == 0)
        printf ("(%d + %d) \n", i, j);

}
```

- ☐ tutte le coppie di numeri interi compresi tra 0 e 100 la cui somma è pari
- ☐ il codice cicla all'infinito

- ☐ tutte le coppie di numeri interi compresi tra 0 e 100 la cui somma è multiplo di 51
- ☐ tutte le coppie di numeri interi compresi tra 0 e 100 la cui somma è multiplo di 51, ma con ripetizioni: es (50+1) e (1+50)

Domande a Risposta Multipla - Obbligatorie per tutti

Q5 Struct

```c
struct a {
  int *info;
};

struct b {
  struct a *info;
};

struct c {
  struct b *info;
};
```

Considerando i tipi definiti sopra, quale delle seguenti funzioni compila correttamente senza errori e senza warning?

```c
int fun_A(struct c *x) {
  return x->info.info.info[0];
}
```

```c
int fun_B(struct c *x) {
  return *( x->info->info->info );
}
```

```c
int fun_C(struct c *x) {
  return x->info->info->info;
}
```

```c
int fun_D(struct c *x) {
  return x->info->info->info[0];
}
```

- ☐ `fun_A`
- ☐ `fun_B`
- ☐ `fun_C`
- ☐ `fun_D`

Q6 Iterazione

Quali delle seguenti funzioni restituisce il valore `1` se l'array di interi in input contiene solo numeri pari e restituisce `0` altrimenti? *Se l'array in input è vuoto la funzione deve restituire il valore 1.*

```c
int fun_E(int *v, int size) {
  int i=0;
  int pari = 1;
  for (i=0; i<size; i++)
    if (v[i]%2==0)
      pari = 1;
```

```
    else
      pari = 0;
  return pari;
}
```

```
int fun_F(int *v, int size) {
  int i=0;
  for (i=0; i<size; i++)
    if (v[i]%2!=0)
      return 0;
  return 1;
}
```

```
int fun_G(int *v, int size) {
  int i=0;
  if (size==0)
    return 1;

  return ( v[i]%2==0 ) && fun_G(v+1, size-1);
}
```

```
int fun_H(int *v, int size) {
  int i=0;
  int pari = 1;
  for (i=0; i<size; i++)
    pari = pari && (v[i]%2==0);
  return pari;
}
```

- ☐ `fun_E`
- ☐ `fun_F`
- ☐ `fun_G`
- ☐ `fun_H`

Q7 Stringhe

Claudio ha scritto una funzione che data una stringa, ogni volta che vengono individuati due caratteri uguali consecutivi questi vengono sostituiti con un doppio sharp. Ad esempio, chiamata con argomento la stringa “bugs bunny is a rabbit”, queste viene modificata in “bugs bu##y is a ra##it”.

Quali errori ci sono nella funzione qui sotto?

```
void sharp(char* s) {
  while ( *s++ != '\0' ) {
    if ( *(s+1) == *s )
      *(s+1) = *s = '#';
  }
}
```

- ☐ Si potrebbe verificare un accesso oltre la fine della stringa all’istruzione `while ( *s++ != '\0'){`
- ☐ Si potrebbe verificare un accesso oltre la fine della stringa all’istruzione `*(s+1)== *s`
- ☐ Il ciclo while non termina mai
- ☐ `*(s+1)` dovrebbe essere sostituito con `*(s-1)` dappertutto

Q8 Ricorsione

Dato il codice seguente, quali affermazioni sono vere?

```
int f8(int *a, int a_size) {
  if (a_size==0)
```

```
3     return 0;
4
5   if (a_size%2==0)
6     return *a + f8(a+2,a_size-2);
7   else
8     return *a + f8(a+2,a_size-2);
9 }
```

- ☐ se il vettore in input ha lunghezza pari, la funzione calcola la somma degli elementi con indice pari
- ☐ se il vettore in input ha lunghezza dispari, la funzione calcola la somma degli elementi con indice dispari
- ☐ se il vettore in input ha lunghezza dispari, la funzione non termina
- ☐ non è consentito avere due chiamate ricorsive nella stessa funzione

Soluzioni Domande

1. Scope

- ☒ `12`

2. Puntatori

- ☒ `1 2 3` perché vengono modificati i puntatori alle variabili `a`, `b`, `c` ma non i valori che questi riferiscono

3. Array e Matrici

- ☒ No, andrebbe applicata la correzione `res += m[i*n_rows + j];`
- ☒ No, e in ogni caso sarebbe stato possibile utilizzare un solo ciclo for

4. Loops

- ☒ Il codice cicla all'infinito

5. Struct

- ☒ `fun_B`
- ☒ `fun_D`

6. Iterazione

- ☒ `fun_F`
- ☒ `fun_G`
- ☒ `fun_H`

7. Stringhe

- ☒ Si potrebbe verificare un accesso oltre la fine della stringa all'istruzione `*(s+1)== *s`
- ☒ `*(s+1)` dovrebbe essere sostituito con `*(s-1)` dappertutto

8. Ricorsione

- ☒ Se il vettore in input ha lunghezza pari, la funzione calcola la somma degli elementi con indice pari
- ☒ Se il vettore in input ha lunghezza dispari, la funzione non termina

---

Esercizi Appello

Parte A (voto 18-25)

A - Array

Scrivere una funzione `dot` che, dati due array di float `A` e `B` aventi la stessa lunghezza diversa da zero, restituisca il prodotto interno dot tra i due vettori calcolato come segue:

$$dot(A, B) = \sum_i A[i] \cdot B[i]$$

Esempio:

```
float A[] = { 1.0f, 2.0f, 0.0f, 0.0f, 5.0f };
float B[] = { 2.0f, 1.0f, 3.0f, 0.0f, 1.0f };

printf("%f \n", dot( A, B, 5) );
```

Output:

```
9.0  /* perché 1*2 + 2*1 + 0*3 + 0*0 + 5*1 = 9 */
```

A - MaxCols

Scrivere una funzione che data una matrice di interi rettangolare `A` restituisca un nuovo vettore che in posizione `i`-esima contenga il massimo degli elementi nella `i`-esima colonna della matrica `A`.

Esempio:

```
int i;
int M[3][5] = { {1,2,33,4,5},
                {4,5,6,10,20},
                {11,12,13,14,15} };
int *m = maxcols(&M[0][0], 3, 5);
for (i=0; i<5; i++)
  printf("%d ", m[i]);
```

Output:

```
11 12 33 14 20
```

A - X factor

“x factor” è un nuovo metodo di cifrare una stringa. La decodifica è semplice:

1. per ogni occorrenza del carattere `x` (minuscolo) viene eliminata la `x` e il carattere successivo
2. ma nel caso di due `x` consecutive, allora viene tenuta una sola delle due
3. in tutti gli altri casi non si effettuano modifiche.

Scrivere una funzione che data una stringa restituisce una nuova stringa con la sua decodifica.

Esempio:

```c
char *s = "xzxyexxcerptxo";
printf("%s ", decode(s) );
```

Output:

```
excerpt
```

A - Longest sub-List

Scrivere una funzione che data una lista di interi (sia positivi che negativi) restituisca la lunghezza della sotto-lista più lunga costituita solo da interi positivi strettamente maggiori di 0. La lista è definita come segue.

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```c
/* given that: l1 contains {-1, 1, -10, 2, 3, 4, 5} */
printf("%d \n", longest(l1) );
/* given that: l2 contains {5, -3, 5, 2, 1, 4, 5, -7, -8} */
printf("%d \n", longest(l2) );
```

Output:

```
4   /* la sotto-lista è:  2, 3, 4, 5 */
5   /* la sotto-lista è:  5, 2, 1, 4, 5 */
```

Parte B (voto 18-30L)

B - Trasposta

Scrivere una funzione che data una matrice di interi, restituisca una nuova matrice ottenuta ottenuta trasponendo la prima, ovvero invertendo righe e colonne come nell'esempio:

Input:

```
1 2 3
4 5 6
```

Output:

```
1 4
2 5
3 6
```

B - Kick out

Dato un array di array di interi (`int**`) e un vettore con le lughezze degli array, l'operazione di `kick` inserisce il valore `0` all'inizio del primo array e fa slittare tutti gli altri elementi eliminando però l'ultimo elemento dell'ultimo array come nell'esempio:

Esempio:

```c
  int A0[] = {10,20,30};
  int A1[] = {40};
  int A2[] = {50, 60};
  int lens[] = {3,1,2}; /* Lunghezze degli array A0, A1, A2 */
  int *AA[3];
```

```
  AA[0] = A0;
  AA[1] = A1;
  AA[2] = A2;
  kick(AA, 3, lens);
```

Prima della chiamata, l'array `AA` contiene i tre array di seguito:

```
10 20 30
40
50 60
```

Dopo la chiamata, l'array `AA` deve contenere i tre array di seguito:

```
0 10 20
30
40 50
```

B - Sottolista pesante

Data una lista di interi (sia positivi che negativi) definiamo il suo peso come la somma dei suoi elementi. Il peso di una lista vuota è pari a 0. Scrivere una funzione che data una lista di interi restituisca il peso della sotto-lista più pesante. La lista è definita come segue.

```
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```
/* given that: l1 contains {-1, 1, -10, 2, 3, 4, 5} */
printf("%d \n", pesante(l1) );
/* given that: l2 contains {5, -3, 5, 2, 1, 4, 5, -7, -8} */
printf("%d \n", pesante(l2) );
```

Output:

```
14   /* è il peso della sotto-lista:  2, 3, 4, 5 */
19   /* è il peso della sotto-lista:  5, -3, 5, 2, 1, 4, 5 */
```

B - Lego Railway

Abbiamo a disposizione binari di diversa lunghezza per comprire un dato percorso con un nuovo tratto di binari. Scrivere una funzione che date le lunghezze disponibili `lens` e il numero di binari per ciascuna lunghezza `counts` calcoli il numero di modi possibili per coprire esattamente la lunghezza desiderata `target`.

Esempio:

```
  int target;
  int n = 4;
  int lens[]   = {  1, 3, 7, 10};
  int counts[] = { 10, 5, 4,  5};
  /* abbiamo 4 tipi di binari diversi */
  /* 10 binari di lunghezza 1, 5 di lunghezza 3,
     4 di lunghezza 7 e 5 di lunghezza 10 */
  target = 3;
  printf("%d \n", lego(lens, counts, n, target));
  target = 8;
  printf("%d \n", lego(lens, counts, n, target));
```

Output:

```
2   /* ci sono 2 modi di raggiungere il target 3:
       con 3 binari lunghi 1, e con 1 binario lungo 3 */
4   /* ci sono 4 modi di raggiungere il target 8:
       con 8 binari lunghi 1, con 5 binari lunghi 1 e 1 lungo 3,
       con 2 binari lunghi 1 e 2 lunghi 3,
       con 1 binario lungo 1 e 1 lungo 7    */
```

Soluzioni Esercizi

A - Array

```
float dot(float* A, float* B, int size) {
  float sum = 0.0f;
  int i;
  for (i=0; i<size; i++)
    sum += A[i]*B[i];
  return sum;
}
```

A - MaxCols

```
int * maxcols(int* M, int n_rows, int n_cols) {
  int i,j;
  int *m = (int*) malloc( n_cols*sizeof(int) );
  if (!m) exit(EXIT_FAILURE);
  for (j=0; j<n_cols; j++)
    m[j] = M[0*n_cols + j];
  for (j=0; j<n_cols; j++)
    for (i=1; i<n_rows; i++)
      m[j] = M[i*n_cols + j] < m[j] ? m[j] : M[i*n_cols + j] ;
  return m;
}
```

A - X-Facor

```
char * decode(char *s) {
  char *t = malloc(strlen(s)*sizeof(char)+1);
  char *src = s, *dst = t;
  while (*src!='\0') {
    if ( *src == 'x' && *(src+1) =='x') {
      *dst = 'x';
      dst++;
      src += 2;
    } else if ( *src == 'x' && *(src+1) !='x') {
      src += 2;
    } else {
      *dst = *src;
      src++;
      dst++;
    }
  }
  *dst = '\0';
  return t;
```

```
}
```

A - Longest sub-list

```
int longest ( list_t *l ) {
  int best = 0;
  int count = 0;
  while ( l!=NULL ) {
    if ( l->data>0 ) {
      count++;
      best = count>best ? count : best;
    } else
      count = 0;
    l = l->next;
  }
  return best;
}
```

B - Trasposta

```
int * transpose(int *M, int n_rows, int n_cols) {
  int *T = (int*) malloc(n_rows*n_cols*sizeof(int));
  int i,j;
  for (i=0; i<n_rows; i++) {
    for (j=0; j<n_cols; j++)
      T[j*n_rows + i] = M[i*n_cols + j];
  }
  return T;
}
```

B - Kick out

```
void kick(int **AA, int AA_size, int *lens) {
  int i, j;
  for (i=AA_size-1; i>0; i--) {
    for (j=lens[i]-1; j>0; j--)
      AA[i][j] = AA[i][j-1];
    AA[i][0] = AA[i-1][ lens[i-1]-1 ];
  }
  for (j=lens[0]-1; j>0; j--)
    AA[0][j] = AA[0][j-1];
  AA[0][0] = 0;
}
```

B - Sottolista pesante

```
int pesante(list_t *l) {
  list_t *start = l;
  list_t *end;
  int res;
  if (l==NULL)
    return 0;

```

```
  res = l->data;
  while (start!=NULL) {
    int peso = 0;
    end = start;
    while (end!=NULL) {
      peso += end->data;
      if (peso>res) res = peso;
      end = end->next;
    }
    start = start->next;
  }
  return res;
}
```

B - Lego Railway

```
int lego(int *lens, int *counts, int n, int target) {
  int i;
  int c = 0;

  if (target==0) /* trovato */
    return 1;

  if (target<0 || n==0) /* non trovato */
    return 0;

  for (i=0; i<=counts[0]; i++) {
    c += lego(lens+1, counts+1, n-1, target-i*lens[0]);
  }
  return c;
}
```

Appello del 15/06/2021

Istruzioni

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti è obbligatoria solo per chi non ha ricevuto un voto finale sufficiente nelle esercitazioni. La seconda metà è obbligatoria per tutti. Nella seconda parte, lo studente potrà scegliere tra completare la Sezione A o completare la Sezione B. Completare correttamente tutta la Sezione A consentirà di avere un voto massimo di 25, mentre completare correttamente tutta la Sezione B consentirà di avere un voto massimo pari a 30L. Non è quindi obbligatorio completare entrambe le sezioni, ad es. chi completa la Sezione B non deve completare la Sezione A.

In ogni caso é obbligatorio attenersi allo standard ANSI C.

Domande a Risposta Multipla - Per chi non ha un voto sufficiente nelle esercitazioni

Q1 Scope

Dato il seguente codice:

```c
int x = 0;
int f1(int w) {
  int z = w + x++;
  {
    int z = w + x++;
  }
  return z;
}
```

Qual è l'output della seguente istruzione ?

```c
  printf( "%d %d\n", f1(10), f1(10) );
```

- ☐ 10 10
- ☐ 10 12
- ☐ 11 13
- ☐ 12 14

Q2 Puntatori

Data la seguente funzione:

```c
void f2(int a1, int *a2, int *a3) {
  a1++;
  *(++a2);
  (*a3)++;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
  int a = 0, b = 0;
  f2(a, &a, &a);
  printf("%d %d\n", a, b);
```

- ☐ `3 0` perché la funzione `f2` incrementa `a` tre volte.
- ☐ `1 1` perché la funzione `f2` incrementa `b` nel secondo statement *spostando* l'indirizzo `a2` e in seguito incrementa `a` nel terzo.
- ☐ `0 0` perché la funzione `f2` incrementa delle copie di `a` ma non la locazione di memoria originaria.
- ☐ `1 0` perché la funzione `f2` incrementa `a` solo nel terzo statement.

Q3 Array e Matrici

E' corretto dire che la seguente funzione calcola la somma degli elementi in ciascuna colonna della matrice quadrata `m` in input?

```
void f3(int *m, int n_rows, int *s) {
  int i,j;
  for (i=0; i<n_rows; i++) {
    s[i] = 0;
    for (j=0; j<n_rows; j++) {
      s[i] += m[i*n_rows + j];
    }
  }
}
```

- ☐ No, la funzione calcola la somma degli elementi di ogni riga.
- ☐ Non si può dire. Dipende dai valori presenti nella matrice quadrata `m`.
- ☐ Si.
- ☐ Si, anche se lo stesso risultato poteva essere ottenuto con un solo ciclo **`for`**.

Q4 Loops

Un numero intero si dice *quadrato perfetto* se la sua radice quadrata è un numero intero. Ad esempio 9 è un quadrato perfetto perché la sua radice quadrata è 3, mentre 10 non è un quadrato perfetto perché la sua radice quadrata è 3.16.

Selezionare le affermazioni vere in merito al codice seguente.

```
void f4() {
  int i = 0, j = 0, N = 100;
  for (i=0; i<N; i++) {
    for (j=0; j<=i; j++){
      if (i*i==j)
        printf ("%d \n", i);
    }
  }
}
```

- ☐ la funzione stampa tutti i quadrati perfetti minori di 100.
- ☐ la funzione stampa tutti i numeri minori di 100 che non sono quadrati perfetti.
- ☐ la funzione sarebbe corretta se scambiassimo i due cicli **`for`**
- ☐ la funzione sarebbe corretta se correggessimo la condizione del comando **`if`**

Domande a Risposta Multipla - Obbligatorie per tutti

Q5 Struct

```c
struct card {
  int value; // 1,2,3, ...
  int suit;  // hearts, diamonds, ...
};

struct deck {
  struct card *cards;
  int num_cards;
};
```

Considerando i tipi definiti sopra, selezionare le funzioni che implementano correttamente lo scambio tra la prima e l'ultima carta del mazzo (*deck*)?

```c
void fun_A(struct deck *d) {
  struct card aux;
  aux = d->cards[0];
  d.cards[0] = d.cards[d.num_cards-1];
  d.cards[d.num_cards-1] = aux;
}
```

```c
void fun_B(struct deck *d) {
  struct card *aux;
  aux = d->cards[0];
  d->cards[0] = d->cards[d->num_cards-1];
  d->cards[d->num_cards-1] = aux;
}
```

```c
void fun_C(struct deck *d) {
  struct card aux;
  aux = d->cards[0];
  d->cards[0] = d->cards[d->num_cards-1];
  d->cards[d->num_cards-1] = aux;
}
```

```c
void fun_D(struct deck *d) {
  struct card *aux;
  aux = &( d->cards[0] );
  d->cards[0] = d->cards[d->num_cards-1];
  d->cards[d->num_cards-1] = *aux;
}
```

- ☐ `fun_A`
- ☐ `fun_B`
- ☐ `fun_C`
- ☐ `fun_D`

Q6 Iterazione

Quali delle seguenti funzioni restituisce il valore `1` se l'array di interi in input contiene valori in ordine strettamente crescente e restituisce `0` altrimenti? *Se l'array in input è vuoto la funzione deve restituire il valore 1.*

```c
int fun_E(int *v, int size) {
  int i=0;
  int increasing = 1;
  for (i=0; i<size-1 && increasing; i++)
    if (! (v[i]<v[i+1]) )
      increasing = 0;
  return increasing;
```

```
}
```

```
int fun_F(int *v, int size) {
  int i=0;
  int increasing = 1;
  for (i=0; i<size-1; i++)
    if (v[i]<v[i+1])
      increasing = 1;
    else
      increasing = 0;
  return increasing;
}
```

```
int fun_G(int *v, int size) {
  int i=0;
  int increasing = 1;
  for (i=0; i<size-1; i++) {
    increasing = increasing && (v[i]<v[i+1]);
    if (increasing)
      return 1;
  }
  return increasing;
}
```

```
int fun_H(int *v, int size) {
  int i=0;
  for (i=0; i<size-1; i++)
    if (v[i]>=v[i+1])
      return 0;
  return 1;
}
```

- ☐ `fun_E`
- ☐ `fun_F`
- ☐ `fun_G`
- ☐ `fun_H`

Q7 Stringhe

Claudio ha scritto una funzione che data una stringa `s` ed un intero `n`, restituisce una nuova stringa ottenuta replicando `n` volte la stringa `s`. Ad esempio, chiamata con argomenti la stringa `TicToc` e il valore `3`, viene restituita la nuova stringa `TicTocTicTocTicToc`.

Quali errori ci sono nella funzione qui sotto?

```
char* concat(char* s, int n) {
  int j;
  int new_l = strlen(s)*n;
  char *new_s = (char*) malloc (new_l);
  char *p = new_s;
  if (!new_s) return NULL;
  for (j=0; j<n; j++) {
    char* q = s;
    while(*(q++)!='\0') {
      *(p++) = *q;
    }
  }
  new_s[new_l-1] = '\0';
  return new_s;
}
```

- ☐ l'istruzione `new_s[new_l-1] = '\0';` andrebbe sostituita con `new_s[new_l] = '\0';`.
- ☐ Non viene allocata sufficiente memoria per memorizzare la nuova stringa.
- ☐ l'istruzione `*(q++)!=NULL` incrementa `q` anticipatamente.
- ☐ l'istruzione `*(p++)` incrementa `p` anticipatamente.

Q8 Ricorsione

Dato il codice seguente, quali affermazioni sono vere?

```c
int f8(int *a, int a_size) {
  if (a_size==0)
    return 1;

  if (a_size%2==0)
    return (*a == *(a+a_size-1)) && f8(a+1,a_size-2);
  else
    return f8(a+1,a_size-2);
}
```

- ☐ se il vettore in input ha lunghezza dispari, la funzione non termina.
- ☐ se il vettore in input ha lunghezza pari, non tutti gli elementi del vettore in input vengono letti al termine della ricorsione.
- ☐ se il vettore in input ha lunghezza pari, verifica se il vettore è palindromo.
- ☐ se il vettore in input ha lunghezza dispari, la funzione restituisce 1.

Soluzioni Domande

1. Scope

- ☒ `10 12`

2. Puntatori

- ☒ `1 0` perché la funzione `f2` incrementa `a` solo nel terzo statement.

3. Array e Matrici

- ☒ No, la funzione calcola la somma degli elementi di ogni riga.

4.Loops

- ☒ la funzione sarebbe corretta se correggessimo la condizione del comando **`if`**

5. Struct

- ☒ `fun_C`

6. Iterazione

- ☒ `fun_E`
- ☒ `fun_H`

7. Stringhe

- ☒ Non viene allocata sufficiente memoria per memorizzare la nuova stringa.
- ☒ l'istruzione `*(q++)!=NULL` incrementa `q` anticipatamente.

8. Ricorsione

- ☒ se il vettore in input ha lunghezza dispari, la funzione non termina.
- ☒ se il vettore in input ha lunghezza pari, verifica se il vettore è palindromo.

Esercizi Appello

Parte A (voto 18-25)

A - Reverse

Scrivere una funzione `is_reverse` che, dati due array di interi `A` e `B` aventi la stessa lunghezza diversa da zero, restituisca il valore `1` se il primo array contiene gli stessi elementi del secondo in ordine inverso, e restituisca `0` altrimenti

Esempio:

```c
int A[] = { 7, 9, 12, 4, 21 };
int B[] = { 21, 4, 12, 9, 7 };
int C[] = { 21, 4, 3333333, 9, 7 };

printf("%d \n", is_reverse( A, B, 5) );
printf("%d \n", is_reverse( A, C, 5) );
```

Output:

```
1
0
```

A - Dominant

Chiamiamo una matrice quadrata `A` dominante se per ogni colonna l'elemento massimo della colonna stessa si trova sulla diagonale principale. Scrivere una funzione che data una matrice `A` restituisce `1` se questa è dominante e `0` altrimenti.

Esempio:

```c
int i;
int M[4][4] = { {20, 2,  33, 4 },
                {4,  25, 30,  10},
                {11, 12, 44, 14},
                {2,  8,  12, 15}, };
            //   20, 25, 44, 155 sono i massimi per colonna
            //   e si trovano tutti sulla diagonale principale
printf("%d ", dominant( &(M[0][0]), 4, 4) );
```

Output:

```
1
```

A - Password generator

Per memorizzare facilmente una password, definiamo un algoritmo di trasformazione di una stringa `s` come segue:

1. leggo la stringa `s` da sinistra a destra
2. ogni volta che incontro una vocale, questa viene rimossa e copiata in coda alla stringa iniziale

Scrivere una funzione che data una stringa restituisce il risultato della trasformazione. *(assumiamo di avere a che fare solo con vocali minuscole).*

Esempio:

```c
char *s = "chiave_segreta";
printf("%s ", pwd(s) );
```

Output:

```
chv_sgrtiaeeea
```

A - Sottolista fortunata

Scrivere una funzione che data una lista di interi (sia positivi che negativi) restituisca il valore `1` se contiene una sotto-lista i cui elementi hanno somma pari a 13, e restituisca `0` altrimenti.

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```c
/* given that: l1 contains {-1, 1, 10, 2, 3, 4, 5} */
printf("%d \n", lucky(l1) );
/* given that: l2 contains {1, -3, 1, -2, 1, 4, -5, 7, 0} */
printf("%d \n", lucky(l2) );
```

Output:

```
1   /* la sotto-lista è:  1, 10, 2 */
0   /* non esiste una sottolista */
```

Parte B (voto 18-30L)

B - Scacchi

Scrivere una funzione che data una matrice quadrata con lato di dimensione pari, restituisca la somma degli elementi nelle caselle nere della ideale scacchiera della stessa dimensione come nell'esempio:

Input:

```
                +--------+
 1  2  3  4     |##  ##  |
 5  6  7  8     |  ##  ##|
 9 10 11 12     |##  ##  |
13 14 15 16     |  ##  ##|
                +--------+
```

Output:

```
68  /* 1 + 3 + 6 + 8 + 9 + 11 + 14 + 16 */
```

B - Catena

Un array di stringhe forma una catena se gli ultimi `n` caratteri di ciascuna stringa sono uguali a primi `n` della successiva. Scrivere una funzione che dato un array di stringhe e il valore `n` restituisce `1` in presenza di una catena e `0` altrimenti.

Esempio:

```
char* a[] = {"nave", "vela", "latta"};
char* b[] = {"nave", "avena"};
printf("%d\n", catena(a, 3, 2) );
printf("%d\n", catena(b, 2, 3) );
```

Output:

```
1   /* na-ve -> ve-la -> la-tta */
1   /* n-ave -> ave-na */
```

B - Ripetizioni

Date due liste di interi scrivere una funzione per conteggiare il numero di occorrenze della prima lista come sottolista della seconda, anche con sovrapposizioni.

La struttura dati lista è definita come segue:

```
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```
/* given that: l1 contains {1,2,1} */
/*         and l2 contains {0,1,2,1,2,1,0,1,2,1} */
printf("%d \n", reps(l1, l2) );
```

Output:

```
3  /* che iniziano in posizione 1, 3 e 7 di l2 */
```

B - Staccionata

Abbiamo una scorta infinita di pali di legno di 4 colori diversi con i quali dobbiamo costruire una staccionata composta da `n` pali in tutto, con un solo vincolo: due pali dello stesso colore devono avere almeno altri due pali di colore diverso a separarli. Scrivere una funzione che dato `n` restituisca il numero di configurazioni possibili.

Esempio:

```
printf("%d \n", fence(1) );
printf("%d \n", fence(4) );
```

Output:

```
4
48
/* se chiamiamo i colori a,b,c,d.

  nel primo caso le configurazioni sono:
    a, b, c, d.
  nel secondo caso le configurazioni sono:
    abca, abcd, abda, abdc,
    acba, acbd, acda, acdb,
```

```
    adba, adbc, adca, adcb, ...
    ... e così via con sequenze simil iche iniziano per b, c, d.
```

Soluzioni Esercizi

A - Reverse

```c
int is_reverse(int* A, int* B, int size) {
  int i;
  for (i=0; i<size; i++)
    if (A[i] != B[size-i-1])
      return 0;
  return 1;
}
```

A - Dominant

```c
int dominant(int* M, int n_rows, int n_cols) {
  int i,j;
  for (i=0; i<n_cols; i++) {
    for (j=0; j<n_rows; j++)
      if ( M[i*n_cols + i] < M[j*n_cols + i] )
        return 0;
  }
  return 1;
}
```

A - Password Generator

```c
char * pwd(char *s) {
  char *consonanti = malloc(strlen(s)*sizeof(char)+1);
  char *vocali     = malloc(strlen(s)*sizeof(char)+1);
  char *p_c = consonanti;
  char *p_v = vocali;

  char *src = s;
  while (*src!='\0') {
    if ( *src=='a' || *src=='e' || *src=='i' || *src=='o' || *src=='u' ) {
      *p_v = *src;
      p_v++;
    } else {
      *p_c = *src;
      p_c++;
    }
    src++;
  }
  *p_c = *p_v = '\0';
  p_v = vocali;
  while (*p_v!='\0') {
    *p_c = *p_v;
    p_c++;
    p_v++;
  }
```

```
  *p_c = '\0';

  free(vocali);
  return consonanti;
}
```

A - Sottolista Fortunata

```
int lucky ( list_t *l ) {
  while ( l!=NULL ) {
    list_t *l2 = l;
    int sum = 0;
    while (l2!=NULL) {
      sum += l2->data;
      if (sum==13)
        return 1;
      l2 = l2->next;
    }
    l = l->next;
  }
  return 0;
}
```

B - Scacchi

```
int chess(int *M, int n_rows, int n_cols) {
  int i, j;
  int sum = 0;
  for (i=0; i<n_rows/2; i++) {
    for (j=0; j<n_cols/2; j++) {
      sum += M[i*2*n_cols + j*2];
      sum += M[(i*2+1)*n_cols + j*2+1];
    }
  }
  return sum;
}
```

B - Catena

```
int catena(char *a[], int a_size, int n) {
  int i;
  for (i=0; i<a_size-1; i++) {
    int len = strlen(a[i]);
    int j;
    for (j=0; j<n; j++) {
      if (a[i][len-n+j] != a[i+1][j])
        return 0;
    }
  }
  return 1;
}
```

B - Ripetizioni

```c
int is_prefix(list_t *a, list_t *b) {
  while (a!=NULL && b!=NULL && a->data==b->data) {
    a = a->next;
    b = b->next;
  }
  return a==NULL;
}

int reps(list_t *a, list_t *b) {
  int c = 0;
  while (b!=NULL) {
    c += is_prefix(a,b);
    b = b->next;
  }
  return c;
}
```

B - Staccionata

```c
int fence_rec(int n, int prev_1, int prev_2) {
  int count = 0;
  int color;

  if (n==0) return 1; /* fence completed */

  for (color=1; color<=4; color++) {
    if (color!=prev_1 && color!=prev_2)
      count += fence_rec(n-1, color,prev_1);
  }
  return count;
}

int fence(int n) {
  return fence_rec(n, 0, 0);
}
```

Appello del 23/08/2021

Istruzioni

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti è obbligatoria solo per chi non ha ricevuto un voto finale sufficiente nelle esercitazioni. La seconda metà è obbligatoria per tutti. Nella seconda parte, lo studente potrà scegliere tra completare la Sezione A o completare la Sezione B. Completare correttamente tutta la Sezione A consentirà di avere un voto massimo di 25, mentre completare correttamente tutta la Sezione B consentirà di avere un voto massimo pari a 30L. Non è quindi obbligatorio completare entrambe le sezioni, ad es. chi completa la Sezione B non deve completare la Sezione A.

In ogni caso é obbligatorio attenersi allo standard ANSI C.

Domande a Risposta Multipla - Per chi non ha un voto sufficiente nelle esercitazioni

Q1 Scope

Dato il seguente codice:

```c
int x = 0;
int z = 0;
int f1(int w) {
  z = ++x + w;
  {
    int z = x;
    x = x * 2;
    z = z * 2;
  }
  return z;
}
```

Qual è l'output della seguente istruzione ?

```c
  printf( "%d %d\n", f1(0), f1(1) );
```

- ☐ 1 2
- ☐ 2 4
- ☐ 1 3
- ☐ 1 4

Q2 Puntatori

Data la seguente funzione:

```c
void f2(int a1, int *a2, int b1, int *b2) {
  a1++;
  a2++;
  b1++;
```

```
  (*b2)++;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
int a = 0, b = 0;
f2(a, &a, b, &b);
printf("%d %d\n", a, b);
```

- ☐ `2 2` perché la funzione `f2` incrementa sia `a` che `b` per due volte.
- ☐ `0 1` perché la funzione `f2` modifica solo la variabile `b`.
- ☐ `1 1` perché la funzione `f2` incrementa sia `a` che `b` una sola volta.
- ☐ `0 0` perché la funzione `f2` modifica copie di `a` e `b` ma non le locazioni di memoria originarie.

Q3 Array e Matrici

E' corretto dire che la seguente funzione aggiunge un valore dato `x` a ciascun elemento della matrice quadrata `m` in input?

```
void f3(int *m, int n_rows, int x) {
  int i,j;
  for (i=0; i<n_rows; i++) {
    for (j=0; j<n_rows; j++) {
      m[i][j] += x;
    }
  }
}
```

- ☐ No, il codice non compila.
- ☐ Si, il codice é corretto perchè `m` ha lo stesso numero di righe e colonne `n_rows`.
- ☐ No, dovremmo applicare le correzioni `i<=n_rows` e `j<=n_rows`.
- ☐ Si, anche se lo stesso risultato poteva essere ottenuto con un solo ciclo **`for`**.

Q4 Loops

Quale valore restituisce la chiamata alla seguente funzione?

```
int f4() {
  int i = 0, j=10;
  int sum = 0;
  while ( ++i <= j )
    while ( j-- >=0 )
      if ( i==j )
        sum += i;
  return sum;
}
```

- ☐ `55`, ossia la somma dei numeri da `1` a `10`.
- ☐ `45`, ossia la somma dei numeri da `0` a `9`.
- ☐ `1`, perchè il ciclo interno viene eseguito una sola volta.
- ☐ Non si può usare la variabile `j` nella condizione del ciclo esterno.

Domande a Risposta Multipla - Obbligatorie per tutti

Q5 Struct

```c
struct card {
  int value; // 1,2,3, ...
  int suit;  // hearts, diamonds, ...
};

struct player {
  struct card cards[5];
};

struct game {
  struct player *players;
  int num_players;
};
```

Considerando i tipi definiti sopra utili per modellare una partita di poker, selezionare le funzioni che dato un array di giocatori restituiscono 1 *se e solo se* uno dei giocatori ha tutte le carte dello stesso seme (*suit*).

```c
int fun_A(struct game *G) {
  int i,j;
  for (i=0; i<G->num_players; i++) {
    int color = 1;
    for (j=1; j<5; j++) {
      color = color && G->players[i]->cards[j]->suit==G->players[i]->cards[0]->suit;
    }
    if (color)
      return 1;
  }
  return 0;
}
```

```c
int fun_B(struct game *G) {
  int i,j;
  for (i=0; i<G->num_players; i++) {
    struct card *cards = G->players[i].cards;
    int color = 1;
    for (j=1; j<5; j++) {
      color = color && cards[j].suit==cards->suit;
    }
    if (color)
      return 1;
  }
  return 0;
}
```

```c
int fun_C(struct game *G) {
  int i,j;
  for (i=0; i<G->num_players; i++) {
    struct card *cards = G->players[i].cards;
    int color = 1;
    for (j=1; j<5; j++) {
      color = color && cards[j]->suit==cards->suit;
    }
    if (color)
      return 1;
  }
  return 0;
}
```

```c
int fun_D(struct game *G) {
```

```
  int i,j;
  for (i=0; i<G->num_players; i++) {
    int color = 1;
    for (j=1; j<5; j++) {
      color = color && G->players[i].cards[j].suit==G->players[i].cards[0].suit;
    }
    if (color)
      return 1;
  }
  return 0;
}
```

- ☐ fun_A
- ☐ fun_B
- ☐ fun_C
- ☐ fun_D

Q6 Iterazione

Quali delle seguenti funzioni restituisce il valore `1` se l'array di interi in input contiene valori in ordine crescente e consecutivi tra loro e restituisce `0` altrimenti? *Se l'array in input ha lunghezza inferiore a `2` la funzione deve restituire il valore `1`.*

```
int fun_E(int *v, int size) {
  int i=0;
  int consecutive = 1;
  for (i=0; i<size-1; i++) {
    if (v[i]+1==v[i+1])
      consecutive = 1;
    else
      consecutive = 0;
  }
  return consecutive;
}
```

```
int fun_F(int *v, int size) {
  int i=0;
  int consecutive = 1;
  for (i=0; i<size-1 && consecutive; i++) {
    consecutive = consecutive && (v[i]+1==v[i+1]);
  }
  return consecutive;
}
```

```
int fun_G(int *v, int size) {
  int i=0;
  int consecutive = 1;
  for (i=0; i<size-1; i++) {
    consecutive = consecutive && (v[i]+1==v[i+1]);
    if (!consecutive)
      return consecutive;
  }
  return consecutive;
}
```

```
int fun_H(int *v, int size) {
  int i=0;
  for (i=0; i<size-1; i++)
    if (v[i]>v[i+1])
      return 0;
```

```
  return 1;
}
```

- ☐ `fun_E`
- ☐ `fun_F`
- ☐ `fun_G`
- ☐ `fun_H`

Q7 Stringhe

Claudio ha scritto una funzione che data una stringa `s` che contiene due asterischi `*` genera una nuova stringa come segue: la sottostringa delimitata dagli asterischi viene replicata e gli asterischi eliminati. Ad esempio, chiamata con argomento la stringa `*super*fun`, viene restituita la nuova stringa `supersuperfun`. *(Assumiamo `s` contenga sempre esattamente due asterischi).*

Quali errori ci sono nella funzione qui sotto?

```
char* asterisk(char* s) {
  int new_l = (strlen(s)-2)*2 + 1;
  char *new_s = (char*) malloc (new_l);
  char* first_star = 0;
  char *src;
  char *dst = new_s;
  for ( src=s; src!='\0'; src++ ) {
    if (*src=='*') {
      if ( first_star == 0 ) {
        first_star = src;
      } else {
        char *i = first_star+1;
        while (i!=src)
          *(dst++) = *(i++);
      }
    } else {
      *(dst++) = *src;
    }
  }
  *dst = *src;
  return new_s;
}
```

- ☐ L'istruzione `*(dst++)= *(i++);` andrebbe sostituita con `(*dst)++ = (*i)++;`.
- ☐ Non viene allocata sufficiente memoria per memorizzare la nuova stringa.
- ☐ La nuova stringa non contiene il carattere di terminazione `\0`.
- ☐ L'espressione `src!='\0'` andrebbe sostituita con `*src!='\0'`.

Q8 Ricorsione

Dire quali le sono implementazioni corrette della seguente funzione ricorsiva.

$$f(n) = \begin{cases} 1 & \text{se } n = 0 \\ 2 & \text{se } n = 1 \\ 2 * f(n-1)/f(n-2) & \text{altrimenti} \end{cases}$$

```
float fun_1(int n) {
  if ( n==0 ) return 1;
  if ( n==1 ) return 2;
  return 2 * fun_1(n-1) / fun_1(n-2);
}

```

```c
float fun_2(int n) {
  float f_, f__, res;
  int i;
  if ( n==0 || n==1 ) return n+1;

  f_ = 2; f__ = 1;
  for (i=2; i<=n; i++) {
    res = 2 * f_ / f__;
    f_ = res;
    f__ = f_;
  }
  return res;
}

float fun_3(int n) {
  float f_, f__, res;
  int i;
  if ( n>=0 ) f__ = res = 1;
  if ( n>=1 ) f_  = res = 2;

  for (i=2; i<=n; i++) {
    res = 2 * f_ / f__;
    f__ = f_;
    f_ = res;
  }

  return res;
}

float fun_4(int n) {
  int i;
  float *v = (float*) malloc (n*sizeof(float));
  if (!v) exit(EXIT_FAILURE);

  v[0] = 1;
  v[1] = 2;
  for (i=2; i<=n; i++)
    v[i] = 2 * v[i-1] / v[i-2];

  free(v);

  return v[n];
}
```

- ☐ fun_1
- ☐ fun_2
- ☐ fun_3
- ☐ fun_4

Soluzioni Domande

1. Scope

- ☒ 1 4

2. Puntatori

☒ `0 1` perché la funzione `f2` modifica solo la variabile `b`.

3. Array e Matrici -

☒ No, il codice non compila.

4. Loops

☒ `1`, perchè il ciclo interno viene eseguito una sola volta.

5. Struct

☒ `fun_B`
☒ `fun_D`

6. Iterazione

☒ `fun_F`
☒ `fun_G`

7. Stringhe

☒ L'espressione `src!='\0'` andrebbe sostituita con `*src!='\0'`.

8. Ricorsione

☒ `fun_1`

☒ `fun_3`

Commento:. La risposta `fun_4` è sbagliata per moltissimi motivi. Se `n` è così grande da non avere sufficiente memoria la funzione non resituirebbe il valore voluto. Bisogna allocare un vettore con `n+1` elementi. In seguito alla `free` l'area di memoria corrispondente viene liberata.

Esercizi Appello

Parte A (voto 18-25)

A - Number of Matches

Scrivere una funzione `n_matches` che, dati due array di interi `A` e `B` aventi la stessa lunghezza diversa da zero, restituisca quante volte i due vettori hanno lo stesso valore nella stessa posizione.

Esempio:

```c
int A[] = { 1,2,3,4,5 };
int B[] = { 1,5,3,4,2 };
int C[] = { 1,0,3,4,2 };

printf("%d \n", n_matches( A, B, 5) );
printf("%d \n", n_matches( B, C, 5) );
```

Output:

```
3   /* posizioni 0,2,3   */
4   /* posizioni 0,2,3,4 */
```

A - Mirror

Chiamiamo una matrice quadrata `A` *mirror* se è sempre vero che la sua i-esima riga è uguale alla i-esima colonna. Scrivere una funzione che data una matrice restituisca `1` se la matrice è *mirror* e `0` altrimenti.

Esempio:

```c
int i;
int M[4][4] = { { 1, 2, 3, 4 },
                { 2, 0, 0, 2 },
                { 3, 0, 0, 2 },
                { 4, 2, 2, 7 }, };
    /* prima riga uguale alla prima colonna,
       seconda riga uguale alla seconda colonna, etc. */
printf("%d ", mirror( &(M[0][0]), 4) );
```

Output:

```
1
```

A - Shift Cypher

Scrivere una funzione che data una stringa ne restituisce la versione decodificata secondo l'algoritmo seguente.

1. leggo la stringa `s` da sinistra a destra
2. se trovo il simbolo `<`, questo viene rimosso e il carattere successivo viene posto all'inizio della stringa decodificata
3. ogni altro simbolo resta invariato nella stringa decodificata

Esempio:

```c
char *s = "gram<o<r<pming";
printf("%s ", shift(s) );
```

Output:

```
programming
  /*   I° shift "ogram<r<pming"
      II° shift "rogram<pming"
     III° shift "programming"
  NB: i passi sono esemplificativi e non necessariamente rispecchiano il codice da
      implementare.
  */
```

A - Sottolista unitaria

Scrivere una funzione che data una lista di float restituisca il valore `1` se contiene una sotto-lista di elementi consecutivi i cui elementi hanno prodotto pari a `1`, e restituisca `0` altrimenti.

```c
typedef struct flist {
  float data;
  struct flist *next; /* next == NULL at the end of the list */
} flist_t;
```

Esempio:

```c
/* given that: l1 contains { 9, 9, 1, 9, 0} */
printf("%d \n", uni(l1) );
/* given that: l2 contains { 7, 5, 4, 0.05, 14} */
printf("%d \n", uni(l2) );
```

Output:

```
1   /* la sotto-lista è: 1 */
1   /* la sotto-lista è: 5, 4, 0.05 */
```

Parte B (voto 18-30L)

B - Peel

Scrivere una funzione che data una matrice rettangolare, restituisca la matrice risultante dalla rimozione della prima e ultima riga e della prima e ultima colonna come nell'esempio sotto. *(Assumiamo la matrice abbia almeno 3 righe e 3 colonne.)*

Input:

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
```

Output:

```
  6  7
 10 11
```

B - Longest Prefix

Dato un array di stringhe, scrivere una funzione che resituisce la lunghezza del prefisso comune più lungo.

Esempio:

```c
  char* a[] = {"veloce", "velocipede", "veliero"};
  char* b[] = {"mele", "pere"};
  printf("%d\n", prefix(a, 3) );
  printf("%d\n", prefix(b, 2) );
```

Output:

```
3   /* vel-oce, vel-ocipede, vel-iero */
0   /* non ci sono prefissi comuni */
```

B - Rotazione

Data una lista `l1` e un intero `n`, definiamo rotazione `n` di `l1` la lista che si ottiene spostando all'inizio della lista gli ultimi `n` elementi di `l1`.

Date due liste `l1` e `l2` scrivere una funzione che restituisce il valore `n` per cui `l2` è una rotazione `n` di `l1`, e `-1` se `l2` non può essere una rotazione di `l1`.

La struttura dati lista è definita come segue:

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```c
/* given that: l1 contains {4,5,6,7,8,9,10} */
/*         and l2 contains {8,9,10,4,5,6,7} */
printf("%d \n", rotated(l1, l2) );
```

Output:

```
3  /* ruotando gli ultimi 3 elementi di l1 */
```

B - Percorso

Sia dato un percorso lineare accidentato. Possiamo attraversare questo percorso solo ricoprendolo con delle tavole rettangolari di diversa lunghezza senza che si sovrappongano a vicenda e tali che coprano esattamente la lunghezza del percorso senza eccedere.

Le tavole a disposizione sono codificate da due array `lens` and `counts` che contengono per ciascuna tipologia di tavola la sua lunghezza e il numero di tavole disponibili. Scrivere una funzione che date le informazioni sopra in input e la lunghezza del percorso, restituisca `1` se è possibile coprirlo e `0` altrimenti.

Esempio:

```c
int path1 = 11;
int path2 = 12;
int lens[]   = { 2, 5};
int counts[] = { 5, 1};
int n = 2;
/* abbiamo 2 tipi di tavole diverse:
       5 di lunghezza 2 e 1 di lunghezza 5 */
printf("%d \n", path(lens, counts, n, path1));
printf("%d \n", path(lens, counts, n, path2));
```

Output:

```
1  /* usando 3 lunghe 2 e 1 lunga 5 */
0  /* non esiste nessuna configurazione */
```

Soluzioni Esercizi

A - Number of Matches

```c
int n_matches(int* A, int* B, int size) {
  int i, count=0;
  for (i=0; i<size; i++)
    if (A[i] == B[i])
      count++;
  return count;
}
```

A - Mirror

```c
int mirror(int* M, int n) {
  int i,j;
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++)
      if ( M[i*n + j] != M[j*n + i] )
        return 0;
  }
  return 1;
}
```

A - Shift Cypher

```c
char * shift(char *s) {
  char *decoded;
  char *src;
  int num_shifts = 0;
  int i, l, r;

  // count '<'
  src = s;
  while (*src!='\0') {
    if ( *src=='<' )
      num_shifts++;
    src++;
  }
  decoded = (char*) malloc( (strlen(s)-num_shifts)*sizeof(char)+1);
  if (!decoded) exit(EXIT_FAILURE);

  // scan again from start
  src = s;
  l = num_shifts-1;
  r = num_shifts;
  while (*src!='\0') {
    if ( *src=='<' ) {
      src++;
      decoded[l--] = *src;
    } else {
      decoded[r++] = *src;
    }
    src++;
  }

  decoded[r] = '\0';

  return decoded;
}
```

A - Sottolista Unitaria

```c
int uni ( flist_t *l ) {
  while ( l!=NULL ) {
    flist_t *l2 = l;
    float prod = 1.0f;
    while (l2!=NULL) {
      prod *= l2->data;
      if ( prod == 1 )
```

```
        return 1;
      l2 = l2->next;
    }
    l = l->next;
  }
  return 0;
}
```

B - Peel

```
int * peel(int *M, int n_rows, int n_cols) {
  int *T = (int*) malloc((n_rows-2)*(n_cols-2)*sizeof(int));
  int i,j;
  for (i=1; i<n_rows-1; i++) {
    for (j=1; j<n_cols-1; j++)
      T[(i-1)*(n_cols-2) + j-1] = M[i*n_cols + j];
  }
  return T;
}
```

B - Longest Prefix

```
int prefix(char *a[], int a_size) {
  int i;
  int longest;
  if (a_size==0) return 0;
  longest = strlen(a[0]);
  for (i=1; i<a_size; i++) {
    int other_len = strlen(a[i]);
    int new_longest = 0;
    while ( new_longest < longest &&
          new_longest < other_len &&
          a[0][new_longest] == a[i][new_longest] ) {
      new_longest++;
    }
    longest = new_longest;
  }
  return longest;
}
```

B - Rotazione

```
int is_rotated(list_t *a, list_t *b, int n) {
  int c = 0;
  list_t *b_copy = b;
  // skip n elements of b (b_copy)
  while (b_copy!=NULL && c<n) {
    b_copy = b_copy->next;
    c++;
  }
  // check actually skipped
  if (c!=n)
    return 0;
  // compare remaining of b
  while (b_copy!=NULL && a!=NULL && b_copy->data==a->data) {
```

```
    b_copy = b_copy->next;
    a = a->next;
  }
  // if b (b_copy) was not consumed to the end
  if (b_copy!=NULL)
    return 0;
  // compare remaining of a with beginning of b
  while (b!=NULL && a!=NULL && b->data==a->data) {
    b = b->next;
    a = a->next;
  }
  // check that a was consumed to the end
  return a==NULL;
}

int rotated(list_t *l1, list_t *l2) {
  int n = 0;
  list_t *l = l1;
  while (l!=NULL) {
    if ( is_rotated(l1, l2, n) )
      return n;
    l = l->next;
    n++;
  }
  return -1;
}
```

B - Percorso

```
int path(int *lens, int *counts, int n, int remaining) {
  int i;
  int c = 0; /* non trovato*/

  if (remaining==0) /* trovato */
    return 1;

  if (remaining<0 || n==0) /* non trovato */
    return 0;

  for (i=0; i<=counts[0] && !c; i++) {
    c = c || path(lens+1, counts+1, n-1, remaining-i*lens[0]);
  }
  return c;
}
```

Prova Intermedia del 03/11/2021

Istruzioni

La prima parte dell'esame prevede 4 domande a risposta multipla obbligatorie per tutti. La seconda parte include 4 esercizi, ma è richiesto di completarne solo 2.

In ogni caso è obbligatorio attenersi allo standard ANSI C.

Domande a risposta multipla

1. Scope 1

Considerare il seguente codice:

```c
int z = 10;
int f1(int z) {
  return z++;
}
int f2(int x) {
  z = z + x;
  return ++z;
}
```

Cosa visualizza a schermo l'istruzione `printf( "%d %d\n", f2(10), f1(10));`?

- ☐ `21 11`
- ☐ `21 20`
- ☐ `21 10`
- ☐ `21 21`

1. Scope 2

Considerare il seguente codice:

```c
int w = 10;
int g1(int w) {
  return w++;
}
int g2(int x) {
  w = w + x;
  return ++w;
}
```

Cosa visualizza a schermo l'istruzione `printf( "%d %d\n", g1(10), g2(10));`?

- ☐ `11 21`
- ☐ `10 21`
- ☐ `10 22`
- ☐ `11 22`

2. Puntatori 1

Considerare la seguente funzione:

```c
void rotate(int *a, int *b, int*c) {
  *a = *b;
  *b = *c;
  *c = *a;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```c
{
  int x = 1, y = 2, z = 3;
  rotate ( &x, &y, &z );
  printf( "%d %d %d\n", x, y, z );
}
```

- ☐ il codice non è corretto perché avremmo dovuto scrivere: `printf( "%d %d %d\n", *x, *y, *z );`
- ☐ il codice non è corretto perché avremmo dovuto scrivere: `*a = b; *b = c; *c = a;`
- ☐ `2 3 2`
- ☐ `2 3 1`

2. Puntatori 2

Considerare la seguente funzione:

```c
void swap(int *a, int *b, int*c) {
  *a = b;
  *b = c;
  *c = a;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```c
{
  int x = 1, y = 2, z = 3;
  swap ( &x, &y, &z );
  printf( "%d %d %d\n", x, y, z );
}
```

- ☐ il codice non è corretto perché avremmo dovuto scrivere: `printf( "%d %d %d\n", *x, *y, *z );`
- ☐ `2 3 1`
- ☐ `2 3 2`
- ☐ Impossibile stabilire l'output

2. Puntatori 3

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```c
{
  int v[] = {1,2,3,4,5};
  int *w = v;
  int *z = v + 2;
  v[0] = *z;
```

```
  z = w + 3;
  *z = *(w + 1);
  printf( "%d %d %d %d %d\n", v[0], v[1], v[2], v[3], v[4] );
}
```

- ☐ il codice non compila perché l'istruzione `z = w + 3;` non ha senso
- ☐ il codice non compila perché l'assegnamento `int *w = v;` di un vettore ad un puntatore non è consentito
- ☐ il codice non compila perché avremmo dovuto scrivere `v[0] = z;`
- ☐ Nessuna delle altre risposte

2. Puntatori 4

Considerare la seguente funzione:

```
void vv(int *a, int *b) {
  int *aux = a;
  a = b;
  b = aux;
  b[0] = 9;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
{
  int v[] = {1,2,3,4,5};
  int w[] = {6,7,8,9,10};
  vv ( v, w );
  printf( "%d %d %d %d %d\n", v[0], v[1], v[2], v[3], v[4] );
}
```

- ☐ `1 2 3 4 5`
- ☐ `6 7 8 9 10`
- ☐ `9 2 3 4 5`
- ☐ `9 7 8 9 10`

3. Matrici 1

E' corretto dire che la seguente funzione calcola la somma degli elementi nella seconda colonna di una matrice quadrata? Indicare le affermazioni corrette.

```
int mat(int *m, int lato) {
  int res = 0;
  int i, j;
  m += 2;
  for (j=0; j<lato; j++) {
    res += *m;
    m += lato;
  }
  return res;
}
```

- ☐ No, il codice non è corretto perché `j` non viene mai usata all'interno del ciclo `for`
- ☐ No, il codice calcola la somma degli elementi nella terza colonna
- ☐ No, il codice calcola la somma degli elementi nella terza riga
- ☐ Si, il codice "salta" da una riga all'altra

4. Loop 1

Considerare il codice seguente:

```
{
  int i, j;
  for (i=0; i<10; i += 2)
    for (j=0; j<10; j+=3) {
      if (i==0) break;
      if (j==6) printf("OK\n");
      if (j==9) break;
    }
}
```

Quali delle seguenti affermazioni sono corrette?

- ☐ il doppio ciclo termina raggiunta l'istruzione `if (i==0)break;`
- ☐ il codice stampa la stringa `OK` una sola volta;
- ☐ il codice stampa la stringa `OK` per quattro volte
- ☐ il codice stampa la stringa `OK` per cinque volte

4. Loop 2

Considerare la funzione e il codice seguente:

```
int check(int *v, size_t v_size) {
  /* assume v_size>0 */
  int i;
  int res = (*v % 3) == 0;
  for (i=1; res && i<v_size; i++) {
    if (res == 0) return 0;
    res = res && ( (v[i] % 3) == 0 );
  }

  return res;
}
```

```
{
  int v[] = {3,6,9,12};
  int vv[] = {2,6,9,12};
  printf("%d\n", check(v, 4));
  printf("%d\n", check(vv, 4));
}
```

Quali delle seguenti affermazioni sono corrette?

- ☐ la funzione restituisce 1 se `v` contiene solo multipli di 3 e restituisce 0 altrimenti
- ☐ l'istruzione `if (res == 0)return 0;` permette di terminare prima la funzione
- ☐ il codice non è corretto perché il primo elemento di `v` viene ignorato
- ☐ il codice non è corretto perché il primo elemento di `v` viene considerato due volte

Soluzioni Domande

1. Scope 1

- ☒ `21 10`

1. Scope 2

☒ `10 21`

2. Puntatori 1

☒ `2 3 2`

2. Puntatori 2

☒ Impossibile stabilire l'output

2. Puntatori 3

☒ Nessuna delle altre risposte

2. Puntatori 4

☒ `9 2 3 4 5`

3. Matrici 1

☒ No, il codice calcola la somma degli elementi nella terza colonna

4. Loop 1

☒ il codice stampa la stringa `OK` per quattro volte

4. Loop 2

☒ la funzione restituisce 1 se `v` contiene solo multipli di 3 e restituisce 0 altrimenti

---

Esercizi

Palindromo

Scrivere una funzione `palindromo` che, dati due array di interi `A` e `B` aventi la stessa lunghezza diversa da zero, restituisca il valore `1` se il primo array contiene gli stessi elementi del secondo in ordine inverso, e restituisca `0` altrimenti

Esempio:

```c
int A[] = { 10, 20, 30, 40, 50 };
int B[] = { 50, 40, 30, 20, 10 };
int C[] = { 50, 40, 10, 20, 30 };

printf("%d \n", palindromo( A, B, 5) );
printf("%d \n", palindromo( A, C, 5) );
```

Output:

```
1
0
```

Quasi uguali

Scrivere una funzione `quasi_uguali` che, dati due array di interi `A` e `B` aventi la stessa lunghezza diversa da zero, restituisca il valore `1` se il primo array contiene gli stessi elementi del secondo incrementati di `1`, e restituisca `0` altrimenti

Esempio:

```c
int A[] = { 2,3,4,5,6 };
int B[] = { 1,2,3,4,5 };
int C[] = { 2,3,4,5,6 };

printf("%d \n", quasi_uguali( A, B, 5) );
printf("%d \n", quasi_uguali( A, C, 5) );
```

Output:

```
1
0
```

Incluso

Scrivere una funzione `incluso` che, dati due array di interi `A` e `B` di lunghezza diversa da zero, restituisca il valore `1` se gli elementi del primo array sono tutti contenuti nel secondo in qualsiasi ordine, e restituisca `0` altrimenti.

Esempio:

```c
int A[] = { 1,2,3 };
int B[] = { 1,4,2,5,3 };
int C[] = { 2,3,4,5,6 };

printf("%d \n", incluso( A, 3, B, 5) );
printf("%d \n", incluso( B, 5, C, 5) );
```

Output:

```
1
0
```

Escluso

Scrivere una funzione `escluso` che, dati due array di interi `A` e `B` di lunghezza diversa da zero, restituisca il valore `1` se nessuno degli elementi del primo array è contenuti nel secondo, e restituisca `0` altrimenti.

Esempio:

```c
int A[] = { 1,2,3 };
int B[] = { 10,11,12,13,14 };
int C[] = { 6,7,8,9,10 };

printf("%d \n", escluso( A, 3, B, 5) );
printf("%d \n", escluso( B, 5, C, 5) );
```

Output:

```
1
0
```

Sotto-array A

Dato una array di interi (sia positivi che negativi) definiamo il suo peso come la somma dei suoi elementi. Scrivere una funzione `pesante` che date un array non vuoto di interi restituisca il peso del suo sotto-array di elementi consecutivi più pesante.

Esempio:

```c
int A[] = { -1, 1, -10, 2, 3, 4, 5 };
int B[] = { 5, -3, 5, 2, 1, 4, 5, -7, -8 };

printf("%d \n", pesante( A, 7) );
printf("%d \n", pesante( B, 9) );
```

Output:

```
14 /* è il peso del sotto-array: 2, 3, 4, 5 */
19 /* è il peso del sotto-array: 5, -3, 5, 2, 1, 4, 5 */
```

Sotto-array B

Scrivere una funzione `zero` che dato un array non vuoto di interi (sia positivi che negativi) restituisca la lunghezza del suo più lungo sotto-array di elementi consecutivi la cui somma è pari a zero. La funzione restituisce 0 se tale sotto-array non esiste.

Esempio:

```c
int A[] = { -1, 1, -10, 2, 3, 4, 5 };
int B[] = { 5, -3, 5, 2, 1, 4, 5, -7, -8 };

printf("%d \n", zero( A, 7) );
printf("%d \n", zero( B, 9) );
```

Output:

```
5 /* corrisponde al sotto-array: 1, -10, 2, 3, 4 */
0 /* non ci sono sotto-array a somma 0 */
```

Nuovo array A

Scrivere una funziona `dispari` che dato un array di interi restituisce un nuovo array (e la sua lunghezza) contenente solo gli elementi dispari dell'array dato.

Esempio:

```c
int A[] = { 1,2,3,4,5,6,7,8,9 };
int new_size, i;
int *new_array = dispari( A, 9, &new_size);
for (i=0; i<new_size; i++)
  printf("%d ", new_array[i]);
printf("\n");
```

Output:

```
1 3 5 7 9
```

Nuovo array B

Scrivere una funziona `cancella` che dato un array di interi restituisce un nuovo array (e la sua lunghezza) contenente solo gli elementi dell'array dato preceduti da uno zero. *(nota `00x` diventa `0x`)*

Esempio:

```c
int A[] = { 0,1,0,2,0,3,9,9,0,0,0,1,0 };
int new_size, i;
int *new_array = cancella( A, 13, &new_size);
for (i=0; i<new_size; i++)
  printf("%d ", new_array[i]);
printf("\n");
```

Output:

```
1 2 3 0 0 1
```

Soluzioni Esercizi

Palindromo

```c
int palindromo(int* A, int* B, int size) {
  int i;
  for (i=0; i<size; i++)
    if (A[i] != B[size-i-1])
      return 0;
  return 1;
}
```

Quasi Uguali

```c
int quasi_uguali(int* A, int* B, int size) {
  int i;
  for (i=0; i<size; i++)
    if (A[i] != B[i] + 1)
      return 0;
  return 1;
}
```

Incluso

```c
int incluso(int* A, int A_size, int* B, int B_size) {
  int i,j;
  for (i=0; i<A_size; i++)  {
    int found = 0;
    for (j=0; !found && j<B_size; j++)
    if (A[i] == B[j])
      found = 1;
    if (!found)
      return 0;
  }
  return 1;
}
```

Escluso

```c
int escluso(int* A, int A_size, int* B, int B_size) {
  int i,j;
  for (i=0; i<A_size; i++)  {
    int found = 0;
    for (j=0; !found && j<B_size; j++)
    if (A[i] == B[j])
      found = 1;
    if (found)
      return 0;
  }
  return 1;
}
```

Sotto Array A

```c
int pesante(int* A, int A_size) {
  int i,j;
  int max_peso = A[0];
  for (i=0; i<A_size; i++)  {
    int peso = 0;
    for (j=i; j<A_size; j++) {
      peso += A[j];
      if (peso>max_peso)
        max_peso = peso;
    }
  }
  return max_peso;
}
```

Sotto Array B

```c
int zero(int* A, int A_size) {
  int i,j;
  int max_len = 0;
  for (i=0; i<A_size; i++)  {
    int peso = 0;
    for (j=i; j<A_size; j++) {
```

```
      peso += A[j];
      if (peso==0 && j-i+1>max_len) {
        max_len = j-i+1;
      }
    }
  }
  return max_len;
}
```

Nuovo Array A

```
int* dispari(int* A, int A_size, int *new_size) {
  int i, count=0;
  int *new_array;
  *new_size = 0;
  for (i=0; i<A_size; i++)
    if ( A[i]%2 )
      count++;
  if (count==0) return 0;

  new_array = (int*) malloc(count*sizeof(int));
  if (!new_array) exit(EXIT_FAILURE);

  for (i=0; i<A_size; i++)
    if ( A[i]%2 )
      new_array[(*new_size)++] = A[i];
  return new_array;
}
```

Nuovo Array B

```
int* cancella(int* A, int A_size, int *new_size) {
  int i, count=0;
  int *new_array;
  *new_size = 0;
  for (i=0; i<A_size-1; i++)
    if ( A[i]==0 )
      count++;
  if (count==0) return 0;

  new_array = (int*) malloc(count*sizeof(int));
  if (!new_array) exit(EXIT_FAILURE);

  for (i=0; i<A_size-1; i++)
    if ( A[i]==0 )
      new_array[(*new_size)++] = A[i+1];
  return new_array;
}
```

Appello del 14/01/2022

Istruzioni

Scrivere subito nome, cognome e matricola sul testo e su tutti i fogli protocollo. Sia il testo che i fogli dovranno essere riconsegnati.

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti è obbligatoria solo per chi non ha superato le esercitazioni. La seconda metà è obbligatoria per tutti. Le domande a risposta multipla possono avere più risposte corrette o nessuna. Usate il testo per rispondere.

Nella seconda parte lo studente dovrà risolvere solo 3 esercizi su 5 in 2 ore. Scrivete le soluzioni nei fogli protocollo.

Il punteggio finale è dato dalla somma dei punteggi.

Appello A

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 pt.)

Dato il seguente codice:

```
int x=0;
int f1(int w) {
  if (x++ == 0)
    return x+1;
  else
    return x-1;
  x = x + w;
}
```

Qual è l'output della seguente istruzione?

```
printf( "%d %d\n", f1(10), f1(10) );
```

- ☐ 0 11
- ☐ 2 11
- ☐ 2 1
- ☐ 0 1

Q2 (1.5 pt.)

Considerare la seguente funzione:

```c
void add(int *a, int *b) {
  a += b;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```c
int x = 1, y = 2;
add ( &x, &y );
printf( "%d %d \n", x, y );
```

- ☐ 3 1
- ☐ 1 2
- ☐ Il codice non compila
- ☐ Nessuna delle altre risposte

Q3 (1.5 pt.)

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```c
int v[] = {1,2,3,4,5};
int *w = v;
w += 2;
v[0] = *w;
*w = *(w + 2);
printf( "%d %d %d %d %d\n", v[0], v[1], v[2], v[3], v[4] );
```

- ☐ 5 2 3 4 5
- ☐ 3 2 5 4 5
- ☐ il codice non compila perché `w += 2;` non è corretto
- ☐ il codice non compila perché `v[0] = *w;` non è corretto

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 pt.)

Considerando i tipi definiti sotto, qual è il modo corretto per accedere al nome di una squadra nella prima partita?

```c
struct team {
  char *name;
  int foundation_year;
};

struct match {
  struct team *teams;
  int goals_by_home_team;
  int goals_by_visitor_team;
};

struct tournament {
  struct match *matches;
  int num_matches;
};

struct tournament T;
```

```
/* altre inizializzazioni */
```

- ☐ `printf("%d", T.matches[0].teams[0].name)`
- ☐ `printf("%d", T->matches[0]->teams[0]->name)`
- ☐ `printf("%d", T.matches->teams->name)`
- ☐ `printf("%d", T.matches.teams.name)`

Q5 (1.5 pt.)

Claudio ha scritto una funzione per la concatenazione di due stringhe inserendo due asterischi tra le due. Ad esempio, chiamata con gli argomenti "`I Love`" e "`Coding`", la funzione dovrebbe restituire "`I Love**Coding`".

Quali errori ci sono nella funzione qui sotto?

```
char* concat(const char *a, const char *b) {
  size_t i=0, j=0;
  char * c = (char*) malloc ( strlen(a) + strlen(b) + 3 );
  if (!c) return NULL;
  for (j=0; j<strlen(a); j++)
    c[i++] = a[j];
  c[i++] = '*';
  c[i++] = '*';
  for (j=0; j<strlen(b); j++)
    c[i++] = b[j];
  c[i] = '\0';
  return c;
}
```

- ☐ Nessuno
- ☐ E' stata allocata più memoria del necessario
- ☐ `c[i++]` andrebbe sostituito con `c[++i]`
- ☐ Il carattere `'\0'` sovrascrive l'ultimo carattere della stringa

Q6 (1.5 pt.)

Selezionare le affermazioni corrette relative al codice seguente:

```
int f(int *a, int a_size) {
    if (a_size==1)
        return a[1];

    return a[1] + f(a+1, a_size-1);
}
```

- ☐ la funzione calcola la somma degli elementi di un array
- ☐ se l'array in input non è vuoto la funzione è corretta
- ☐ nel caso base si accede oltre la fine dell'array
- ☐ basta aggiungere il caso base `a_size==0` per correggere la funzione

Soluzioni Domande

Q1

- ☒ 2 1

Q2

☒ Il codice non compila

Q3

☒ 3 2 5 4 5

Q4

☒ `printf("%d", T.matches[0].teams[0].name)`
☒ `printf("%d", T.matches->teams->name)`

Q5

☒ Nessuno

Q6

☒ nel caso base si accede oltre la fine dell'array

Esercizi

Somma Rapporti (4 pt.)

Scrivere una funzione `somma_rapporti` dati due array di float `A` e `B` aventi la stessa lunghezza diversa da zero restituisca il risultato della seguente operazione:

$$\text{somma\_rapporti}(A, B) = \sum_i \frac{A[i]}{B[i]}.$$

Esempio:

```
float A[] = { 4.0f, 0.0f, 25.0f };
float B[] = { 2.0f, 3.0f, 5.0f };

printf( "%.2f \n", somma_rapporti(A, B, 3));
```

Output:

```
7.00 /* 4/2 + 0/3 + 25/5 */
```

La firma della funzione è : `float somma_rapporti( float *A, float *B, int size )`.

Matrice Stocastica (5 pt.)

Una matrice quadrata è detta stocastica se tutti i suoi elementi sono non negativi e se la somma degli elementi su ciascuna riga è pari a uno. Scrivere una funzione `stoc` che ritorna `1` se la matrice in input è stocastica e `0` altrimenti.

Esempio:

```c
float A[3][3] = { {0.1, 0.5, 0.4},   /* 0.1 + 0.5 + 0.4 = 1.0 */
                  {0.0, 0.5, 0.5},   /* 0.0 + 0.5 + 0.5 = 1.0 */
                  {0.3, 0.4, 0.3} }; /* 0.3 + 0.4 + 0.3 = 1.0 */

printf( "%d \n", stoc(&A[0][0], 3));
```

Output:

```
1
```

La firma della funzione è : `int stoc( float *M, int num_cols )`.

String Drop (6 pt.)

Scrivere una funzione `drop` che date due stringhe restituisce una nuova stringa ottenuta rimuovendo dalla prima tutte le lettere presenti nella seconda.

Esempio:

```c
char *a = "!p###la#!y!in#g!";
char *b = "!#";
printf( "%s \n", drop(a, b) );
```

Output:

```
playing
```

La firma della funzione è : `char * drop ( char *a, char * b )`.

Lista Bilanciata (7 pt.)

Un lista si dice bilanciata se esiste un valore i tale per cui la somma dei primi i elementi della lista è pari alla somma degli elementi rimanenti. Scrivere una funzione che, data una lista definita come segue, restituisca 1 se la lista è bilanciata e 0 altrimenti.

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```c
/* given that: l contains {1,10,2,4,5} */
printf("%d \n", bilanciata(l) );
```

Output:

```
1 /* 1+10 == 2+4+5 */
```

La firma della funzione è : `int bilanciata(list_t *l)`.

Count Vector (7 pt.)

Scrivere una funzione ricorsiva che dati due array conteggi il numero di volte che il primo array appare nel secondo.

Esempio:

```
int v1[] = {1, 3, 1};
int v2[] = {1, 3, 1, 3, 1, 1, 3, 1, 7};
printf("%d \n", count_array(v1, 3, v2, 9) );
```

Output:

```
3 /* a partire dalle posizioni 0, 2 e 5 di v2*/
```

La firma della funzione è : `int count_array( int *a, int a_size, int *b, int b_size )`.

Soluzioni Esercizi

Somma Rapporti

```
float somma_rapporti( float *A, float *B, int size ) {
  /* assume size>0 */
  float r = 0.0f;
  int i;
  for ( i=0; i<size; i++)
    r += A[i] / B[i];
  return r;
}
```

Matrice Stocastica

```
int stoc( float *M, int num_cols ) {
  /* assume num_cols >0 */
  int r, c;
  for (r=0; r<num_cols; r++) {
    float sum = 0;
    for (c=0; c<num_cols; c++) {
      if ( M[r*num_cols + c] < 0.0 )
        return 0;
      sum += M[r*num_cols + c];
    }
    if (sum!=1.0)
      return 0;
  }
  return 1;
}
```

String Drop

```
char * drop ( char *a, char *b ) {
  char *s = (char*) malloc(strlen(a)+1);
  char *i = a;
  char *t = s;
```

```
  while (*i) {
    char *j = b;
    int found = 0;
    while (!found && *j) {
      if (*i==*j) found = 1;
      j++;
    }
    if (!found) {
      *t = *i;
      t++;
    }
    i++;
  }
  *t = '\0';
  s = realloc(s, t-s+1);
  return s;
}
```

Lista Bilanciata

```
int list_sum(list_t *l) {
  int sum = 0;
  while (l) {
    sum+= l->data;
    l = l->next;
  }
  return sum;
}

int bilanciata(list_t *l) {
  int left_sum = 0;
  while (l) {
    int right_sum = list_sum(l);
    if (left_sum==right_sum)
      return 1;
    left_sum += l->data;
    l = l->next;
  }
  return left_sum==0;
}
```

Count Vector

```
int is_prefix( list_t *a, list_t *b ) {
  while (a && b) {
    if (a->data!=b->data)
      return 0;
    a = a->next;
    b = b->next;
  }
  return a==NULL;
}

int count_list( list_t *a, list_t *b ) {
  if (!b)
    return 0;
  return is_prefix(a,b) + count_list(a, b->next);
```

```
15 }
```

Appello B

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 pt.)

Dato il seguente codice:

```
int z=0;
int g1(int w) {
  if (++z == 0)
    return z++;
  else
    return z--;
}
```

Qual è l'output della seguente istruzione?

```
printf( "%d %d\n", g1(10), g1(10) );
```

- ☐ 1 1
- ☐ 0 1
- ☐ 2 2
- ☐ 1 2

Q2 (1.5 pt.)

Considerare la seguente funzione:

```
void add(int *a, int *b) {
  *a += *b;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
int x = 1, y = 2;
add ( x, y );
printf( "%d %d \n", x, y );
```

- ☐ 3 1
- ☐ 1 2
- ☐ Non si possono sommare due puntatori
- ☐ Nessuna delle altre risposte

Q3 (1.5 pt.)

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
int v[] = {1,2,3,4,5};
int *z = v;
z += *z;
*z = 0;
```

```
v[2] = v[ v[1] ];
printf( "%d %d %d %d %d\n", v[0], v[1], v[2], v[3], v[4] );
```

- ☐ 1 0 1 4 5
- ☐ 0 2 2 4 5
- ☐ il codice non compila perché `v[ v[1] ];` non è corretto
- ☐ il codice non compila perché `z += *z;` non è corretto

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 pt.)

Considerando i tipi definiti sotto, qual è il modo corretto per accedere all'anno di fondazione di una squadra della prima partita?

```
struct team {
  char *name;
  int foundation_year;
};

struct match {
  struct team *teams;
  int goals_by_home_team;
  int goals_by_visitor_team;
};

struct tournament {
  struct match *matches;
  int num_matches;
};

struct tournament T;
/* altre inizializzazioni */
```

- ☐ `printf("%d", T->matches[0]->teams[0]->foundation_year)`
- ☐ `printf("%d", T.matches[0].teams[0].foundation_year)`
- ☐ `printf("%d", T.matches->teams->foundation_year)`
- ☐ `printf("%d", T.matches.teams.foundation_year)`

Q5 (1.5 pt.)

Claudio ha scritto una funzione per la concatenazione di due stringhe inserendo due asterischi tra le due. Ad esempio, chiamata con gli argomenti "`I Love`" e "`Coding`", la funzione dovrebbe restituire "`I Love**Coding`".

Quali errori ci sono nella funzione qui sotto?

```
char* glue(const char *a, const char *b) {
  size_t i=0, j=0;
  char * c = (char*) malloc ( strlen(a) + strlen(b) + 2 );
  if (!c) return NULL;
  for (j=0; j<strlen(a); j++)
    c[i] = a[j];
    i++;
  c[i] = '*';
  i++;
```

```
  c[i] = '*';
  i++;
  for (j=0; j<strlen(b); j++)
    c[i] = b[j];
    i++;
  c[i] = '\0';
  return c;
}
```

- ☐ Nessuno
- ☐ Non viene allocata abbastanza memoria
- ☐ La stringa finale ha al massimo 4 caratteri
- ☐ Il carattere `'\0'` sovrascrive l'ultimo carattere della stringa

Q6 (1.5 pt.)

Selezionare le affermazioni corrette relative al codice seguente:

```
int f(int *a, int a_size) {
    if (a_size==0)
        return 0;

    return a[a_size] + f(a, a_size-1);
}
```

- ☐ la funzione calcola la somma degli elementi di un array
- ☐ la funzione accede oltre la fine dell'array
- ☐ basta aggiungere il caso base `a_size==1` per correggere la funzione
- ☐ nella chiamata ricorsiva dovremmo usare `a+1` per muoverci all'elemento successivo dell'array

Soluzioni Domande

Q1

- ☒ 1 1

Q2

- ☒ Nessuna delle altre risposte

Q3

- ☒ 1 0 1 4 5

Q4

- ☒ `printf("%d", T.matches[0].teams[0].foundation_year)`
- ☒ `printf("%d", T.matches->teams->foundation_year)`

Q5

- ☒ Non viene allocata abbastanza memoria
- ☒ La stringa finale ha al massimo 4 caratteri

Q6

- ☒ la funzione accede oltre la fine dell'array

Esercizi

Prodotto Differenze (4 pt.)

Scrivere una funzione `prod_diff` dati due array di float `A` e `B` aventi la stessa lunghezza diversa da zero restituisca il risultato della seguente operazione:

$$\text{prod\_diff}(A, B) = \prod_i (A[i] - B[i]).$$

Esempio:

```c
float A[] = { 1.0f, 5.0f, 4.0f };
float B[] = { 2.0f, 3.0f, 0.0f };

printf( "%.2f \n", prod_diff(A, B, 3));
```

Output:

```
-8.00 /* (1-2) * (5-3) * (4-0) */
```

La firma della funzione è : `float prod_diff( float *A, float *B, int size )`.

Matrice Stocastica (5 pt.)

Una matrice quadrata è detta stocastica se tutti i suoi elementi sono non negativi e se la somma degli elementi su ciascuna colonna è pari a uno. Scrivere una funzione `stocastic` che ritorna `1` se la matrice in input è stocastica e `0` altrimenti.

Esempio:

```c
float A[3][3] = { {0.1, 0.5, 0.2},
                  {0.6, 0.5, 0.5},
                  {0.3, 0.0, 0.3} };
  /* somme sulle
         colonne = 1.0, 1.0, 1.0    */
printf( "%d \n", stocastic(&A[0][0], 3));
```

Output:

```
1
```

La firma della funzione è : `int stocastic( float *M, int num_cols )`.

String Replicate (6 pt.)

Scrivere una funzione `replicate` che date due stringhe restituisce una nuova stringa ottenuta raddoppiando nella prima tutte le lettere presenti nella seconda.

Esempio:

```c
char *a = "matonela";
char *b = "tl";
printf( "%s \n", replicate(a, b) );
```

Output:

```
mattonella
```

La firma della funzione è : `char * replicate ( char *a, char * b )`.

Lista Doppiamente crescente (7 pt.)

Un lista si dice doppiamente crescente se esiste un valore i tale per cui sia i primi i elementi, sia i rimanenti sono ordinati in maniera crescente. Scrivere una funzione che, data una lista definita come segue, restituisca 1 se la lista è doppiamente crescente e 0 altrimenti.

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```c
/* given that: l contains {10,20,30,1,2,3,4,5} */
printf("%d \n", doppia_crescente(l) );
```

Output:

```
1 /* {10,20,30} e {1,2,3,4,5} sono entrambi crescenti */
```

La firma della funzione è : `int doppia_crescente(list_t *l)`.

Count List (7 pt.)

Scrivere una funzione ricorsiva che date due liste conteggi il numero di volte che la prima lista appare nella seconda.

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```c
/* given that: l1 contains {1, 3, 1}
/*             l2 contains {1, 3, 1, 3, 1, 1, 3, 1, 7} */
printf("%d \n", count_list(l1, l2) );
```

Output:

```
3 /* a partire dalle posizioni 0, 2 e 5 di l2*/
```

La firma della funzione è : `int count_list( list_t *a, list_t *b )`.

Soluzioni Esercizi

Prodotto Differenze

```c
float prod_diff( float *A, float *B, int size ) {
  /* assume size>0 */
  float r = 1.0f;
  int i;
  for ( i=0; i<size; i++)
    r *= A[i] - B[i];
  return r;
}
```

Matrice Stocastica

```c
int stocastic( float *M, int num_cols ) {
  /* assume num_cols >0 */
  int r, c;
  for (c=0; c<num_cols; c++) {
    float sum = 0;
    for (r=0; r<num_cols; r++) {
      if ( M[r*num_cols + c] < 0.0 ) return 0;
      sum += M[r*num_cols + c];
    }
    if (sum!=1.0)
      return 0;
  }
  return 1;
}
```

String Replicate

```c
char * replicate ( char *a, char *b ) {
  char *s = (char*) malloc(strlen(a)*2+1);
  char *i = a;
  char *t = s;
  while (*i) {
    char *j = b;
    int found = 0;
    while (!found && *j) {
      if (*i==*j) found = 1;
      j++;
    }
    if (found) {
      *t = *i;
      t++;
    }
    *t = *i;
    t++;
    i++;
  }
  *t = '\0';
```

```
  s = realloc(s, t-s+1);
  return s;
}
```

Lista Doppiamente crescente

```
int list_asc(list_t *l) {
  int last;
  if (!l) return 1;
  last = l->data;
  l = l->next;
  while (l) {
    if (last>l->data)
      return 0;
    last = l->data;
    l = l->next;
  }
  return 1;
}

int doppia_crescente(list_t *l) {
  int last;
  if (!l) return 1;
  last = l->data;
  l = l->next;
  while (l) {
    int right_asc = list_asc(l);
    if (right_asc)
      return 1;
    if (last>l->data)
      return 0;
    last = l->data;
    l = l->next;
  }
  return 1;
}
```

Count List

```
int is_prefixarray( int *a, int a_size, int *b, int b_size ) {
  int i = 0;
  while ( i<a_size && i<b_size ) {
    if (a[i]!=b[i])
      return 0;
    i++;
  }
  return i==a_size;
}

int count_array( int *a, int a_size, int *b, int b_size ) {
  if (b_size==0)
    return 0;
  return is_prefixarray(a, a_size, b, b_size) +
      count_array(a, a_size, b+1, b_size -1);
}
```

Appello del 28/01/2022

Istruzioni

Scrivere subito nome, cognome e matricola sul testo e su tutti i fogli protocollo. Sia il testo che i fogli dovranno essere riconsegnati.

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti è obbligatoria solo per chi non ha superato le esercitazioni. La seconda metà è obbligatoria per tutti. Le domande a risposta multipla possono avere più risposte corrette o nessuna. Usate il testo per rispondere.

Nella seconda parte lo studente dovrà risolvere solo 3 esercizi su 5 in 2 ore. Scrivete le soluzioni nei fogli protocollo.

Il punteggio finale è dato dalla somma dei punteggi.

Appello A

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 pt.)

Dato il seguente codice:

```
int x=0;
int f1(int w) {
  int z = w + ++x;
  {
    int x = z*2;
  }
  return x;
}
```

Qual è l'output della seguente istruzione?

```
printf( "%d %d\n", f1(10), f1(10) );
```

- ☐ 20 22
- ☐ 22 24
- ☐ 1 2
- ☐ 0 1

Q2 (1.5 pt.)

Dato il seguente codice:

```c
int * f2(int *x, int *y) {
  int z = *x + *y;
  return &z;
}
```

```c
int a = 1, b = 2;
int *c = f2(&a, &b);
printf("%d \n", *c);
```

Quali affermazioni sono corrette?

- ☐ E' errato restituire `&z`
- ☐ Avremmo dovuto scrivere `int *z = *x + *y;`
- ☐ Avremmo dovuto scrivere `int c = f2(&a, &b);`
- ☐ La funzione stampa 3

Q3 (1.5 pt.)

Dato il seguente codice:

```c
int diag(int M[3][3]) {
  int i, j;
  int sum = 0;
  for (i=0; i<3; i++)
    for (j=0; j<3; j++)
      if (i==j)
        sum += M[i][i];
  return sum;
}
```

Quali affermazioni sono corrette?

- ☐ La funzione calcola la somma degli elementi sulla diagonale di una matrice 3x3
- ☐ `M[i][i];` deve essere sostituito con `M[i][j];`
- ☐ `if (i==j)` viene eseguito 9 volte
- ☐ `if (i==j)` viene eseguito 3 volte

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 pt.)

Data una matrice rappresentata tramite un array di array costruito come segue, qual è il modo corretto per accedere all'elemento nella terza riga e seconda colonna?

```c
int **m = (int**) malloc(rows * sizeof(int*));
for (i=0; i<rows, i++)
    m[i] = (int*) malloc(cols * sizeof(int));
```

- ☐ `m[2*cols + 1]`
- ☐ `m[2][1]`
- ☐ `*(m + 2*cols + 1)`
- ☐ nessuno dei metodi sopra perché `m` è un doppio puntatore

Q5 (1.5 pt.)

Claudio ha scritto una funzione per verificare che, date due stringhe, la prima sia inclusa nella seconda. Quali affermazioni sono corrette?

```c
int included(char *small, char *large) {
    while (*large) {
        int i=0;
        while (small[i] && large[i] && small[i] == large[i])
            i++;
        if (!small[i])
            return 1;
         large++;
    }
    return 0;
}
```

- ☐ la funzione ritorna 1 se `small` è inclusa in `large`, e 0 altrimenti
- ☐ la funzione ritorna 1 se `large` è inclusa in `small`, e 0 altrimenti
- ☐ i puntatori del chiamante alle due stringhe vengono modificati
- ☐ il codice è corretto solo se `small` è più corta di `large`

Q6 (1.5 pt.)

Claudio ha scritto una funzione per verificare se la stringa in input `a` sia prefisso della stringa in input `b`. Quali affermazioni sono corrette?

```c
int is_prefix(char *a, char *b) {
    if ( *a=='\0' )
        return 1;
    return *a==*b && is_prefix (a+1, b+1);

}
```

- ☐ La funzione non ritorna mai 0
- ☐ La ricorsione potrebbe non terminare mai perché non ci sono controlli sulla fine della stringa `b`
- ☐ L'implementazione è corretta
- ☐ L'implementazione è sbagliata

Soluzioni Domande

Q1

- ☒ 1 2

Q2

- ☒ E' errato restituire `&z`

Q3

- ☒ La funzione calcola la somma degli elementi sulla diagonale di una matrice 3x3
- ☒ `if (i==j)` viene eseguito 9 volte

Q4

☒ `m[2][1]`

Q5

☒ la funzione ritorna 1 se `small` è inclusa in `large`, e 0 altrimenti

Q6

☒ L'implementazione è corretta

Esercizi

Smallest (4 pt.s)

Scrivere una funzione `smallest` che dato un array di interi maggiori di 0 restituisca il più piccolo numero pari in esso contenuto, o restituisca `-1` nel caso non ci siano numeri pari nell'array.

Esempio:

```c
int a[] = {10, 2, 3, 8, 1};
int b[] = {1, 3, 5, 7, 9};
printf("%d", smallest(a, 5));
printf("%d", smallest(b, 5));
```

Output:

```
2
-1
```

La firma della funzione è : `int smallest( int *A, int size ).`

Matrice Dominante (5 pt.s)

Una matrice si dice dominante se ogni elemento è maggiore di tutti gli altri elementi nelle righe inferiori della sua stessa colonna. Scrivere una funzione `dominante` che data una matrice restituisce `1` se questa è dominante e `0` altrimenti.

Esempio:

```c
int M[3][3] = { {3,4,5},
                {2,3,3},
                {1,8,1} };
int N[2][3] = { {3,4,5},
                {2,3,3}};
printf("%d\n", dominante( &(M[0][0]), 3, 3) );
printf("%d\n", dominante( &(N[0][0]), 2, 3) );
```

Output:

```
0 /* 8 non è minore di 3 e 4 */
1
```

La firma della funzione è : `int dominante( int *M, int n_rows, int n_cols )`.

No doppie (6 pt.s)

Scrivere una funzione `clean` che data una stringa, restituisca una nuova stringa dove occorrenze consecutive della stessa lettera sono state ridotte ad una sola occorrenza.

Esempio:

```
1 char * a = "ccccccassaa";
2 printf("%s\n", clean( a ) );
```

Output:

```
casa
```

La firma della funzione è : `char * clean( char *s )`.

Merge (7 pt.s)

Date due liste di interi ordinate in maniera crescente, scrivere una funzione `merge` che costruisca una lista interamente nuova con tutti gli elementi delle due liste ordinati in maniera crescente.

Esempio:

```
1 typedef struct list {
2     int data;
3     struct list *next; /* next == NULL at the end of the list */
4 } list_t;
5
6 /* given that: a contains {1,2,3,4,5} and
7                b contains {1,2,7,9} */
8 list_t *c = merge(a,b);
9 /* c now contains {1,1,2,2,3,4,5,7,9} */
```

La firma della funzione è : `list_t * merge( list_t *a, list_t *b )`.

Combinazioni (8 pt.s)

Scrivere una funzione **ricorsiva** `comb` che dato un array di int `A`, un intero `len` e un int `target`, restituisca il numero di sotto-insiemi di `A` (elementi anche non consecutivi) di grandezza `len` la cui somma è pari a `target`.

Esempio:

```
1 int A[] = { 10, 10, 20, 5, 15 };
2 int B[] = { 10, 10, 10, 20 };
3 int target = 40;
4 int len = 3;
5 printf("%d\n", comb( A, 5, len, target ) );
6 printf("%d\n", comb( B, 4, len, target ) );
```

Output:

```
2 /* (10, 10, 20), (20, 5, 15)  */
3 /* (B[0], B[1], B[3]), (B[0], B[2], B[3]), (B[1], B[2], B[3])  */
```

La firma della funzione è : `int comb( int *A, int A_size, int len, int target )`.

Soluzioni Esercizi

Smallest

```
1  int smallest( int *A, int size ) {
2    int smallest_even;
3    int found = 0;
4    if (size>0) {
5      int i;
6      for ( i=0; i<size; i++) {
7        if (A[i]%2==0) {
8          if (!found) {
9            found = 1;
10           smallest_even = A[i];
11         } else {
12           if (smallest_even > A[i])
13             smallest_even = A[i];
14         }
15       }
16     }
17   }
18   if (!found)
19     return -1;
20   else
21     return smallest_even;
22 }
```

Matrice Dominante

```
1  int dominante( int *M, int n_rows, int n_cols ) {
2    int r, c;
3    for (c=0; c<n_cols; c++) {
4      for (r=1; r<n_rows; r++) {
5        if ( M[(r-1)*n_cols + c] <= M[r*n_cols + c] )
6          return 0;
7      }
8    }
9    return 1;
10 }
```

No doppie

```
1  char * clean ( char *s ) {
2    char *new_s = (char*) malloc(strlen(s)+1);
3    char *t = new_s;
4    while (*s) {
5      if ( t==new_s || /* first character */
6           *s != *(t-1) ) {
```

```
      *t = *s;
      t++;
    }
    s++;
  }
  *t = '\0';
  new_s = realloc(new_s, t-new_s+1);
  return new_s;
}
```

Merge

```
list_t * merge(list_t *a, list_t *b) {
  list_t *c = NULL, *tail = NULL;
  while ( a || b ) { /* at least one */
    list_t *smaller = NULL;
    if ( a && !b )
      smaller = a;
    else if ( !a && b )
      smaller = b;
    else if ( a->data < b->data)
      smaller = a;
    else
      smaller = b;
    if ( !c ) { /* list is still empty */
      tail = c = (list_t *) malloc(sizeof(list_t));
    } else {
      tail->next = (list_t *) malloc(sizeof(list_t));
      tail = tail->next;
    }
    tail->data = smaller->data;
    tail->next = NULL;

    if (smaller==a)
      a = a->next;
    else
      b = b->next;
  }

  return c;
}
```

Combinazioni

```
int comb( int *A, int A_size, int len, int target ) {
  if (target==0 && len==0)
    return 1;
  if (A_size==0 || len==0)
    return 0;
  return comb(A+1, A_size-1, len, target) +
         comb(A+1, A_size-1, len-1, target-A[0]);
}
```

Appello B

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 pt.)

Dato il seguente codice:

```
int z=0;
int g1(int w) {
  {
    int z = 10 + ++w;
  }
  return z + w;
}
```

Qual è l'output della seguente istruzione?

```
printf( "%d %d\n", g1(10), g1(10) );
```

- ☐ 21 32
- ☐ 21 22
- ☐ 21 21
- ☐ 11 11

Q2 (1.5 pt.)

Dato il seguente codice:

```
int g2(int *x, int *y) {
  int *z = *x + *y;
  return *z;
}
```

```
int a = 1, b = 2;
int c = g2(&a, &b);
printf("%d \n", c);
```

Quali affermazioni sono corrette?

- ☐ E' errato dichiarare `int *z`
- ☐ Avremmo dovuto scrivere `printf("%d \n", &c);`
- ☐ Avremmo dovuto scrivere `int * g2(int *x, int *y){`
- ☐ Avremmo dovuto scrivere `return z;`

Q3 (1.5 pt.)

Dato il seguente codice:

```
int diag(int M[3][3]) {
  int i, j;
  int sum = 0;
```

```
4   for (i=0; i<3; i++)
5     for (j=0; j<3; j++)
6       if (i==j)
7         sum += M[i][i];
8   return sum;
9 }
```

Quali affermazioni sono corrette?

- ☐ Le righe 5 e 6 possono essere eliminate
- ☐ il codice non compila per via delle parentesi graffe mancanti
- ☐ avremmo potuto scrivere `int diag(int M[][]){` per gestire matrici di qualsiasi dimensione
- ☐ Il codice calcola la somma degli elementi nella i-esima riga della matrice `M`.

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 pt.)

Data una matrice rappresentata tramite un array di array costruito come segue, qual è il modo corretto per accedere all'elemento nella terza riga e seconda colonna?

```
1 int **m = (int**) malloc(rows * sizeof(int*));
2 for (i=0; i<rows, i++)
3     m[i] = (int*) malloc(cols * sizeof(int));
```

- ☐ `*(m+2)(m+1)`
- ☐ `m[2*rows + 1]`
- ☐ `*(m + 2*rows + 1)`
- ☐ nessuno dei metodi sopra

Q5 (1.5 pt.)

Claudio ha scritto una funzione per verificare che, date due stringhe, la prima sia inclusa nella seconda. Quali affermazioni sono corrette?

```
1  int included(char *small, char *large) {
2      while (*large) {
3          int i=0;
4          while (small[i] && large[i] && small[i] == large[i])
5              i++;
6          if (!small[i])
7              return 1;
8           large++;
9      }
10     return 0;
11 }
```

- ☐ la funzione ritorna 1 se `small` è inclusa in `large`, e 0 altrimenti
- ☐ la funzione ritorna 1 se `large` è inclusa in `small`, e 0 altrimenti
- ☐ il test `small[i] == large[i]` potrebbe accedere oltre la fine di una delle stringhe
- ☐ dovremmo cambiare `if (!small[i])` in `if (!small[i] && !large[i])`

Q6 (1.5 pt.)

Claudio ha scritto una funzione per verificare se la stringa in input `a` sia prefisso della stringa in input `b`. Quali affermazioni sono corrette?

```
int is_prefix(char *a, char *b) {
    if ( ! *a )
        return 1;
    if ( ! *b )
        return 0;
    return *a==*b && is_prefix (a+1, b+1);

}
```

- ☐ se `a` e `b` sono esattamente identiche, la ricorsione supera la fine delle stringhe
- ☐ l'ordine dei due `if` è fondamentale
- ☐ L'implementazione è corretta
- ☐ L'implementazione è sbagliata

Soluzioni Domande

Q1

- ☒ 11 11

Q2

- ☒ E' errato dichiarare `int *z`
- ☒ Avremmo dovuto scrivere `return z;`

Q3

- ☒ Le righe 5 e 6 possono essere eliminate

Q4

- ☒ nessuno dei metodi sopra

Q5

- ☒ la funzione ritorna 1 se `small` è inclusa in `large`, e 0 altrimenti

Q6

- ☒ l'ordine dei due `if` è fondamentale
- ☒ L'implementazione è corretta

Esercizi

Largest (4 pt.s)

Scrivere una funzione `largest` che dato un array di interi maggiori di 0 restituisca il più grande numero pari in esso contenuto, o restituisca `-1` nel caso non ci siano numeri pari nell'array.

Esempio:

```
int a[] = {10, 2, 3, 8, 1};
int b[] = {1, 3, 5, 7, 9};
printf("%d", largest(a, 5));
printf("%d", largest(b, 5));
```

Output:

```
10
-1
```

La firma della funzione è : `int largest( int *A, int size )`.

Matrice Dominata (5 pt.s)

Una matrice si dice dominata se ogni elemento è minore di tutti gli altri elementi nelle colonne successive della sua stessa riga. Scrivere una funzione `dominata` che data una matrice restituisce `1` se questa è dominata e `0` altrimenti.

Esempio:

```
int M[3][3] = { {1,2,3},
                {4,5,1},
                {4,7,9} };
int N[2][3] = { {1,2,3},
                {1,5,9}};
printf("%d\n", dominata( &(M[0][0]), 3, 3) );
printf("%d\n", dominata( &(N[0][0]), 2, 3) );
```

Output:

```
0 /* 1 nella seconda riga non è maggiore di 4 e 5 */
1
```

La firma della funzione è : `int dominata( int *M, int n_rows, int n_cols )`.

No ripetizioni (6 pt.s)

Scrivere una funzione `cut` che data una stringa, restituisca una nuova stringa dove compare solo la prima occorrenza di ciascun carattere.

Esempio:

```
char * a = "casaccio";
printf("%s\n", cut( a ) );
```

```

*Output:*
```{style=plain}
casio
```
```

La firma della funzione è : `char * cut( char *s )`.

Union (7 pt.s)

Date due liste di interi ordinate in maniera crescente, scrivere una funzione `list_union` che costruisca una lista interamente nuova con tutti gli elementi delle due liste senza doppioni. *(non è richiesto di mantenere l'ordinamento)*

Esempio:

```c
typedef struct list {
    int data;
    struct list *next; /* next == NULL at the end of the list */
} list_t;

/* given that: a contains {1,2,3,4,5} and
               b contains {1,2,7,9} */
list_t *c = list_union(a,b);
/* c now contains {1,2,3,4,5,7,9} */
```

La firma della funzione è : `list_t * list_union( list_t *a, list_t *b )`.

Combinazioni (8 pt.s)

Scrivere una funzione **ricorsiva** `count` che dato un array di int `A`, un intero `len` e un int `target`, restituisca il numero di sotto-insiemi di `A` (elementi anche non consecutivi) di grandezza strettamente minore di `len` la cui somma è pari a `target`.

Esempio:

```c
int A[] = { 10, 30, 20, 5, 15 };
int B[] = { 10, 10, 10, 20 };
int target = 40;
int len = 4;
printf("%d\n", count( A, 5, len, target ) );
printf("%d\n", count( B, 4, len, target ) );
```

Output:

```
2 /* (10, 30), (20, 5, 15)  */
3 /* (B[0], B[1], B[3]), (B[0], B[2], B[3]), (B[1], B[2], B[3])  */
```

La firma della funzione è : `int count( int *A, int A_size, int len, int target )`.

Soluzioni Esercizi

Largest

```c
int max(int a, int b) {
  return (a>b)?a:b;
}

int largest( int *A, int size ) {
  int largest_even;
  int found = 0;
  if (size>0) {
```

```
    int i;
    for ( i=0; i<size; i++ ) {
      if (A[i]%2==0) {
        largest_even = found ? max(A[i], largest_even) : A[i];
        found = 1;
      }
    }
  }
  return found ? largest_even : -1;
}
```

Matrice Dominata

```
int dominata( int *M, int n_rows, int n_cols ) {
  int r, c;
  for (r=0; r<n_rows; r++) {
    for (c=1; c<n_cols; c++) {
      if ( M[r*n_cols + c-1] >= M[r*n_cols + c] )
        return 0;
    }
  }
  return 1;
}
```

No ripetizioni

```
int is_in(char a, char *s) {
  while (*s) {
    if (a==*s)
      return 1;
    s++;
  }
  return 0;
}

char * cut ( char *s ) {
  char *new_s = (char*) malloc(strlen(s)+1);
  char *t = new_s;
  *t = '\0';
  while (*s) {
    if ( !is_in(*s, new_s) ) {
      *t = *s;
      *(t+1) = '\0';
      t++;
    }
    s++;
  }
  new_s = realloc(new_s, t-new_s+1);
  return new_s;
}
```

Union

```
list_t * list_union(list_t *a, list_t *b) {
  list_t *c = NULL, *tail = NULL;
```

```
  while ( a || b ) { /* at least one */
    list_t *smaller = NULL;
    if ( a && !b )
      smaller = a;
    else if ( !a && b )
      smaller = b;
    else if ( a->data < b->data)
      smaller = a;
    else
      smaller = b;
    if ( !c ) { /* list is still empty */
      tail = c = (list_t *) malloc(sizeof(list_t));
      tail->data = smaller->data;
      tail->next = NULL;
    } else if ( tail->data != smaller->data ) {
      tail->next = (list_t *) malloc(sizeof(list_t));
      tail = tail->next;
      tail->data = smaller->data;
      tail->next = NULL;
    }

    if (smaller==a)
      a = a->next;
    else
      b = b->next;
  }

  return c;
}
```

Combinazioni

```
int count( int *A, int A_size, int len, int target ) {
  if (target==0 && len>0)
    return 1;
  if (A_size==0 || len==0 )
    return 0;
  return count(A+1, A_size-1, len, target) +
         count(A+1, A_size-1, len-1, target-A[0]);
}
```

Appello del 17/06/2022

Istruzioni

Scrivere subito nome, cognome e matricola sul testo e su tutti i fogli protocollo. Sia il testo che i fogli dovranno essere riconsegnati.

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti é obbligatoria solo per chi non ha superato le esercitazioni. La seconda metà é obbligatoria per tutti. Le domande a risposta multipla possono avere più risposte corrette o nessuna. Usate il testo per rispondere.

Nella seconda parte lo studente dovrà risolvere solo 3 esercizi su 5 in 2 ore. Scrivete le soluzioni nei fogli protocollo.

Il punteggio finale è dato dalla somma dei punteggi.

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 pt.)

Dato il seguente codice:

```
1  int x=10;
2  int f1(int w) {
3    if (x%2==0) {
4      int x = 1;
5      return w + x;
6    } else {
7      x = 2;
8      return w - x;
9    }
10 }
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
1  printf( "%d %d\n", f1(10), f1(10) );
```

- ☐ il codice non compila perché non è possibile dichiarare nuovamente `int x` alla linea 4
- ☐ `11 11`
- ☐ `11 8`
- ☐ il codice non compila perché manca uno statement return

Q2 (1.5 pt.)

Data la seguente funzione:

```
1  int f2(int *x, int *y, int *z) {
2    *x = *y + 1;
3    *y = *z + 1;
```

```
  *z = *x;
  return x;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
int a = 1, b = 2, c = 3;
printf("%d \n", f2(&a, &b, &c));
```

- ☐ 3
- ☐ 2
- ☐ il tipo di ritorno della funzione è diverso dal tipo dell'espressione ritornata
- ☐ nessuna della risposte precedenti

Q3 (1.5 pt.)

Dato il seguente codice, quali delle seguenti affermazioni sono vere?

```
void f3(int M[3][3]) {
  int i, j;
  for (i=0; i<3; i++)
    for (j=0; j<3; j++)
      M[i][j] = M[j][i];
}
```

- ☐ la funzione traspone la matrice ricevuta come argomento
- ☐ la funzione traspone una copia della matrice ricevuta come argomento
- ☐ la funzione traspone due volte la matrice ricevuta come argomento con l'effetto di lasciarla invariata
- ☐ nessuna delle risposte precedenti

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 pt.)

Claudio ha scritto una funzione che data una stringa, ne crea una copia. Quali errori ci sono nella funzione qui sotto?

```
char * string_copy(char *s) {
  char *d = malloc(strlen(s));
  char *d_copy = d;
  while (*s!='\0') {
    *d_copy = *s;
    d_copy++;
    s++;
  }
  *d_copy = '\0';
  return d;
}
```

- ☐ il ciclo while non termina mai
- ☐ la funzione dovrebbe ritornare `d_copy` al posto di `d`
- ☐ la memoria allocata non è sufficiente
- ☐ la stringa `d` non viene mai modificata

Q5 (1.5 pt.)

Cosiderando i tipi definiti sotto, qual è il modo corretto per accedere ad uno dei voti dello studente `S`?

```c
struct mark {
  char* course_name;
  int score;
};

struct career {
  struct mark *marks;
  int num_marks;
};

struct student {
  char* name;
  char* surname;
  int id;
  struct career c;
};

struct student S;
/* altre inizializzazioni */
```

- ☐ `printf("%d", S->career->marks->score)`
- ☐ `printf("%d", S->career->marks[0]->score)`
- ☐ `printf("%d", S.c.marks[0].score)`
- ☐ `printf("%d", S.c.marks.score)`

Q6 (1.5 pt.)

Claudio ha scritto una funzione per individuare se in un array di interi è presente una tripla di elementi, anche non consecutivi, la cui somma è pari a 0. Quali delle seguenti affermazioni sono vere?

```c
int f6 (int *A, int A_size) {
  int i, j, k;
  for (i=0; i<A_size; i++)
    for (j=0; j<A_size; j++)
      for (k=0; k<A_size; k++)
        if ( A[i]+A[j]+A[k] == 0 )
          return 1;
  return 0;
}
```

- ☐ il codice funziona solo se `A` ha almeno 3 elementi
- ☐ il codice esegue delle operazioni ridondanti ma è corretto
- ☐ il codice ritorna 1 se `A` contiene uno 0
- ☐ la funzione restituisce sempre 0

Esercizi

Es1 (4 pt.s)

Scrivere una funzione `euclidean` che, dati due array di float `A` e `B` aventi la stessa lunghezza diversa da zero, restituisca il quadrato della loro distanza euclidea calcolato come segue:

$$euclidean(A, B) = \sum_i (A[i] - B[i])^2$$

Esempio:

```c
float A[] = { 1.0f, 2.0f, 4.0f, 8.0f };
float B[] = { 1.0f, 1.0f, 4.0f, 4.0f };

printf( "%f \n", euclidean(A, 4, B, 4));
```

Output:

```
17.0  /* perché (1-1)^2 + (2-1)^1 + (4-4)^2 + (8-4)^2 = 17 */
```

La firma della funzione è : `float euclidean( float *A, int A_size, float *B, int B_size )`.

Es2 (5 pt.s)

Una matrice è detta incrociata se le sue due diagonali hanno gli stessi valori nello stesso ordine. Scrivere una funzione che data una matrice quadrata di interi `A` restituisca uno 1 se questa è incrociata e 0 altrimenti.

Esempio:

```c
int M[3][3] = { {1,5,1},
                {0,2,0},
                {3,7,3} };
printf("%d\n", incrociata( &(M[0][0]), 3, 3) );
```

Output:

```
1 /* entrambe le diagonali contengono i valori 1, 2, 3 */
```

La firma della funzione è : `int incrociata( int *M, int n_rows, int n_cols )`.

Es3 (6 pt.s)

Scrivere una funzione che data una stringa, restituisca una nuova stringa dove ogni spazio è stato sostituito con un doppio asterisco.

Esempio:

```c
char * a = "I Love Coding";
printf("%s\n", asterisco( a ) );
```

Output:

```
I**Love**Coding
```

La firma della funzione è : `char * asterisco( char *s )`.

Es4 (7 pt.s)

Scrivere una funzione che data una lista restituisca una nuova lista (fatta di nuovi nodi) che sia inversa rispetto alla prima.

La struttura dati lista è definita come segue:

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```c
/* se l1 = {1,2,3,4} */
list_t * l2 = reverse(l1)
/* l2 = {4,3,2,1} */
```

La firma della funzione è : `list_t * reverse(list_t *l)`.

Es5 (8 pt.s)

Domino Colorato. Abbiamo una scorta infinita di pedine colorate con 3 colori diversi con i quali dobbiamo costruire una sequenza composta da n pedine in totale. Dobbiamo rispettare un vincolo: due pedine dello stesso colore non possono stare vicine. Scrivere una funzione che dato n restituisca il numero di sequenze possibili di lunghezza n che rispettano questo vincolo.

Esempio:

```c
  printf("%d \n", domino(1) );
  printf("%d \n", domino(3) );
```

Output:

```
3
12
/* se chiamiamo i colori r,g,b

  nel primo caso abbiamo le 3 configurazioni
   r, g, b
  nel secondo caso abbiamo le 12 configurazioni
   rgr, rgb, rbr, rbg,
   grg, grb, gbr, gbg,
   brg, brb, bgr, bgb
```

La firma della funzione è : `int domino(int n)`. E' possibile usare altre funzioni accessorie per implementare la ricorsione.

Soluzioni

Q1

☒ 11 11

Q2

☒ il tipo di ritorno della funzione è diverso dal tipo dell'espressione ritornata

Q3

☒ nessuna delle risposte precedenti

Q4

☒ la memoria allocata non è sufficiente

Q5

☒ `printf("%d", S.c.marks[0].score)`

Q6

☒ il codice ritorna 1 se `A` contiene uno 0

Es1

```c
float euclidean( float *A, int A_size, float *B, int B_size ) {
  float e = 0.0f;
  int i;
  for (i=0; i<A_size; i++)
    e += (A[i]-B[i])*(A[i]-B[i]);
  return e;
}
```

Es2

```c
int incrociata( int *M, int n_rows, int n_cols ) {
  int row;
  for (row = 0; row < n_rows; row++) {
    int col = n_cols - row - 1;
    if ( M[row*n_cols + row] !=  M[row*n_cols + col] )
      return 0;
  }
  return 1;
}
```

Es3

```c
char * asterisco( char *s ) {
  size_t i=0, j=0;
  char * c = (char*) malloc ( 2*strlen(s) + 1 );
  if (!c) return NULL;
  for (i=0; i<strlen(s); i++) {
```

```
    if ( s[i]==' ' ) {
      c[j++] = '*';
      c[j++] = '*';
    } else {
      c[j++] = s[i];
    }
  }
  c[j++] = '\0';
  c = realloc (c, j);
  return c;
}
```

Es4

```
list_t * reverse(list_t *l) {
  list_t *reversed = NULL;
  while (l) {
    /* put first element of l at the begining of rev */
    list_t *new_node = (list_t*) malloc(sizeof(list_t));
    if (!new_node) exit(EXIT_FAILURE);

    new_node->data = l->data;
    new_node->next = reversed;
    reversed = new_node;

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

Es5

```
int domino_rec(int n, int prev) {
  int count = 0;
  int color;

  if (n==0) return 1; /* completed */

  for (color=1; color<=3; color++) {
    if (color!=prev)
      count += domino_rec(n-1, color);
  }
  return count;
}

int domino(int n) {
  return domino_rec(n, 0);
}
```

Appello del 06/09/2022

Istruzioni

Scrivere subito nome, cognome e matricola sul testo e su tutti i fogli protocollo. Sia il testo che i fogli dovranno essere riconsegnati.

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti é obbligatoria solo per chi non ha superato le esercitazioni. La seconda metà é obbligatoria per tutti. Le domande a risposta multipla possono avere più risposte corrette o nessuna. Usate il testo per rispondere.

Nella seconda parte lo studente dovrà risolvere solo 3 esercizi su 5 in 2 ore. Scrivete le soluzioni nei fogli protocollo.

Il punteggio finale è dato dalla somma dei punteggi.

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 pt.)

Dato il seguente codice:

```
int n = 10;
int f1(int n) {
  int i, sum = 0;
  for ( i=0; i<n; i++) {
    sum += n;
  }
  return sum;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
printf( "%d %d\n", f1(10), f1(6) );
```

- ☐ `100 36`
- ☐ `100 60`
- ☐ `110 42`
- ☐ il codice non compila perché non è possibile dichiarare nuovamente `int n` come parametro della funzione

Q2 (1.5 pt.)

Data la seguente funzione:

```
float * f2(float *base, int *exp) {
  float pow = 1;
  int i;
  for ( i=0; i< *exp; i++ )
```

```
    pow *= *base;
  return &pow;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
float a = 10;
int b = 3;
float * c = f2(&a, &b);
printf("%.2f \n", *c );
```

- ☐ `1000.0`
- ☐ E' errato restituire `&pow`
- ☐ L'invocazione corretta è `float c = f2(a, b);`
- ☐ Avremmo dovuto scrivere `float * pow = 1;`

Q3 (1.5 pt.)

Dato il seguente codice, quali delle seguenti affermazioni sono vere?

```
int f3(int M[3][3]) {
  int i, j, sum=0;
  for (i=0; i<3; i++)
    for (j=0; j<=i; j++)
      sum += M[i][j];
  return sum;
}
```

- ☐ la funzione calcola la somma degli elementi nella diagonale della matrice e sotto di essa
- ☐ la funzione calcola la somma degli elementi sotto la diagonale della matrice (diagonale esclusa)
- ☐ `sum += M[i][j];` viene eseguito 9 volte
- ☐ `sum += M[i][j];` viene eseguito 3 volte

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 pt.)

Claudio ha scritto una funzione per verificare se due stringhe siano uguali. Quali affermazioni sono corrette?

```
int string_equal(char *a, char *b) {
  int i, j;
  for (i=0; i<strlen(a); i++) {
    for (j=0; j<strlen(b); j++) {
      if ( a[i]!=b[j] )
        return 0;
    }
  }
  return 1;
}
```

- ☐ la funzione è corretta solo se le stringhe hanno la stessa lunghezza
- ☐ la funzione non ritorna mai `1`

- ☐ E' sufficiente rimuovere il ciclo `for` più interno
- ☐ la funzione è corretta, ma potrebbe essere implementata anche con un solo ciclo for

Q5 (1.5 pt.)

Data una matrice `m` rappresentata tramite un array di array costruito come segue, quali affermazioni sono corrette?

```
int **m = (int**) malloc(rows * sizeof(int*));
for (i=0; i<rows, i++)
    m[i] = (int*) malloc(cols * sizeof(int));
```

- ☐ Non è consentito dichiarare un doppio puntatore `int **m`
- ☐ `m[i]` è il puntatore al primo elemento della riga `i`-esima
- ☐ `free(m)` de-alloca la memoria dell'intera matrice
- ☐ `m[i*cols + j]` accede all'elemento nella `i`-esima riga e `j`-esima colonna

Q6 (1.5 pt.)

Claudio ha scritto una funzione per individuare se una stringa appare come sottostringa di un altra (es. "tipa" è sottostringa di "prototipare"). Quali delle seguenti affermazioni sono corrette?

```
int f6 (char *a, char *b) {
  int i, j;
  for (i=0; i<strlen(b); i++) {
    int is_substring = 1;
    for (j=0; j<strlen(a); j++)
      is_substring = is_substring && a[j]==b[i+j];
    if (is_substring)
      return 1;
  }
  return 0;
}
```

- ☐ la funzione è corretta e verifica se `a` è sottostringa di `b`
- ☐ la funzione è corretta e verifica se `b` è sottostringa di `a`
- ☐ la funzione non è corretta perché potrebbe accedere oltre la fine della stringa `b`
- ☐ la funzione restituisce sempre 1

Esercizi

Es1 (4 pt.s)

Scrivere una funzione `delta` che, dato un array di interi `A` di lunghezza diversa da zero, restituisca la differenza tra il suo elemento massimo e il suo elemento minimo.

Esempio:

```
int A[] = { 1,5,8,3 };
printf( "%d \n", delta(A, 4));
```

Output:

```
7  /* perché 8-1 = 7 */
```

La firma della funzione è : `int delta( int *A, int A_size )`.

Es2 (5 pt.s)

Una matrice quadrata è detta triangolare inferiore se gli elementi al di sopra della diagonale principale sono tutti nulli. Scrivere una funzione che data una matrice quadrata di interi `A` restituisca uno 1 se questa è triangolare inferiore e 0 altrimenti.

Esempio:

```
int M[3][3] = { {1,0,0},
                {3,2,0},
                {3,7,3} };
printf("%d\n", tri_inf( &(M[0][0]), 3) );
```

Output:

```
1 /* i valori sopra la diagonale sono tutti pari a 0 */
```

La firma della funzione è : `int tri_inf( int *M, int n_rows )`.

Es3 (6 pt.s)

Scrivere una funzione che data una stringa, restituisca una nuova stringa dove i caratteri tra due asterischi consecutivi sono stati rimossi assieme agli asterischi stessi. Assumiamo che la stringa in input abbia sempre un numero pari di asterischi.

Esempio:

```
char * a = "I*do not* Love *wasting my time at*Coding *badly*";
printf("%s\n", asterischi( a ) );
```

Output:

```
I Love Coding
```

La firma della funzione è : `char * parentesi( char *s )`.

Es4 (7 pt.s)

Scrivere una funzione che data una lista di interi restituisca una nuova lista (fatta di nuovi nodi) con soli gli elementi distinti della prima (in altre parole, tolti i duplicati).

La struttura dati lista è definita come segue:

```
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```
/* se l1 = {1,1,2,3,1,2,4,4} */
list_t * l2 = unique(l1)
/* l2 = {1,2,3,4} */
```

La firma della funzione è : `list_t * unique(list_t *l)`.

Es5 (8 pt.s)

Scrivere una funzione **ricorsiva** che dati due array di interi conteggi il numero di volte che il primo array appare nel secondo anche con sovrapposizioni.

Esempio:

```c
int v1[] = {7, 3, 7};
int v2[] = {7, 3, 7, 3, 7, 7, 3, 7, 0};
printf("%d \n", count_array(v1, 3, v2, 9) );
```

Output:

```
3 /* a partire dalle posizioni 0, 2 e 5 di v2*/
```

La firma della funzione è : `int count_array( int *a, int a_size, int *b, int b_size )`.

Soluzioni

Q1

☒ `100 36`

Q2

☒ E' errato restituire `&pow`

Q3

☒ la funzione calcola la somma degli elementi nella diagonale della matrice e sotto di essa

Q4

Nessuna risposta è corretta

Q5

☒ `m[i]` è il puntatore al primo elemento della riga `i`-esima

Q6

☒ la funzione non è corretta perché potrebbe accedere oltre la fine della stringa `b`

Es1

```c
int delta( int *A, int A_size ) {
  int minimum = A[0], maximum = A[0];
  int i, delta;

  for ( i=1; i<A_size; i++) {
    if ( A[i]<minimum )
      minimum = A[i];
    if ( A[i]>maximum )
      maximum = A[i];
  }

  delta = maximum - minimum;
  return delta;
}
```

Es2

```c
int tri_inf( int *M, int n_rows ) {
  int i,j;
  for ( i=0; i<n_rows; i++ ) {
    for ( j=i+1; j<n_rows; j++ ) {
      if ( M[i*n_rows + j]!=0 )
        return 0;
    }
  }
  return 1;
}
```

Es3

```c
char * asterischi( char *s ) {
  size_t i=0, j=0;
  int keep = 1;
  char * c = (char*) malloc ( strlen(s) + 1 );
  if (!c) exit(EXIT_FAILURE);
  for (i=0; i<strlen(s); i++) {
    if ( s[i]=='*' ) {
      keep = !keep;
    } else {
      if (keep)
        c[j++] = s[i];
    }
  }
  c[j++] = '\0';
  c = realloc (c, j);
  return c;
}
```

Es4

```c
int is_in(int x, list_t *l) {
  while (l) {
```

```
    if (l->data==x)
      return 1;
    l = l->next;
  }
  return 0;
}

list_t * unique(list_t *l) {
  list_t *uniq = NULL;
  list_t *uniq_tail = NULL;

  while ( l!=NULL ) {
    if ( ! is_in(l->data, uniq) ) {
      list_t *new_node = (list_t*) malloc(sizeof(list_t));
      if (!new_node) exit(EXIT_FAILURE);
      new_node->data = l->data;
      new_node->next = NULL;

      if ( uniq==NULL ) {
        uniq = uniq_tail = new_node;
      } else {
        uniq_tail->next = new_node;
        uniq_tail = uniq_tail->next;
      }
    }
    l = l->next;
  }

  return uniq;
}
```

Es5

```
int is_prefixarray( int *a, int a_size, int *b, int b_size ) {
  int i = 0;
  while ( i<a_size && i<b_size ) {
    if (a[i]!=b[i])
      return 0;
    i++;
  }
  return i==a_size;
}

int count_array( int *a, int a_size, int *b, int b_size ) {
  if (b_size==0)
    return 0;
  return is_prefixarray(a, a_size, b, b_size) +
      count_array(a, a_size, b+1, b_size -1);
}
```

Appello A del 13/01/2023

Istruzioni

Scrivere subito nome, cognome e matricola sul testo e su tutti i fogli protocollo. Sia il testo che i fogli dovranno essere riconsegnati.

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti é obbligatoria solo per chi non ha superato le esercitazioni. La seconda metà é obbligatoria per tutti. Le domande a risposta multipla possono avere più risposte corrette o nessuna. Usate il foglio di testo per rispondere.

Nella seconda parte lo studente dovrà risolvere e consegnare solo 3 esercizi su 5 in 2 ore. Scrivete le soluzioni nei fogli protocollo.

Il punteggio finale è dato dalla somma dei punteggi.

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 pt.)

Dato il seguente codice:

```
int x=3;
int f1 (int w) {
  if ( w%2 == 0 ) {
    return x * 3;
  } else {
    int x = w * 2;
  }
  return x;
}
```

Qual è l'output del codice seguente?

```
printf( "%d ", f1(x++) );
printf( "%d ", f1(x++) );
```

- ☐ 6 15
- ☐ 4 15
- ☐ 12 5
- ☐ 12 15

Q2 (1.5 pt.)

Dato il seguente codice:

```c
int* f2 (int a, int b, int c) {
  int *x = &a;
  int *y = &b;
  int *z = &c;
  *x = *y + *z;
  return x;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```c
printf( "%d \n", f2(1,2,3) );
```

- ☐ L'output è `5` pari a `2 + 3`
- ☐ Non si possono sommare i due puntatori nell'istruzione `*y + *z`
- ☐ Non si può scrivere `&a`, `&b`, `&c`, perchè gli argomenti `1`, `2`, `3` sono costanti e non hanno un indirizzo come le variabili
- ☐ Non è corretto restituire l'indirizzo della variabile `a`

Q3 (1.5 pt.)

Dato il codice seguente, quali affermazioni sono corrette?

```c
int diag ( int *M, int n_rows, int n_cols ) {
  int sum = 0;
  for (int i=0; i<n_rows; i++)
    for (int j=0; j<n_cols; j++)
      if ( i==j )
        sum += M [ i*n_cols ];
  return sum;
}
```

- ☐ la funzione calcola la somma degli elementi nella prima colonna della matrice
- ☐ la funzione calcola la somma degli elementi nella prima riga della matrice
- ☐ la funzione calcola la somma di tutti gli elementi della matrice
- ☐ la funzione calcola la somma degli elementi nella diagonale principale di una matrice quadrata

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 pt.)

Dato codice seguente, quali istruzioni *non* compilano correttamente?

```c
struct A {
  int *data;
};

struct B {
  struct A data;
};

struct C {
  struct B *data;
};

struct C var;
```

- ☐ `var.data[3].data.data[1];`
- ☐ `*( var.data->data.data );`
- ☐ `var.data->data.data;`
- ☐ `var.data->data->data;`

Q5 (1.5 pt.)

Claudio ha scritto una funzione che data una stringa restituisce una nuova stringa che si ottiene rimuovendo dalla prima tutto ciò che non è una vocale. Dato il codice seguente, quali affermazioni sono corrette?

```c
char * get_vowels(char *s) {
  char *new_s = (char*) malloc(strlen(s));
  char *dest = new_s;
  while ( *s ) {
    bool is_vowel = *s=='a' || *s=='e' || *s=='i' || *s=='o' || *s=='u'
                 || *s=='A' || *s=='E' || *s=='I' || *s=='O' || *s=='U';
    if ( is_vowel )
      *dest = *s;
    dest++;
    s++;
  }
  *(dest++) = '\0';
  new_s = realloc(new_s, dest-new_s);
  return new_s;
}
```

- ☐ il puntatore `dest` deve essere incrementato solo nel corpo dell'istruzione `if`
- ☐ la `malloc` non alloca memoria sufficiente
- ☐ la `malloc` potrebbe fallire e non restituire un puntatore valido
- ☐ sarebbe più corretto restituire `dest`

Q6 (1.5 pt.)

Claudio ha scritto la seguente funzione ricorsiva che elabora i dati in un array. Quali affermazioni sono corrette?

```c
int process_array (int *v, size_t v_len) {
  if ( v_len == 0 )
    return v[0];

  return v[0] + process_array(v+2, v_len - 2);
}
```

- ☐ il codice calcola la somma dei valori nell'array con indice pari (`v[0]`, `v[2]`, ...)
- ☐ il codice calcola la somma dei valori nell'array
- ☐ il caso base è corretto, ma manca il caso base per `v_len<0`
- ☐ il caso base è corretto, ma manca il caso base per `v_len==1`

Esercizi

Es1 (4 pt.s)

Scrivere una funzione che dati due array di interi aventi la stessa lunghezza diversa da zero, restituisca `true` se ogni elemento del primo array è pari al doppio dell'elemento nella stessa posizione del secondo array, e `false` altrimenti.

La firma della funzione è:

```c
bool is_twice (int *A, size_t A_len, int *B, size_t B_len)
```

Esempio:

```c
int A[] = {2,4,6,8,10};
int B[] = {1,2,3,4,5};
printf("%d \n", is_twice(A, 5, B, 5));

int C[] = {10, 20, 30, 40};
int D[] = { 5, 10, 20, 20};
printf("%d \n", is_twice(C, 4, D, 4));
```

Output:

```
1  /* perché 2 è il doppio di 1, 4 di 2, 6 di 3, 8 di 4, e 10 di 5 */
0  /* perché 30 non è il doppio di 20 */
```

Es2 (5 pt.)

Una matrice è detta *mirrored* se tutte le sue colonne sono palindrome, ovvero hanno la stessa sequenza di elementi sia leggendole dall'alto verso il basso che dal basso verso l'alto. Scrivere una funzione che data una matrice di interi restituisca `true` se questa è *mirrored* e `false` altrimenti.

La firma della funzione è:

```c
bool is_mirrored ( int * M, size_t n_rows, size_t n_cols )
```

Esempio:

```c
int A[4][3] = { { 1, 0, 7 },
                { 2, 0, 2 },
                { 2, 0, 2 },
                { 1, 0, 7 } };
printf("%d \n", is_mirrored( &A[0][0], 4, 3));
```

Output:

```
1  /* perché la prima colonna   1,2,2,1 è palindroma
            la seconda colonna 0,0,0,0 è palindroma
          e la terza colonna   7,2,2,7 è palindroma */
```

Es3 (6 pt.)

Definiamo un algoritmo di decodifica di stringhe come segue:

- una lettera successiva a `'x'`, viene rimossa dalla decodifica assieme alla `'x'`
- una lettera successiva a `'+'`, viene raddoppiata nella decodifica, rimuovendo `'+'`
- altrimenti la lettera va mantenuta nella decodifica

Scrivere una funzione che implementa la decodifica descritta sopra. La firma della funzione è:

```
char* decoder (char *s)
```

Esempio:

```
1 char *encoded = "gxcxaxnxexxa+to";
2 printf("%s \n", decoder( encoded ) );
```

Output:

```
gatto  /* la 'g' viene mantenuta, "xc" "xa" "xn" "xe" e "xx" vengono rimossi,
          la 'a' viene mantenuta, "+t" raddoppiata la 't', e la 'o' mantenuta */
```

Es4 (7 pt.)

Un elemento in una lista di interi si dice *fulcro*, se è pari alla somma di tutti gli elementi precedenti e alla somma di tutti gli elementi successivi. Scrivere una funzione che, data una lista definita come segue, restituisca `true` se la lista ha un fulcro e `false` altrimenti.

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

La firma della funzione è: `bool has_fulcrum(list_t * l)`

Esempio:

```c
/* se la lista l1 è {1,1,1,3,2,1} */
bool res = has_fulcrum(l1);
/* res vale true perché 3 è il fulcro della lista;
                    infatti 1+1+1=3, e 2+1=3 */
```

Es5 (8 pt.)

Data una lista di interi `l`, diciamo che `x` è una *sub-lista* di `l` se tutti i suoi elementi compaiono in `l` nello stesso ordine anche non consecutivi. Ad esempio `{1,2,3}` è una sub-lista di `{0,1,10,2,3}` ma non di `{4,3,2,1,0}`. Scrivere una funzione **ricorsiva** che date due liste `x` e `l` restituisca `true` se `x` è una sub-lista di `l` e `false` altrimenti. (`x` e `l` non hanno duplicati).

La firma della funzione è `bool is_sub_list ( list_t * x, list_t * l )`

Esempio:

```c
/* se la lista l1 è {1,2,3} e
      la lista l2 è {0, 5, 1, 9, 2, 4, 3} */
bool res = is_sub_list (l1, l2);
/* res vale true perchè gli elementi di l1 si trovano
   rispettivamente in 3.a, 5.a, e 7.a posizione di l2 */
```

Soluzioni

Q1

☒ `4 15`

Q2

☒ Non è corretto restituire l'indirizzo della variabile `a`

Q3

☒ la funzione calcola la somma degli elementi nella prima colonna della matrice

Q4

- ☒ var.data->data->data;

Q5

- ☒ il puntatore `dest` deve essere incrementato solo nel corpo dell'istruzione `if`
- ☒ la `malloc` non alloca memoria sufficiente
- ☒ la `malloc` potrebbe fallire e non restituire un puntatore valido

Q6

Nessuna risposta corretta

Es1

```c
bool is_twice (int *A, size_t A_len, int *B, size_t B_len) {
  for (size_t i=0; i<A_len; i++)
    if ( A[i] != 2*B[i] )
      return false;
  return true;
}
```

Es2

```c
bool is_mirrored ( int * M, size_t n_rows, size_t n_cols ) {
  for (size_t i=0; i<n_rows/2; i++) {
    for (size_t j=0; j<n_cols; j++) {
      if ( M[ i*n_cols + j ] != M[ (n_rows - i -1)*n_cols + j ] )
        return false;
    }
  }
  return true;
}
```

Es3

```c
char* decoder (char *s) {
  char *decoded = (char*) malloc(strlen(s)+1);
  if (!decoded) exit(EXIT_FAILURE);
  char *dest = decoded;
  while (*s) {
    if ( *s=='x' ) {
      s += 2;
    } else if ( *s=='+' ) {
      *(dest++) = *(s+1);
      *(dest++) = *(s+1);
      s += 2;
    } else {
      *(dest++) = *(s++);
    }
  }
```

```
  *(dest++) = '\0';
  decoded = realloc(decoded, dest-decoded);
  return decoded;
}
```

Es4

```
bool has_fulcrum(list_t * l) {
  int total = 0;
  list_t *ll = l;
  while(ll) {
    total += ll->data;
    ll = ll->next;
  }

  list_t *fulcrum = l;
  int left_sum = 0;
  while (fulcrum) {
    int right_sum = total - left_sum - fulcrum->data;
    if ( right_sum == left_sum && left_sum == fulcrum->data )
      return true;
    left_sum += fulcrum->data;
    fulcrum = fulcrum->next;
  }
  return false;
}
```

Es5

```
bool is_sub_list ( list_t * x, list_t * l ) {
  if (x==NULL)
    return true;
  if (l==NULL)
    return false;
  if ( x->data == l->data )
    return is_sub_list( x->next, l->next );
  else
    return is_sub_list( x, l->next );
}
```

Appello B del 13/01/2023

Istruzioni

Scrivere subito nome, cognome e matricola sul testo e su tutti i fogli protocollo. Sia il testo che i fogli dovranno essere riconsegnati.

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti é obbligatoria solo per chi non ha superato le esercitazioni. La seconda metà é obbligatoria per tutti. Le domande a risposta multipla possono avere più risposte corrette o nessuna. Usate il foglio di testo per rispondere.

Nella seconda parte lo studente dovrà risolvere e consegnare solo 3 esercizi su 5 in 2 ore. Scrivete le soluzioni nei fogli protocollo.

Il punteggio finale è dato dalla somma dei punteggi.

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 pt.)

Dato il seguente codice:

```
int x=3;
int f1 (int w) {
  if ( w%2 == 0 ) {
    return x * 5;
  } else {
    int x = w * 3;
  }
  return x;
}
```

Qual è l'output del codice seguente?

```
printf( "%d ", f1(x++) );
printf( "%d ", f1(x++) );
```

- ☐ 9 25
- ☐ 9 5
- ☐ 4 25
- ☐ 12 25

Q2 (1.5 pt.)

Dato il seguente codice:

```
int* f2 (int a, int b, int c) {
  int *x = &a;
  int *y = &b;
  int *z = &c;
  *x = *y + *z;
  return &x;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
printf( "%d \n", f2(1,2,3) );
```

- ☐ `*x = *y + *z;` è un'istruzione legale
- ☐ Il valore ritornato è del tipo corretto
- ☐ L'output è 5 pari a `2 + 3`
- ☐ Non si può scrivere `&a`, `&b`, `&c`, perchè gli argomenti `1`, `2`, `3` sono costanti e non hanno un indirizzo come le variabili

Q3 (1.5 pt.)

Dato il codice seguente, quali affermazioni sono corrette?

```
int diag ( int *M, int n_rows, int n_cols ) {
  int sum = 0;
  for (int i=0; i<n_rows; i++)
    for (int j=0; j<n_cols; j++)
        sum += M [i + i*n_cols ];
  return sum;
}
```

- ☐ la funzione calcola la somma degli elementi nella diagonale principale di una matrice quadrata
- ☐ la funzione calcola la somma di tutti gli elementi della matrice
- ☐ per calcolare la somma degli elementi sulla diagonale di una matrice quadrata basta eliminare il secondo for
- ☐ per calcolare la somma degli elementi sulla diagonale di una matrice quadrata non basta eliminare il secondo for

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 pt.)

Dato codice seguente, quali istruzioni *non* compilano correttamente?

```c
struct A {
  int *data;
};

struct B {
  struct A data;
};

struct C {
  struct B *data;
};

struct C var;
```

- ☐ `var->data->data->data;`
- ☐ `var.data[0].data.data;`
- ☐ `var.data[0].data.data[0];`
- ☐ `var.data[0].data[0].data[0];`

Q5 (1.5 pt.)

Claudio ha scritto una funzione che data una stringa restituisce una nuova stringa che si ottiene rimuovendo dalla prima tutto ciò che non è una vocale. Dato il codice seguente, quali affermazioni sono corrette?

```c
char * get_vowels(char *s) {
  char *new_s = (char*) malloc(strlen(s)+1);
  if (!new_s) exit(EXIT_FAILURE);
  while ( *s ) {
    bool is_vowel = *s=='a' || *s=='e' || *s=='i' || *s=='o' || *s=='u'
                 || *s=='A' || *s=='E' || *s=='I' || *s=='O' || *s=='U';
    if ( is_vowel )
      *(new_s++) = *(s++);
  }
  *(new_s++) = '\0';
  return new_s;
}
```

- ☐ il puntatore `s` deve essere incrementato in ogni iterazione del ciclo
- ☐ è sbagliato restituire `new_s`
- ☐ la `malloc` non alloca memoria sufficiente
- ☐ `*(new_s++)= '\0';` non è necessario perchè `\0` viene già copiato all'interno del ciclo while

Q6 (1.5 pt.)

Claudio ha scritto la seguente funzione ricorsiva che elabora i dati in un array che ha almeno un elemento. Quali affermazioni sono corrette?

```c
int process_array (int *v, size_t v_len) {
  if ( v_len == 1 )
    return 0;
```

```
4
5   return v[1] + process_array(v+1, v_len - 1);
6 }
```

- ☐ il codice calcola la somma dei valori nell'array con indice dispari (`v[1]`, `v[3]`, ...)
- ☐ il codice calcola la somma dei valori nell'array tranne il primo
- ☐ il caso base è corretto, ma manca il caso base per `v_len==0`
- ☐ la chiamata ricorsiva dovrebbe essere `process_array(v+1, v_len)`

Esercizi

Es1 (4 pt.s)

Scrivere una funzione che dati due array di interi aventi la stessa lunghezza diversa da zero, restituisca il numero di elementi del primo array che sono pari al doppio dell'elemento nella stessa posizione del secondo array.

La firma della funzione è:

```
int count_twice (int *A, size_t A_len, int *B, size_t B_len)
```

Esempio:

```
1 int A[] = {2,4,6,8,10};
2 int B[] = {1,2,3,4,5};
3 printf("%d \n", count_twice(A, 5, B, 5));
4
5 int C[] = {10, 20, 30, 40};
6 int D[] = { 5, 10, 20, 20};
7 printf("%d \n", count_twice(C, 4, D, 4));
```

Output:

```
5  /* perché 2 è il doppio di 1, 4 di 2, 6 di 3, 8 di 4, e 10 di 5 */
3  /* perché 10 è il doppio di 5, 20 di 10 e 40 di 20 */
```

Es2 (5 pt.)

Una matrice è detta *mirrored* se tutte le sue righe sono palindrome, ovvero hanno la stessa sequenza di elementi sia leggendole da destra a sinistra che da sinistra a destra. Scrivere una funzione che data una matrice di interi restituisca `true` se questa è *mirrored* e `false` altrimenti.

La firma della funzione è:

```
bool is_mirrored ( int * M, size_t n_rows, size_t n_cols )
```

Esempio:

```
1 int A[3][4] = { { 1, 2, 2, 1 },
2                 { 0, 0, 0, 0 },
3                 { 7, 2, 2, 7 } };
4 printf("%d \n", is_mirrored( &A[0][0], 3, 4));
```

Output:

```
1  /* perché la prima riga   1,2,2,1 è palindroma
             la seconda riga 0,0,0,0 è palindroma
           e la terza riga   7,2,2,7 è palindroma */
```

Es3 (6 pt.)

Definiamo un algoritmo di decodifica di stringhe come segue:

- una lettera successiva a `'+'`, viene raddoppiata nella decodifica, rimuovendo `'+'`
- due lettere successive uguali vengono entrambe rimosse
- altrimenti la lettera va mantenuta nella decodifica

Scrivere una funzione che implementa la decodifica descritta sopra. La firma della funzione è:

```
char* decoder (char *s)
```

Esempio:

```
char *encoded = "gccaanneea+to";
printf("%s \n", decoder( encoded ) );
```

Output:

```
gatto  /* la 'g' viene mantenuta, "cc" "aa" "nn" "ee" vengono rimosse,
          la 'a' viene mantenuta, "+t" raddoppiata la 't', e la 'o' mantenuta */
```

Es4 (7 pt.)

Un elemento in una lista di interi si dice *fulcro*, se è pari al prodotto della somma di tutti gli elementi precedenti moltiplicata per la somma di tutti gli elementi successivi. Scrivere una funzione che, data una lista definita come segue, restituisca `true` se la lista ha un fulcro e `false` altrimenti.

```
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

La firma della funzione è: `bool has_fulcrum(list_t * l)`

Esempio:

```
/* se la lista l1 è {1,1,1,9,2,1} */
bool res = has_fulcrum(l1);
/* res vale true perché 9 è il fulcro della lista;
                    infatti 1+1+1=3, e 2+1=3, e 9 = 3*3 */
```

Es5 (8 pt.)

Data una lista di interi `l`, diciamo che `x` è un *sotto-insieme* di `l` se tutti i suoi elementi compaiono in `l` anche in ordine diverso e non consecutivi. Ad esempio `{1,2,3}` è un *sotto-insieme* di `{0,1,10,2,3}` e anche di `{4,3,2,1,0}`. Scrivere una funzione **ricorsiva** che date due liste `x` e `l` restituisca `true` se `x` è un *sotto-insieme* di `l` e `false` altrimenti. (`x` e `l` non hanno duplicati).

La firma della funzione è `bool is_subset ( list_t * x, list_t * l )`

Esempio:

```
/* se la lista l1 è {1,2,3} e
     la lista l2 è {0, 5, 1, 9, 2, 4, 3} */
bool res = is_subset (l1, l2);
/* res vale true */
```

Soluzioni

Q1

☒ `4 25`

Q2

☒ `*x = *y + *z;` è un'istruzione legale

Q3

☒ per calcolare la somma degli elementi sulla diagonale basta eliminare il secondo for

Q4

☒ `var->data->data->data;`
☒ `var.data[0].data[0].data[0];`

Q5

☒ il puntatore `s` deve essere incrementato in ogni iterazione del ciclo
☒ è sbagliato restituire `new_s`

Q6

☒ il codice calcola la somma dei valori nell'array tranne il primo

Es1

```c
int count_twice (int *A, size_t A_len, int *B, size_t B_len) {
  int c = 0;
  for (size_t i=0; i<A_len; i++)
    if ( A[i] == 2*B[i] )
      c++;
  return c;
}
```

Es2

```c
bool is_mirrored ( int * M, size_t n_rows, size_t n_cols ) {
  for (size_t i=0; i<n_rows; i++) {
    for (size_t j=0; j<n_cols/2; j++) {
      if ( M[ i*n_cols + j ] != M[ i*n_cols + (n_cols - j -1) ] )
        return false;
    }
  }
  return true;
}
```

Es3

```c
char* decoder (char *s) {
  char *decoded = (char*) malloc(strlen(s)+1);
  if (!decoded) exit(EXIT_FAILURE);
  char *dest = decoded;
  while (*s) {
    if ( *s==*(s+1) ) {
      s += 2;
    } else if ( *s=='+' ) {
      *(dest++) = *(s+1);
      *(dest++) = *(s+1);
      s += 2;
    } else {
      *(dest++) = *(s++);
    }
  }
  *(dest++) = '\0';
  decoded = realloc(decoded, dest-decoded);
  return decoded;
}
```

Es4

```c
bool has_fulcrum(list_t * l) {
  int total = 0;
  list_t *ll = l;
  while(ll) {
    total += ll->data;
    ll = ll->next;
  }

  list_t *fulcrum = l;
  int left_sum = 0;
  while (fulcrum) {
    int right_sum = total - left_sum - fulcrum->data;
    if ( right_sum * left_sum == fulcrum->data )
      return true;
    left_sum += fulcrum->data;
    fulcrum = fulcrum->next;
  }
  return false;
}
```

Es5

```c
bool is_in (int x, list_t * l) {
  while (l) {
    if ( l->data == x )
      return true;
    l = l->next;
  }
  return false;
}
bool is_subset ( list_t * x, list_t * l ) {
  if (x==NULL)
```

```
11      return true;
12    return is_in( x->data, l) && is_subset( x->next, l);
13  }
```

Appello A del 03/02/2023

Istruzioni

Scrivere subito nome, cognome e matricola sul testo e su tutti i fogli protocollo. Sia il testo che i fogli dovranno essere riconsegnati. Avete a disposizione 2 ore per completare la prova.

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti è obbligatoria solo per chi non ha superato le esercitazioni. La seconda metà è obbligatoria per tutti. Le domande a risposta multipla possono avere più risposte corrette o nessuna. Usate il foglio di testo per rispondere.

Nella seconda parte lo studente dovrà risolvere e consegnare solo 3 esercizi su 5. Scrivete le soluzioni nei fogli protocollo.

Il punteggio finale è dato dalla somma dei punteggi.

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 punti)

Dato il seguente codice:

```
int a=0;
int f1 (int b) {
  int c = b + 1;
  { int d = c + 1; }
  { int a = d + 1; }
  return a;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
printf( "%d ", f1(7) );
```

- ☐ `10`
- ☐ `9`
- ☐ `0`
- ☐ il codice non compila

Q2 (1.5 punti)

Dato il seguente codice:

```c
void f2 (int *a, int *b) {
  int *aux = a;
  a = b;
  b = aux;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```c
int a = 1, b = 2;
f2( &a, &b);
printf( "%d %d \n", a, b);
```

- ☐ `1 2`
- ☐ `2 1` perchè le variabili vengono scambiate
- ☐ l'istruzione `a = b` non compila perché `a` e `b` sono puntatori
- ☐ è più corretto scrivere `int *aux = &a;`

Q3 (1.5 punti)

Dato il seguente codice:

```c
int* copy ( int *M, int n_rows, int n_cols ) {
  int *M_copy = (int*) malloc(n_rows * n_cols * sizeof(int));
  for (int i=0; i<n_rows*n_cols; i++)
    M_copy[i] = M[i];
  return M_copy;
}
```

Quali affermazioni sono corrette?

- ☐ la funzione crea una copia della matrice in input `M`
- ☐ la funzione è sbagliata perché per copiare la matrice `M` è necessario un doppo ciclo `for`
- ☐ la chiamata `malloc` potrebbe fallire restituendo `NULL`
- ☐ manca un `+1` nella chiamata alla `malloc`

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 punti)

Dato codice seguente, quali istruzioni compilano correttamente?

```c
struct A {
  struct A *next;
};

struct A var[10];
```

- ☐ `var[0].next.next;`
- ☐ `var[0].next->next;`
- ☐ `var[0]->next->next;`
- ☐ `var[0].next->next->next;`

Q5 (1.5 punti)

La seguente funzione estende la stringa `dest` concatenando la stringa `src` sapendo che il puntatore `dest` è stato allocato per "contenere" al massimo `dest_size` bytes.

```c
void strconcat ( char * dest, size_t dest_size, char * src ) {
  size_t new_size = strlen(dest) + strlen(src) + 1;
  if ( new_size <= dest_size ) {
    while (*dest != '\0')
      dest++;
    while (*src != '\0')
      *(dest++) = *(src++);
    *(dest) = '\0';
  }
}
```

Quali affermazioni sono corrette?

- ☐ il test corretto è `new_size < dest_size`
- ☐ la nuova dimensione corretta è `size_t new_size = strlen(dest)+strlen(src)+2;`
- ☐ il primo carattere di `src` sovrascrive il carattere `\0` di `dest`
- ☐ il primo carattere di `src` non sovrascrive il carattere `\0` di `dest`

Q6 (1.5 punti)

Claudio ha scritto la seguente funzione ricorsiva che elabora i dati in un array. Quali affermazioni sono corrette?

```c
void process_array (int *v, size_t v_len) {
  if ( v_len > 0 ) {
    process_array(v, v_len - 1);
    printf("%d ", v[v_len -1]);
  }
}
```

- ☐ la funzione stampa gli elementi dell'array
- ☐ la funzione stampa gli elementi dell'array, ma in ordine inverso
- ☐ la funzione non stampa il primo elemento dell'array
- ☐ la funzione non stampa l'ultimo elemento dell'array

Esercizi

Es1 (4 punti)

Scrivere una funzione che dato un array di interi positivi di lunghezza diversa da zero, restituisca `true` se tutti gli elementi dell'array sono numeri primi e `false` altrimenti.

La firma della funzione è:

```c
bool all_prime (int *A, size_t A_len)
```

Esempio:

```c
int A[] = {7, 5, 3, 11};
printf("%d \n", all_prime(A, 4));

int B[] = {1, 13, 2, 6};
printf("%d \n", all_prime(B, 4));
```

Output:

```
1  /* perché 7, 5, 3, 11 sono tutti primi */
0  /* perché 1 e 6 non sono primi */
```

Es2 (5 punti)

Una matrice di interi si dice *perfetta* se la somma degli elementi sulla prima riga è pari a 2^1, la somma degli elementi sulla seconda riga è pari a 2^2 e, in generale, la somma degli elementi sulla riga i-esima è pari a 2^i. Scrivere una funzione che data una matrice di interi restituisca `true` se questa è *perfetta* e `false` altrimenti.

La firma della funzione è:

```c
bool is_perfect ( int * M, size_t n_rows, size_t n_cols )
```

Esempio:

```c
int A[4][3] = { { 0, 1, 1 },
                { 2, 0, 2 },
                { 5, 1, 2 },
                { 6, 7, 3 } };
```

```
5 printf("%d \n", is_perfect( &A[0][0], 4, 3));
```

Output:

```
1  /* perché  0+1+1 = 2,
              2+0+2 = 4,
              5+1+2 = 8,
              6+7+3 = 16  */
```

Es3 (6 punti)

Scrivere una funzione `str_remove` che date due stringhe restituisca una nuova stringa ottenuta rimuovendo le occorrenze della seconda dalla prima.

La firma della funzione è:

```
char * str_remove ( char *a, char *b )
```

Esempio:

```
1 printf("%s \n", str_remove( "i!drop!me!LOVE!drop!me!coding!drop!me!!", "!drop!me!" ) );
```

Output:

```
iLOVEcoding!
```

Es4 (7 punti)

Scrivere una funzione `filter` che date due liste di interi, definite come segue, restituisca una nuova lista ottenuta eliminando gli elementi della prima lista che *non* sono inclusi nella seconda lista.

```
1 typedef struct list {
2   int data;
3   struct list *next; /* next == NULL at the end of the list */
4 } list_t;
```

La firma della funzione è:

```
list_t * filter ( list_t *a, list_t *b )
```

Esempio:

```
1 /* se la lista l1 è {7, 1, 0, 3, 4, 1, 3},
2             e l2 è {1, 3} */
3 list_t *l3 = filter(l1, l2);
4 /*      l3 contiene {1,3,1,3} */
```

Es5 (8 punti)

Scrivere una funzione **ricorsiva** `count_subsets` che dato un array di interi distinti non negativi `A` con almeno un elemento e un intero `n` restituisca il numero di sotto-insiemi di `A` (elementi anche non consecutivi) la cui somma sia strettamente maggiore di `n`.

La firma della funzione è:

```
int count_subsets ( int *A, int A_size, int n )
```

Esempio:

```
int v[] = { 1, 2, 3, 0, 6, 7};
printf("%d \n", count_subsets(v, 6, 15) );
```

Output:

```
10 /* le combinazioni possibili sono:
   (1, 2, 6, 7),  (1, 3, 6, 7), (1, 2, 3, 6, 7),
   (1, 2, 0, 6, 7), (1, 3, 0, 6, 7), (1, 2, 3, 0, 6, 7),
   (2, 3, 6, 7), (2, 3, 0, 6, 7)
   (3, 6, 7), (3, 0, 6, 7) */
```

Soluzioni

Q1

- ☒ il codice non compila

Q2

- ☒ `1 2`

Q3

- ☒ la funzione crea una copia della matrice in input `M`
- ☒ la chiamata `malloc` potrebbe fallire restituendo `NULL`

Q4

- ☒ `var[0].next->next;`
- ☒ `var[0].next->next->next;`

Q5

- ☒ il primo carattere di `src` sovrascrive il carattere `\0` di `dest`

Q6

- ☒ la funzione stampa gli elementi dell'array

Es1

```
bool is_prime(int x) {   // assume x>0
  if ( x==1 )
    return false;

  for (int i=2; i<x; i++) // può essere ottimizzato
```

```
    if ( x%i==0 )
      return false;
  return true;
}

bool all_prime (int *A, size_t A_len) {  // assume A_len>0
  for (size_t i=0; i<A_len; i++)
    if ( ! is_prime(A[i]) )
      return false;
  return true;
}
```

Es2

```
int pow_2 (int x) {   // assume x>=0
  int p = 1;
  for (int i=0; i<x; i++)
    p*=2;
  return p;
}

bool is_perfect ( int * M, size_t n_rows, size_t n_cols ) {
  for (size_t row = 0; row<n_rows; row++){
    int row_sum = 0;
    for (size_t col = 0; col<n_cols; col++)
      row_sum += M[ row*n_cols + col ];
    if ( row_sum != pow_2(row+1) )
      return false;
  }
  return true;
}
```

Es3

```
bool is_match ( char *a, char *b ) {
  while ( *b ) {
    if (*a!=*b)
      return false;
    a++;
    b++;
  }
  return true;
}

char * str_remove ( char *a, char *b ) {
  char * new_s = (char*) malloc ( strlen(a) +1 );
  if (!new_s) exit(EXIT_FAILURE);
  char * dest = new_s;
  while (*a) {
    if ( is_match(a,b) ) {
      a += strlen(b);
    } else {
      *(dest++) = *(a++);
    }
  }
  *(dest++) = '\0';
  new_s = realloc(new_s, dest-new_s);
```

```
  return new_s;
}
```

Es4

```
bool is_in ( int x, list_t * l ) {
  while ( l ) {
    if ( l->data == x)
      return true;
    l = l->next;
  }
  return false;
}

list_t *  filter ( list_t *a, list_t *b ) {
  list_t * new_l = NULL;
  list_t * tail = new_l;
  while (a) {
    if ( is_in(a->data, b) ) {
      list_t * new_node = (list_t*) malloc(sizeof(list_t));
      if (!new_node) exit(EXIT_FAILURE);
      new_node->data = a->data;
      new_node->next = NULL;
      if (new_l==NULL)
        new_l = tail = new_node;
      else {
        tail->next = new_node;
        tail = new_node;
      }
    }
    a = a->next;
  }

  return new_l;
}
```

Es5

```
int count_subsets ( int *A, int A_size, int n ) {
  if (n<0)       // caso base: superato il target n
    return pow(2, A_size);    // tutti i sotto-insiemi possibili
  if (A_size==0) // caso base: non ci sono altri elementi in A
    return 0;
  return count_subsets( A+1, A_size-1, n-A[0] )  // sotto-insiemi che includono A[0]
       + count_subsets( A+1, A_size-1, n );      // sotto-insiemi che non includono A[0]
}
```

Appello B del 03/02/2023

Istruzioni

Scrivere subito nome, cognome e matricola sul testo e su tutti i fogli protocollo. Sia il testo che i fogli dovranno essere riconsegnati. Avete a disposizione 2 ore per completare la prova.

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti è obbligatoria solo per chi non ha superato le esercitazioni. La seconda metà è obbligatoria per tutti. Le domande a risposta multipla possono avere più risposte corrette o nessuna. Usate il foglio di testo per rispondere.

Nella seconda parte lo studente dovrà risolvere e consegnare solo 3 esercizi su 5. Scrivete le soluzioni nei fogli protocollo.

Il punteggio finale è dato dalla somma dei punteggi.

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 punti)

Dato il seguente codice:

```c
int a=9;
int f1 (int b) {
  int c = b + 1;
  { int d = c + 1;
    { int a = d + 1; }
  }
  return a;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```c
printf( "%d ", f1(5) );
```

- ☐ 8
- ☐ 9
- ☐ 5
- ☐ il codice non compila

Q2 (1.5 punti)

Dato il seguente codice:

```
void f2 (int *a, int *b) {
  a = b;
  b = a;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```
int a = 1, b = 2;
f2( &a, &b);
printf( "%d %d \n", a, b);
```

- ☐ `1 2`
- ☐ `2 1` perchè le variabili vengono scambiate
- ☐ lo scambio sarebbe corretto con `aux=a; a=b, b=aux;`
- ☐ `2 2`

Q3 (1.5 punti)

Dato il seguente codice:

```
int* copy ( int *M, int n_rows, int n_cols ) {
  int *M_copy = (int*) malloc(n_rows * n_cols);
  for (int i=0; i<n_rows*n_cols; i++)
    M_copy[i] = M[i];
  return M_copy;
}
```

Quali affermazioni sono corrette?

- ☐ la funzione crea una copia della matrice in input `M`
- ☐ la funzione copia solo la prima riga della matrice in input `M`
- ☐ la funzione è sbagliata perché per copiare la matrice `M` è necessario un doppo ciclo **`for`**
- ☐ non viene allocata sufficiente memoria per copiare la matrice `M`

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 punti)

Dato codice seguente, quali istruzioni compilano correttamente?

```c
struct A {
  struct A *next;
};

struct A *var;
```

- ☐ var.next;
- ☐ (*var).next;
- ☐ ((var).next).next;
- ☐ var->next->next->next;

Q5 (1.5 punti)

La seguente funzione estende la stringa `dest` concatenando la stringa `src` sapendo che il puntatore `dest` è stato allocato per "contenere" al massimo `dest_size` bytes.

```c
void strconcat ( char * dest, size_t dest_size, char * src ) {
  size_t new_size = strlen(dest) + strlen(src) + 1;
  if ( new_size <= dest_size ) {
    while (*dest != '\0')
      dest++;
    while (*src != '\0')
      *(dest++) = *(src++);
    *(dest) = '\0';
  }
}
```

Quali affermazioni sono corrette?

- ☐ la nuova dimensione corretta è `size_t new_size = strlen(dest)+strlen(src);`
- ☐ il chiamante non riesce ad accedere alla stringa perchè la funzione modifica `dest`
- ☐ sarebbe più corretto scrivere `while (dest != NULL)`
- ☐ il primo carattere di `src` sovrascrive il carattere `\0` di `dest`

Q6 (1.5 punti)

Claudio ha scritto la seguente funzione ricorsiva che elabora i dati in un array. Quali affermazioni sono corrette?

```c
void process_array (int *v, size_t v_len) {
  if ( v_len > 0 ) {
    process_array(v+1, v_len - 1);
    printf("%d ", v[0]);
  }
}
```

- ☐ la funzione stampa gli elementi dell'array
- ☐ la funzione stampa gli elementi dell'array, ma in ordine inverso
- ☐ la funzione non stampa il primo elemento dell'array
- ☐ la funzione non stampa l'ultimo elemento dell'array

Esercizi

Es1 (4 punti)

Scrivere una funzione che dato un array di interi positivi di lunghezza diversa da zero, restituisca `true` se nessuno gli elementi dell'array è primo e `false` altrimenti.

La firma della funzione è:

```c
bool no_prime (int *A, size_t A_len)
```

Esempio:

```c
int A[] = {10, 4, 8, 11};
printf("%d \n", no_prime(A, 4));

int B[] = {1, 4, 8, 6};
printf("%d \n", no_prime(B, 4));
```

Output:

```
0  /* perché 11 è primo */
1  /* perché nessuno tra 1, 4, 8, 6 è primo */
```

Es2 (5 punti)

Una matrice di interi si dice *perfetta* se la somma degli elementi sulla colonna riga è pari a 2^1, la somma degli elementi sulla seconda colonna è pari a 2^2 e, in generale, la somma degli elementi sulla colonna i-esima è pari a 2^i. Scrivere una funzione che data una matrice di interi restituisca `true` se questa è *perfetta* e `false` altrimenti.

La firma della funzione è:

```c
bool is_perfect ( int * M, size_t n_rows, size_t n_cols )
```

Esempio:

```c
int A[4][3] = { { 0, 1, 1 },
                { 0, 2, 2 },
                { 1, 1, 2 },
                { 1, 0, 3 } };
```

```
5 printf("%d \n", is_perfect( &A[0][0], 4, 3));
```

Output:

```
1  /* perché  0+0+1+1 = 2,
              1+2+1+0 = 4,
              1+2+2+3 = 8 */
```

Es3 (6 punti)

Scrivere una funzione `str_dup` che date due stringhe restituisca una nuova stringa ottenuta raddoppiando le occorrenze della seconda nella prima.

La firma della funzione è:

```
char * str_dup ( char *a, char *b )
```

Esempio:

```
1 printf("%s \n", str_dup( "i love coding and i love playing", "love" ) );
```

Output:

```
i lovelove coding and i lovelove playing
```

Es4 (7 punti)

Scrivere una funzione `filter` che date due liste di interi, definite come segue, restituisca una nuova lista ottenuta eliminando gli elementi della prima lista che sono inclusi nella seconda lista.

```
1 typedef struct list {
2   int data;
3   struct list *next; /* next == NULL at the end of the list */
4 } list_t;
```

La firma della funzione è:

```
list_t * filter ( list_t *a, list_t *b )
```

Esempio:

```
1 /* se la lista l1 è {7, 1, 0, 3, 4, 1, 3},
2              e l2 è {1, 3} */
3 list_t *l3 = filter(l1, l2);
4 /*      l3 contiene {7, 0, 4} */
```

Es5 (8 punti)

Scrivere una funzione **ricorsiva** `count_subsets` che dato un array di interi distinti non negativi `A` con almeno un elemento e un intero `n` restituisca il numero di sotto-insiemi di `A` (elementi anche non consecutivi) la cui somma sia pari a `n`.

La firma della funzione è:

```
int count_subsets ( int *A, int A_size, int n )
```

Esempio:

```c
int v[] = { 1, 2, 3, 0, 6, 7};
printf("%d \n", count_subsets(v, 6, 6) );
```

Output:

```
4  /* le combinazioni possibili sono:
   (6),  (6, 0),
   (1, 2, 3), (1, 2, 3, 0) */
```

Soluzioni

Q1

☒ 9

Q2

☒ 1 2

Q3

☒ non viene allocata sufficiente memoria per copiare la matrice `M`

Q4

☒ `(*var).next;`
☒ `var->next->next->next;`

Q5

☒ il primo carattere di `src` sovrascrive il carattere `\0` di `dest`

Q6

☒ la funzione stampa gli elementi dell'array, ma in ordine inverso

Es1

```c
bool is_prime(int x) {   // assume x>0
  if ( x==1 )
    return false;

  for (int i=2; i<x; i++) // può essere ottimizzato
    if ( x%i==0 )
      return false;
  return true;
```

```
}

bool no_prime (int *A, size_t A_len) {
  for (size_t i=0; i<A_len; i++)
    if ( is_prime(A[i]) )
      return false;
  return true;
}
```

Es2

```
int pow_2 (int x) {   // assume x>=0
  int p = 1;
  for (int i=0; i<x; i++)
    p*=2;
  return p;
}

bool is_perfect ( int * M, size_t n_rows, size_t n_cols ) {
  for (size_t col = 0; col<n_cols; col++) {
    int col_sum = 0;
    for (size_t row = 0; row<n_rows; row++)
      col_sum += M[ row*n_cols + col ];
    if ( col_sum != pow_2(col+1) )
      return false;
  }
  return true;
}
```

Es3

```
bool is_match ( char *a, char *b ) {
  while ( *b ) {
    if (*a!=*b)
      return false;
    a++;
    b++;
  }
  return true;
}

char * str_dup ( char *a, char *b ) {
  size_t b_len = strlen(b);
  char * new_s = (char*) malloc ( 2*strlen(a) +1 );
  if (!new_s) exit(EXIT_FAILURE);
  char * dest = new_s;
  while (*a) {
    if ( is_match(a,b) ) {
      char *b_copy = b;
      while (*b_copy) {
        *(dest) = *(b_copy);
        *(dest+b_len) = *(b_copy);
        dest++;
        b_copy++;
      }
      dest += b_len;
      a += b_len;
```

```
    } else {
      *(dest++) = *(a++);
    }
  }
  *(dest++) = '\0';
  new_s = realloc(new_s, dest-new_s);
  return new_s;
}
```

Es4

```
bool is_in ( int x, list_t * l ) {
  while ( l ) {
    if ( l->data == x)
      return true;
    l = l->next;
  }
  return false;
}

list_t *  filter ( list_t *a, list_t *b ) {
  list_t * new_l = NULL;
  list_t * tail = new_l;
  while (a) {
    if ( ! is_in(a->data, b) ) {
      list_t * new_node = (list_t*) malloc(sizeof(list_t));
      if (!new_node) exit(EXIT_FAILURE);
      new_node->data = a->data;
      new_node->next = NULL;
      if (new_l==NULL)
        new_l = tail = new_node;
      else {
        tail->next = new_node;
        tail = new_node;
      }
    }
    a = a->next;
  }

  return new_l;
}
```

Es5

```
int count_subsets ( int *A, int A_size, int n ) {
  if (A_size==0) // caso base: non ci sono altri elementi in A
    return 0;
  return (A[0]==n) + // con A[0] raggiungo il target n
      count_subsets( A+1, A_size-1, n-A[0] ) + // sotto-insiemi che includono A[0]
          count_subsets( A+1, A_size-1, n );      // sotto-insiemi che non includono A[0]
}
```

Appello del 16/06/2023

Scrivere subito nome, cognome e matricola sul testo e su tutti i fogli protocollo. Sia il testo che i fogli dovranno essere riconsegnati. Avete a disposizione 2 ore per completare la prova.

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti è obbligatoria solo per chi non ha superato le esercitazioni. La seconda metà è obbligatoria per tutti. Le domande a risposta multipla possono avere più risposte corrette o nessuna. Usate il foglio di testo per rispondere.

Nella seconda parte lo studente dovrà risolvere e consegnare solo 3 esercizi su 5. Scrivete le soluzioni nei fogli protocollo.

Il punteggio finale è dato dalla somma dei punteggi.

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 punti)

Dato il seguente codice:

```c
int sum = 0;
int f1(int n) {
  for ( int i=0; i<n; i++ ) {
    sum += n;
  }
  return sum;
}
```

Qual è l'output del codice seguente?

```c
printf( "%d %d\n", f1(2), f1(3) );
```

- ☐ 3 9
- ☐ 1 3
- ☐ 1 4
- ☐ 4 13

Q2 (1.5 punti)

Dato il seguente codice:

```c
void f2(int *a, int *b, int *mult) {
  *mult = 0;
  for ( int i=0; i< *b; i++ )
    *mult += *a;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```c
int a = 10;
int b = 3;
int c = -30;
f2(&a, &b, &c);
printf("%d \n", c );
```

- ☐ `-30`
- ☐ `0`
- ☐ `30`
- ☐ è più corretto scrivere `mult = 0;`

Q3 (1.5 pt.)

Dato il seguente codice, quali delle seguenti affermazioni sono vere?

```c
int f3(int M[3][3]) {
  int sum=0;
  for (int i=0; i<3; i++)
    for (int j=0; j!=i; j++)
      sum += M[i][j];
  return sum;
}
```

- ☐ la funzione calcola la somma di tutti gli elementi della matrice eslcusa la diagonale principale
- ☐ la funzione calcola la somma degli elementi sotto la diagonale della matrice (diagonale esclusa)
- ☐ `sum += M[i][j];` viene eseguito 9 volte
- ☐ `sum += M[i][j];` viene eseguito 3 volte

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 punti)

Claudio ha scritto una funzione per verificare se due stringhe siano uguali. Quali affermazioni sono corrette?

```c
bool string_equal(char *a, char *b) {
  for (int i=0; i<strlen(a); i++) {
    if ( a[i]!=b[i] )
      return false;
    else
      return true;
  }
  return true;
}
```

- ☐ la funzione è corretta solo se le stringhe hanno la stessa lunghezza
- ☐ la funzione non ritorna mai `true`
- ☐ è sufficiente rimuovere il ramo **`else`**
- ☐ il ramo **`else`** va rimosso, ma questo non è sufficiente a correggere la funzione

Q5 (1.5 punti)

Data una matrice `m` rappresentata tramite un array di array costruito come segue, quali affermazioni sono corrette?

```c
int **m = (int**) malloc(rows * sizeof(int*));
for (int i=0; i<rows, i++)
    m[i] = (int*) malloc(cols * sizeof(int));
```

- ☐ `*(m[i])` accede all'elemento nella `i`-esima riga e prima colonna
- ☐ `*(m[i] + j)` accede all'elemento nella `i`-esima riga e `j`-esima colonna
- ☐ `m[i][j]` accede all'elemento nella `i`-esima riga e `j`-esima colonna
- ☐ `m[i*cols + j]` accede all'elemento nella `i`-esima riga e `j`-esima colonna

Q6 (1.5 punti)

Claudio ha scritto una funzione per individuare se una stringa appare come sottostringa di un altra (es. "assi" è sottostringa di "scassinatore"). Quali delle seguenti affermazioni sono corrette?

```c
bool f6 (char *a, char *b) {
  for (int i=0; i<strlen(b); i++) {
    bool is_substring = true;
    for (int j=0; j<strlen(a) && (i+j)<strlen(b); j++)
      is_substring = a[j]==b[i+j];
    if (is_substring)
      return true;
  }
  return false;
}
```

- ☐ la funzione non è corretta perché potrebbe accedere oltre la fine della stringa `a`
- ☐ la funzione non è corretta perché potrebbe accedere oltre la fine della stringa `b`
- ☐ la funzione restituisce sempre `true`
- ☐ la funzione non è corretta perché non confronta correttamente i caratteri delle due stringhe

Esercizi

Es1 (4 punti)

Scrivere una funzione `delta` che, dato un array di interi `A` di lunghezza maggiore di 1, restituisca la differenza tra i suoi due elementi più piccoli.

Esempio:

```c
int A[] = { 1,5,8,3 };
printf( "%d \n", delta(A, 4));
```

Output:

```
2  /* perché 3-1 = 2 */
```

La firma della funzione è : `int delta( int *A, int A_size )`.

Es2 (5 punti)

Una matrice quadrata è detta bilanciata se la somma degli elementi al di sopra della diagonale principale è pari alla somma degli elementi al di sotto della diagonale principale. Scrivere una funzione che data una matrice quadrata di interi `A` restituisca `true` se questa è bilanciata e `false` altrimenti.

Esempio:

```c
int M[3][3] = { { 1,10,20},
                {20, 2,30},
                {20,20, 3} };
printf("%d\n", tri_bil( &(M[0][0]), 3) );
```

Output:

```
1 /* 10+20+30 == 20+20+20 */
```

La firma della funzione è : `bool tri_bil( int *M, int n_rows )`.

Es3 (6 punti)

Scrivere una funzione che data una stringa restituisca una nuova stringa contenente solo le sotto-stringhe racchiuse tra parentesi tonde. Assumiamo che le parentesi siano corrette e che non siano annidate.

Esempio:

```c
char * a = "(I ) do not (Love ) wasting my time at (Coding) badly";
printf("%s\n", parentesi( a ) );
```

Output:

```
I Love Coding
```

La firma della funzione è : `char * parentesi( char *s )`.

Es4 (7 punti)

Scrivere una funzione che data una lista di interi restituisca una nuova lista (fatta di nuovi nodi) con soli gli elementi che appaiano almeno due volte.

La struttura dati lista è definita come segue:

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```c
/* se l1 = {1,1,2,3,1,2,4} */
list_t * l2 = dups(l1)
/* l2 = {1,2} */
```

La firma della funzione è : `list_t * dups(list_t *l)`.

Es5 (8 punti)

Scrivere una funzione **ricorsiva** che dati due array di interi non vuoti conteggi il numero di volte che il primo array appare nel secondo *senza* sovrapposizioni.

Esempio:

```c
int v1[] = {7, 3, 7};
int v2[] = {7, 3, 7, 3, 7, 7, 3, 7, 0};
printf("%d \n", count_array(v1, 3, v2, 9) );
```

Output:

```
2 /* a partire dalle posizioni 0 e 5 di v2 */
```

La firma della funzione è :

`int count_array( int *a, int a_size, int *b, int b_size ).`

Soluzioni

Q1

☒ `4 13`

Q2

☒ `30`

Q3

☒ la funzione calcola la somma degli elementi sotto la diagonale della matrice (diagonale esclusa)
☒ `sum += M[i][j];` viene eseguito 3 volte

Q4

☒ il ramo **`else`** va rimosso, ma questo non è sufficiente a correggere la funzione

Q5

☒ `*(m[i])` accede all'elemento nella `i`-esima riga e prima colonna
☒ `*(m[i] + j)` accede all'elemento nella `i`-esima riga e `j`-esima colonna
☒ `m[i][j]` accede all'elemento nella `i`-esima riga e `j`-esima colonna

Q6

☒ la funzione non è corretta perché non confronta correttamente i caratteri delle due stringhe

Es1

```
1 int delta( int *A, int A_size ) {
2   int min_0, min_1;
3   if (A[0]<A[1]) {
4     min_0 = A[0]; min_1 = A[1];
5   } else {
6     min_0 = A[1]; min_1 = A[0];
7   }
8   for ( int i=2; i<A_size; i++) {
```

```
    if ( A[i]<min_0 ) {
      min_1 = min_0;
      min_0 = A[i];
    }
    else if ( A[i]<min_1 )
      min_1 = A[i];
  }

  return min_1-min_0;
}
```

Es2

```
int tri_bil( int *M, int n_rows ) {
  int sum_up = 0, sum_lo = 0;
  for ( int i=0; i<n_rows; i++ ) {
    for ( int j=0; j<n_rows; j++ ) {
      if ( i>j )
        sum_lo += M[i*n_rows + j];
      else if ( i<j )
        sum_up += M[i*n_rows + j];
    }
  }
  return sum_lo==sum_up;
}
```

Es3

```
char * parentesi( char *s ) {
  bool keep = false;
  char * c = (char*) malloc ( strlen(s) + 1 );
  if (!c) exit(EXIT_FAILURE);
  char * dest = c;
  while (*s) {
    if ( *s=='(' )
      keep = true;
    else if ( *s==')' )
      keep = false;
    else if (keep)
      *(dest++) = *s;
    s++;
  }
  *(dest++) = '\0';
  c = realloc (c, dest-c);
  return c;
}
```

Es4

```
bool is_in(int x, list_t *l) {
  while (l) {
    if (l->data==x)
      return true;
    l = l->next;
  }
```

```
  return false;
}

list_t * dups(list_t *l) {
  list_t *dups = NULL;
  list_t *dups_tail = NULL;

  while ( l!=NULL ) {
    if ( !is_in(l->data, dups) &&
        is_in(l->data, l->next) ) {
      list_t *new_node = (list_t*) malloc(sizeof(list_t));
      if (!new_node) exit(EXIT_FAILURE);
      new_node->data = l->data;
      new_node->next = NULL;

      if ( dups==NULL ) {
        dups = dups_tail = new_node;
      } else {
        dups_tail->next = new_node;
        dups_tail = dups_tail->next;
      }
    }
    l = l->next;
  }

  return dups;
}
```

Es5

```
bool is_prefixarray( int *a, int a_size, int *b, int b_size ) {
  if (a_size>b_size)
    return false;
  for (int i=0; i<a_size; i++ )
    if (a[i]!=b[i])
      return false;
  return true;
}

int count_array( int *a, int a_size, int *b, int b_size ) {
  if (b_size<=0)
    return 0;
  if (is_prefixarray(a, a_size, b, b_size))
    return 1 + count_array(a, a_size, b+a_size, b_size-a_size);
  else
    return count_array(a, a_size, b+1, b_size -1);
}
```

Appello del 11/09/2023

Scrivere subito nome, cognome e matricola sul testo e su tutti i fogli protocollo. Sia il testo che i fogli dovranno essere riconsegnati. Avete a disposizione 2 ore per completare la prova.

Il testo prevede 2 parti. La prima parte include quesiti a risposta multipla. La prima metà dei quesiti è obbligatoria solo per chi non ha superato le esercitazioni. La seconda metà è obbligatoria per tutti. Le domande a risposta multipla possono avere più risposte corrette o nessuna. Usate il foglio di testo per rispondere.

Nella seconda parte lo studente dovrà risolvere e consegnare solo 3 esercizi su 5. Scrivete le soluzioni nei fogli protocollo.

Il punteggio finale è dato dalla somma dei punteggi.

Domande a Risposta Multipla - Per chi non ha superato le esercitazioni

Q1 (1.5 punti)

Dato il seguente codice:

```c
int sum = 0;
int f1(int n) {
  for ( int i=0; i<n; i++ ) {
    sum += n;
  }
  return sum;
}
```

Qual è l'output del codice seguente?

```c
printf( "%d %d\n", f1(2), f1(3) );
```

- ☐ 3 9
- ☐ 1 3
- ☐ 1 4
- ☐ 4 13

Q2 (1.5 punti)

Dato il seguente codice:

```c
void f2(int *a, int *b, int *mult) {
  *mult = 0;
  for ( int i=0; i< *b; i++ )
    *mult += *a;
}
```

Qual è l'output del codice seguente e quali affermazioni sono corrette?

```c
int a = 10;
int b = 3;
int c = -30;
f2(&a, &b, &c);
printf("%d \n", c );
```

- ☐ `-30`
- ☐ `0`
- ☐ `30`
- ☐ è più corretto scrivere `mult = 0;`

Q3 (1.5 pt.)

Dato il seguente codice, quali delle seguenti affermazioni sono vere?

```c
int f3(int M[3][3]) {
  int sum=0;
  for (int i=0; i<3; i++)
    for (int j=0; j!=i; j++)
      sum += M[i][j];
  return sum;
}
```

- ☐ la funzione calcola la somma di tutti gli elementi della matrice eslcusa la diagonale principale
- ☐ la funzione calcola la somma degli elementi sotto la diagonale della matrice (diagonale esclusa)
- ☐ `sum += M[i][j];` viene eseguito 9 volte
- ☐ `sum += M[i][j];` viene eseguito 3 volte

Domande a Risposta Multipla - Obbligatorie per tutti

Q4 (1.5 punti)

Claudio ha scritto una funzione per verificare se due stringhe siano uguali. Quali affermazioni sono corrette?

```c
bool string_equal(char *a, char *b) {
  for (int i=0; i<strlen(a); i++) {
    if ( a[i]!=b[i] )
      return false;
    else
      return true;
  }
  return true;
}
```

- ☐ la funzione è corretta solo se le stringhe hanno la stessa lunghezza
- ☐ la funzione non ritorna mai `true`
- ☐ è sufficiente rimuovere il ramo **`else`**
- ☐ il ramo **`else`** va rimosso, ma questo non è sufficiente a correggere la funzione

Q5 (1.5 punti)

Data una matrice `m` rappresentata tramite un array di array costruito come segue, quali affermazioni sono corrette?

```c
int **m = (int**) malloc(rows * sizeof(int*));
for (int i=0; i<rows, i++)
    m[i] = (int*) malloc(cols * sizeof(int));
```

- ☐ `*(m[i])` accede all'elemento nella `i`-esima riga e prima colonna
- ☐ `*(m[i] + j)` accede all'elemento nella `i`-esima riga e `j`-esima colonna
- ☐ `m[i][j]` accede all'elemento nella `i`-esima riga e `j`-esima colonna
- ☐ `m[i*cols + j]` accede all'elemento nella `i`-esima riga e `j`-esima colonna

Q6 (1.5 punti)

Claudio ha scritto una funzione per individuare se una stringa appare come sottostringa di un altra (es. "assi" è sottostringa di "scassinatore"). Quali delle seguenti affermazioni sono corrette?

```c
bool f6 (char *a, char *b) {
  for (int i=0; i<strlen(b); i++) {
    bool is_substring = true;
    for (int j=0; j<strlen(a) && (i+j)<strlen(b); j++)
      is_substring = a[j]==b[i+j];
    if (is_substring)
      return true;
  }
  return false;
}
```

- ☐ la funzione non è corretta perché potrebbe accedere oltre la fine della stringa `a`
- ☐ la funzione non è corretta perché potrebbe accedere oltre la fine della stringa `b`
- ☐ la funzione restituisce sempre `true`
- ☐ la funzione non è corretta perché non confronta correttamente i caratteri delle due stringhe

Esercizi

Es1 (4 punti)

Scrivere una funzione `delta` che, dato un array di interi `A` di lunghezza maggiore di 1, restituisca la differenza tra i suoi due elementi più piccoli.

Esempio:

```c
int A[] = { 1,5,8,3 };
printf( "%d \n", delta(A, 4));
```

Output:

```
2  /* perché 3-1 = 2 */
```

La firma della funzione è : `int delta( int *A, int A_size )`.

Es2 (5 punti)

Una matrice quadrata è detta bilanciata se la somma degli elementi al di sopra della diagonale principale è pari alla somma degli elementi al di sotto della diagonale principale. Scrivere una funzione che data una matrice quadrata di interi `A` restituisca `true` se questa è bilanciata e `false` altrimenti.

Esempio:

```c
int M[3][3] = { { 1,10,20},
                {20, 2,30},
                {20,20, 3} };
printf("%d\n", tri_bil( &(M[0][0]), 3) );
```

Output:

```
1 /* 10+20+30 == 20+20+20 */
```

La firma della funzione è : `bool tri_bil( int *M, int n_rows )`.

Es3 (6 punti)

Scrivere una funzione che data una stringa restituisca una nuova stringa contenente solo le sotto-stringhe racchiuse tra parentesi tonde. Assumiamo che le parentesi siano corrette e che non siano annidate.

Esempio:

```c
char * a = "(I ) do not (Love ) wasting my time at (Coding) badly";
printf("%s\n", parentesi( a ) );
```

Output:

```
I Love Coding
```

La firma della funzione è : `char * parentesi( char *s )`.

Es4 (7 punti)

Scrivere una funzione che data una lista di interi restituisca una nuova lista (fatta di nuovi nodi) con soli gli elementi che appaiano almeno due volte.

La struttura dati lista è definita come segue:

```c
typedef struct list {
  int data;
  struct list *next; /* next == NULL at the end of the list */
} list_t;
```

Esempio:

```c
/* se l1 = {1,1,2,3,1,2,4} */
list_t * l2 = dups(l1)
/* l2 = {1,2} */
```

La firma della funzione è : `list_t * dups(list_t *l)`.

Es5 (8 punti)

Scrivere una funzione **ricorsiva** che dati due array di interi non vuoti conteggi il numero di volte che il primo array appare nel secondo *senza* sovrapposizioni.

Esempio:

```c
int v1[] = {7, 3, 7};
int v2[] = {7, 3, 7, 3, 7, 7, 3, 7, 0};
printf("%d \n", count_array(v1, 3, v2, 9) );
```

Output:

```
2 /* a partire dalle posizioni 0 e 5 di v2 */
```

La firma della funzione è :

`int count_array( int *a, int a_size, int *b, int b_size ).`

Soluzioni

Q1

☒ `4 13`

Q2

☒ `30`

Q3

☒ la funzione calcola la somma degli elementi sotto la diagonale della matrice (diagonale esclusa)
☒ `sum += M[i][j];` viene eseguito 3 volte

Q4

☒ il ramo **`else`** va rimosso, ma questo non è sufficiente a correggere la funzione

Q5

☒ `*(m[i])` accede all'elemento nella `i`-esima riga e prima colonna
☒ `*(m[i] + j)` accede all'elemento nella `i`-esima riga e `j`-esima colonna
☒ `m[i][j]` accede all'elemento nella `i`-esima riga e `j`-esima colonna

Q6

☒ la funzione non è corretta perché non confronta correttamente i caratteri delle due stringhe

Es1

```
1 int delta( int *A, int A_size ) {
2   int min_0, min_1;
3   if (A[0]<A[1]) {
4     min_0 = A[0]; min_1 = A[1];
5   } else {
6     min_0 = A[1]; min_1 = A[0];
7   }
8   for ( int i=2; i<A_size; i++) {
```

```
    if ( A[i]<min_0 ) {
      min_1 = min_0;
      min_0 = A[i];
    }
    else if ( A[i]<min_1 )
      min_1 = A[i];
  }

  return min_1-min_0;
}
```

Es2

```
int tri_bil( int *M, int n_rows ) {
  int sum_up = 0, sum_lo = 0;
  for ( int i=0; i<n_rows; i++ ) {
    for ( int j=0; j<n_rows; j++ ) {
      if ( i>j )
        sum_lo += M[i*n_rows + j];
      else if ( i<j )
        sum_up += M[i*n_rows + j];
    }
  }
  return sum_lo==sum_up;
}
```

Es3

```
char * parentesi( char *s ) {
  bool keep = false;
  char * c = (char*) malloc ( strlen(s) + 1 );
  if (!c) exit(EXIT_FAILURE);
  char * dest = c;
  while (*s) {
    if ( *s=='(' )
      keep = true;
    else if ( *s==')' )
      keep = false;
    else if (keep)
      *(dest++) = *s;
    s++;
  }
  *(dest++) = '\0';
  c = realloc (c, dest-c);
  return c;
}
```

Es4

```
bool is_in(int x, list_t *l) {
  while (l) {
    if (l->data==x)
      return true;
    l = l->next;
  }
```

```
  return false;
}

list_t * dups(list_t *l) {
  list_t *dups = NULL;
  list_t *dups_tail = NULL;

  while ( l!=NULL ) {
    if ( !is_in(l->data, dups) &&
        is_in(l->data, l->next) ) {
      list_t *new_node = (list_t*) malloc(sizeof(list_t));
      if (!new_node) exit(EXIT_FAILURE);
      new_node->data = l->data;
      new_node->next = NULL;

      if ( dups==NULL ) {
        dups = dups_tail = new_node;
      } else {
        dups_tail->next = new_node;
        dups_tail = dups_tail->next;
      }
    }
    l = l->next;
  }

  return dups;
}
```

Es5

```
bool is_prefixarray( int *a, int a_size, int *b, int b_size ) {
  if (a_size>b_size)
    return false;
  for (int i=0; i<a_size; i++ )
    if (a[i]!=b[i])
      return false;
  return true;
}

int count_array( int *a, int a_size, int *b, int b_size ) {
  if (b_size<=0)
    return 0;
  if (is_prefixarray(a, a_size, b, b_size))
    return 1 + count_array(a, a_size, b+a_size, b_size-a_size);
  else
    return count_array(a, a_size, b+1, b_size -1);
}
```