

Nome e Cognome: _____

Matricola: _____

1. 6 punti Riportare le conversioni di tipo implicite e scrivere quanto valgono alla fine le variabili x , k , z , sapendo che `UINT_MAX` vale 4294967295 (le conversioni da `*_MAX` non comportano conversioni).

**`z == (UINT_MAX + 1) - 1 == 4294967295`
`x == 0` (identico ad esempio su slide 21 conversioni)**

```
1 unsigned int x = UINT_MAX, k= 10U, z = -1;
2 int a= -2.51;
3 long long y = LLONG_MAX;
4 long j= 2LL;
5 x = y - x;
6 k = a < k;
```

linea 1: -1 da int a unsigned int
linea 2: 2.5 da long double a int
linea 4: 2LL da long long int a long int
linea 5: d da unsigned int a long long, y-x da long long a unsigned int (regola 2 slide 12 su conversioni)
linea 6: a da in a unsigned int, il risultato a < k ha tipo int che viene quindi convertito a unsigned int

alla linea 6 a viene convertito da -2 a (UINT_MAX + 1) - 2 che non è minore di 10, quindi la comparazione è falsa e k == 0

2. 5 punti Su foglio protocollo, scrivere una singola porzione di codice (*prova.c*) con almeno un oggetto memorizzato in ciascuna zona di memoria possibile. Evidenziare per ognuno la zona di memoria in cui sono memorizzati, e anche se è definito o dichiarato.

3. 6 punti Evidenziare con una freccia ciascun sequence point nel codice sottostante. Quanti effetti collaterali ci sono su a in totale (su carta: non eseguire il ciclo)? Scrivere una espressione con 4 effetti collaterali su una variabile a e 2 effetto collaterale su b , che NON generi un warning *multiple unsequenced modifications*, e una espressione che lo generi. Evidenziare i sequence point in entrambe le espressioni.

```
1 while (a++) {
2     a > 0 ? a++ : a-- ;
3     a++ || a++;
4     a, a--, a+= 1;
5 }
6
```

`a++`, `a++`, `a++`, `a++`, `b++`, `b++` //non genera warning
`a = a++`, `a++`, `a++`, `b++`, `b++` //genera warning
i sequence point sono uno per ogni operatore virgola

Guardando semplicemente al codice, ci sono 5 effetti collaterali se l'operatore OR (||) non viene eseguito (corto circuito), 6 se invece viene eseguito

4. 6 punti Data la seguente *struct* scrivere la definizione una funzione di nome *copia_2metà* che prende come parametro una lista (*lista1*) e crea una nuova lista (*lista2*) che contiene una copia della seconda metà degli elementi (il campo *info*) di *lista1*. *lista2* è ritornato come risultato dalla funzione. PER esempio, se *lista1* è 2-1-6-7, *lista2* conterrà 6-7.

```
1 struct Node {
2     int info;
3     struct Node* pNext;
4 }
```

5. 7 punti Cerchiare le affermazioni vere dato $int\ a[5] = \{INT_MAX - 7, 1287, INT_MIN + 528, -10, 312\}$; $short\ int\ *p = (short\ int*)\ a$; $char\ *q = (char*)\ a$; $p[3] = SHRT_MAX$, $p[5] += 2048$, $q[18] = \sim q[19]$; sapendo che i tre tipi usati occupano 4, 2, e 1 byte, con valori rappresentati in *little endian* e complemento a due. Scrivere la mappa di memoria e giustificare le affermazioni (vere o false).

- A. $(q[0] | q[1]) + q[17]$ B. $*((short*) \& q[17]) > 0$ **C. $q[9] + q[11] > -120$**

Esercizio 2

```
int main() {
    static int a= 0; //a in memoria permanente
    int* b= (int*) malloc(sizeof(int));
    return 0;
}
```

a in memoria permanente
b nello stack (di main)
4 byte allocati con la malloc in memoria dinamica

a e b sono definiti

Esercizio 5

00011111

11111111

11111111

11111110

11100000

10100000

11111111

11111110

00001000

01000000

00000000

00010001

01101111

11111111

11111111

11111111

00011100

10000000

11111111

00000000

A: $q[0] \mid q[1] = 11111111$ che equivale a -1
 $q[17] == 1$
 $-1 - (-1) == 0$ FALSA

C: $q[9] == 2$,
per $q[11]$ applicare l'algoritmo slide 27
sui tipi: sottrarre 1 (11100001) e poi flippare i bit
(00011110) ci dice che $q[11]$ vale -120
 $2 + (-120) == -118 > -120$ VERA

B: $*((short*) \&q[17])$ è minore di zero, visto il bit più significativo uguale a 1 FALSA

Esercizio 4

```
struct Node* copia_meta(struct Node* lista1) {
    struct Node* pScan= lista1;
    int contatore = 0;
    while (pScan != NULL) {
        contatore++;
        pScan = pScan -> pNext;
    }
    contatore /= 2;
    pScan= lista1;
```

```
for(int i = 0; i < contatore; i++) {
    pScan = pScan -> pNext;
}
```

```
struct Node* lista2= NULL;
struct Node* pInsert= NULL;
```

```
while(pScan != NULL) {
    struct Node* pNew= (struct Node*) malloc(sizeof (struct Node));
    pNew -> info = pScan-> info;
```

```
if (lista2 == NULL) {
    lista2= pNew;
    pInsert= pNew;
```

```
else {
    pInsert-> pNext= pNew;
    pInsert= pNew;
```

```
pScan = pScan -> pNext;
```

```
return lista2;
}
```