Prova scritta Programmazione Procedurale - 7 Luglio 2022

Nome e Cognome: _____ Matricola: _____

1. [6 punti] Supponendo che un *long long* occupi 8 byte e un *int* 4 byte, spiegare cosa stampa la *printf* e perché stampa quel valore, mostrando il contenuto della variabile *b* in memoria.

```
long long int a= (LLONG_MAX-UINT_MAX)+1;
int b= a;
printf("b: %d\n", b);
}
```

2. [6 punti] Su foglio protocollo scrivere una funzione che prende come parametro una matrice con n righe e m colonne, e restituisce un'array contentente tutti gli elementi della riga a metà della matrice (se il numero di righe è pari...).

3. [6 punti] Data la seguente *struct Node* definire su foglio protocollo una funzione di nome *inserisci_2 posizione()* che inserisce un nuovo valore intero passato come parametro sempre in seconda posizione, se la lista ha già almeno un elemento (decidere come gestire se la lista ha zero elementi): se la lista originale è 7-4-11 e il valore passato è 5, la nuova lista sarà 7-5-4-11. Supporre di avere un puntatore ad inizio lista globale di nome *pFirst*.

```
struct Node {
  int info= 0;
  struct Node* pNext= NULL;
}
```

4. [5 punti] Evidenziare con una freccia ciascun sequence point nel codice sottostante. Quanti effetti collaterali ci sono su a (supponendo $a = 1$) in totale? Scrivere una espressione con 3 effetti collaterali su una variabile a e 1 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.

```
    while (a++) {
      a > 0? a+=1 : a-= 1;
      a++ || a++;
      a, a--, a+= 1;
    }

```

La condizione del ciclo while non è mai falsa (uguale a 0), provocando così "infiniti" effetti collaterali
corretto anche 5, guardando il codice per una sola iterazione
b= a++, a++, a+=1 (non genera warning)
b= a++ + a++, a+=1 (genera warning)

5. [7 punti] Cerchiare le affermazioni vere dato *int a[5]= {13+2 * 32, 129, [2]= INT_MIN + 47, 130939, 131072 * 2 +65}; short int *p = (short*) a; char *q= (char*) a; q[2]= -1; p[3]= 128 * 2;* sapendo che i tre tipi usati occupano 4, 2, e 1 byte, e $131072 = 2^{17}$ (valori rappresentati in *little endian* e complemento a due). Scrivere la mappa di memoria e giustificare le affermazioni (vere o false). Gli operatori | e & eseguono rispettivamente l'*or* e l'*and* bit-a-bit dei due operandi.

A. $(*(p+8) - *(q+2) - q[6]) > 65)$ B. $(\&a[4] - (a+1)) + *(q+18) - 7$ C. $(*(q+2) \,|\, q[8]) + (q[7] \,\&\, q[13])$

ESERCIZIO 1

LLONG_MAX 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111110 -
UINT_MAX 11111111 11111111 11111111 11111111 00000000 00000000 00000000 00000000 =
00000000 00000000 00000000 00000000 11111111 11111111 11111111 11111110 +
1 10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 =
10000000 00000000 00000000 00000000 11111111 11111111 11111111 11111110
che vale 1 eliminando i 32 bit più significativi
esercizio identico a slide

Esercizio 5
(mappa di memoria, vedere correzione 14 Gennaio 2020)

10110010
00000000
11111111
00000000

A: (65 - (-1) -0) > 65 VERA
B: 3 + 4 - 7 == 0 FALSA
con &a[4] - (a+1) == 3 ci sono tre interi tra quei due puntatori
C: -1 + 1 == 0 FALSA con *(q+2) == 11111111 | 11110100 == q[8]
il risultato dell'or bit a bit è 11111111 che equivale a -1

10000001
00000000
00000000
10000000

Esercizio 2

```
int* fun (int n, int m, int matrice[n][m]) {
    int* rarray = (int*) calloc(m, sizeof(int));
    int riga = (n - 1) % 2;
    for  (int i =0; i < m; i++)
        array[i] = matrice[riga][i];
    return rarray;
}
```

11110100
00000000
00000000
00000001

11011110
11111111
10000000
00000000

Esercizio 3

10000010
00000000
00100000
00000000

```
void inserisci_2posizione (int nuova_info) {
    if (pFirst == NULL) {
        printf("Lista vuota, non inserisco %d", nuova_info);
        return;
    }
    struct Node* nuovo_elem = (struct Node*) malloc(sizeof(struct Node));
    nuovo_elem - > info = nuova_info;
    nuovo_elem -> pNext == NULL;
    if (pFirst -> pNext == NULL)
        pFirst -> pNext = nuovo_elem;
    else
        struct Node* pTemp = pFirst -> pNext;
        pFirst -> pNext = nuovo_elem;
        nuovo_elem -> pNext = pTemp;
}
```

si può rendere ancora più compatta…come?