Prova scritta Programmazione Procedurale con Lab. - 30 Gennaio 2026

Nome e Cognome: _____ Matricola: _____

Le soluzioni che verranno valutate sono solamente quelle riportate in questo foglio. Utilizzare (e consegnare) i fogli protocollo utilizzati per i calcoli.

1. [3 punti] Marcare le affermazioni vere.

   ◯ Le funzioni *malloc()* e *calloc()* restituiscono la quantità di byte allocati.
   ☒ C'è un errore in: $int\, a; int^* p = \&a; free(p);$
   ◯ *malloc()* alloca la memoria ed inizializza tutti i byte a 0.
   ☒ *calloc(5,3)* e *malloc(15)* allocano la stessa quantità di byte in memoria.

2. [3 punti] Elencare le parti che compongono *gcc* in ordine di esecuzione (solo i loro nomi).

   Preprocessore - Compilatore - Assembler - Linker

3. [3 punti] Marcare le affermazioni vere che riguardano la funzione *realloc()* (Soluzione obbligatoria nel riquadro).

   ◯ La *realloc* prende come parametro solamente una quantità di byte.
   ☒ La *realloc* ritorna un valore di tipo $void^*$.
   ☒ La *realloc* può ritornare 0 come valore.
   ◯ L'indirizzo di memoria ritornato dalla funzione è sempre uguale all'indirizzo passato alla *realloc*.

4. [4 punti] Descrivere la regola di conversione applicata alla linea 4.

```
1 int i = -1;
2 unsigned int limit = 200U;
3
4 if ( i < limit )
5  printf("%d", i);
6
```

Regola 1 su slide conversioni tipo

La Regola1 dice che, se in un'espressione il tipo di x è unsigned TipoT (parliamo di tipi interi) il cui grado di conversione è per lo meno tanto alto tanto quello dell'altro operando (y), allora il tipo dell'altro operando (y) è convertito ad unsigned TipoT.

Alla linea 4, l'operando di tipo int (i) viene convertito a unsigned int (il tipo di limit).

5. [6 punti] Scrivere cosa stampa il seguente programma, sapendo che *a* si trova all'indirizzo 0x7ffee4399ffe.

```
1     int a= 0x1b, i= 3, *b= &a;
2
3     for (int* p= &i; (a-= 1) ? (*p++, --a) : ((*p)+=2, a); *p++)
      {
4       a= (a - i);
5       printf("%d %d OK\n", a, *p);
6       if (a <= 0) {
7         a= 1;
8         continue; }
9     }
10    printf("%d %p\n", a, ((short*) b) + 1);
```

22 _ OK
17 _ OK
12 _ OK
7 _ OK
2 _ OK
0 0x7ffee439a000

I valori indicati con _ sono indefiniti perché il puntatore p, dopo essere stato incrementato, viene dereferenziato quando non punta più all'oggetto i. L'accesso a memoria fuori dall'oggetto di origine non è definito in C e dipende dall'implementazione.

6. 5 punti Scrivere una funzione che prende una matrice $n \times n$ (righe per colonne) di valori *int* come parametro e restituisce un array contenente tutti gli elementi sulla seconda diagonale.

```
int *seconda_diagonale(int n, int m[n][n]) {
    if (n <= 0) {
        return NULL;
    }

    int *diag = malloc(n * sizeof(int));
    if (diag == NULL) {
        return NULL;
    }

    for (int i = 0; i < n; i++) {
        diag[i] = m[i][n - 1 - i];
    }

    return diag;
}
```

7. 6 punti Cerchiare le affermazioni vere dato *long long a[3]= {1537, -67, (LLONG_MAX + 1) + 512}; int *p = (int*) a; char *q= (char*) a; p[1]= INT_MAX, p[4]+= 2048, q[19]= ∼q[19];* sapendo che i tre tipi usati occupano 8, 4, 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.) $(\sim (p[3] \,\&\, p[1])) == p[5]$ (B.) $*((long\ long^*)(\&p[1])) < *((long\ long^*)(\&p[2]))$ (C.) $((long\ long^*)(\&p[1])) < ((short\ int^*)(\&p[2]))$