Prova scritta Programmazione Procedurale con Lab. - 4 Giugno 2024

Nome e Cognome: _____ Matricola: _____

1. [6 punti] Elencare le conversioni di tipo, specificando per ogni valore il tipo di partenza ed il tipo ottenuto durante la conversione (*da .. a*). Supporre *UINT_MAX == 4294967295* e *INT_MAX == 2147483647*. Quale è la precisione di un *float*, e quale quella di un *double*? Riportare il valore finale di *a*, *c* e *d*, e se non si è in grado di stabilire il valore, giustificare il perché.

```
1 double f(float a) {return (a-1);}
2
3 int main () {
4   unsigned int a = -1;
5   int b = -2;
6   float c = f((b < a )? b: a);
7   double d = INT_MAX - 2;
8 }
```

l4: -1 da int a unsigned int
l6: in guardia, b da int ad unsigned int, b vale -2 + (UINT_MAX + 1) quindi guardia espressione condizionale falsa, a convertito da unsigned int a float in f(a). a vale -1 + (UINT_MAX + 1) quindi a vale 4294967295
l1: -1 da int a float
l1: a-1 da float a double
l6: risultato di f da double a float (6 cifre significative).
c può quindi non essere esattamente rappresentabile perché a passato a f vale 4294967295, quindi più delle cifre rappresentabili precisamente da un float.
l8: INT_MAX - 2 da int a double, 2147483645 ha meno di 15 cifre significative (che è la precisione del double) e quindi è esattamente rappresentabile

2. [6 punti] Data la seguente definizione di *struct Node*, definire su foglio protocollo una funzione di nome *compara_unisci()* che riceve come parametri due liste e controlla se sono uguali. In caso affermativo, restituisce una lista finita che corrisponde alla concatenazione delle due liste.

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

3. [6 punti] Su foglio protocollo, scrivere una funzione *unisci()* che, presi come parametri due array, per prima cosa controlli se la lunghezza dei due array è la stessa. La funzione restituisce un nuovo array che corrisponde alla unione dei due array eseguito secondo lo schema: *array1[0]*, *array2[0]*, *array1[1]*, *array2[1]*, etc...

4. [5 punti] Per ogni identificatore di variabile e funzione, riportare il suo linkage.

```
1 #define A 3
2 int a= A;
3 int a;
4 extern int b;
5 extern int compare(double, double);
6 static double area(double);
7
8 int* my_func(int c) {
9   static double e= 4.0;
10  double* f= &e;
11  auto int q= 4;
12  extern int b;
13  return (int*) f;
14 }
```

l2: a linkage esterno
l3: a linkage esterno
l4: b linkage esterno
l5: compare linkage esterno
l6: area linkage interno
l8: my_func linkage esterno
l8: c no linkage
l9: e no linkage
l10: f no linkage
l11: q no linkage
l12: b linkage esterno

5. [7 punti] Cerchiare le affermazioni vere dato *int a[6]={1707,-761,37,INT_MAX,(INT_MAX+INT_MIN)+1 ,-1}; long long *p = (long long *) a; short *q= (short*) a; p[2]+= 255, q[7]+= 1, q[5]= ~q[5];* sapendo che i tre tipi usati occupano 8, 4, e 2 byte, con valori rappresentati in *little endian* e complemento a due. Scrivere la mappa di memoria e giustificare le affermazioni (vere o false). L'operatore $\sim$ è l'operatore di negazione bit a bit. A. $((char^*)(\&p[1]) < p[1])$; (B.) $(\sim a[4] < \sim a[2])$.

Esercizio 2

```
struct Node* compara_unisci(struct Node* lista1, struct Node* lista2) {
  if (lista1 != lista2) {
    printf("Le liste sono diverse\n");
    return NULL;
  }
  if (lista1 == NULL) {
    printf("Le liste sono vuote\n");
    return NULL;
  }
  struct Node* nuova_lista = NULL;
  struct Node* pLast_nuova_lista = NULL;
  struct Node* pTemp = lista1;
  while (pTemp != NULL) {
    struct Node* pNew = (struct Node*) malloc(sizeof(struct Node));
    pNew -> info = pTemp -> info;
    pNew -> pNext = NULL;
    if (pLast_nuova_lista == NULL)
      pLast_nuova_lista = pNew;
    else
      pLast_nuova_lista -> pNext = pNew;
      pLast_nuova_lista = pNew;
      pTemp = pTemp -> pNext;
    }
    pTemp = lista2;
    while (pTemp != NULL) {
      struct Node* pNew = (struct Node*) malloc(sizeof(struct Node));
      pNew -> info = pTemp -> info;
      pNew -> pNext = NULL;
      pLast_nuova_lista -> pNext = pNew;
      pLast_nuova_lista = pNew;
      pTemp = pTemp -> pNext;
    }
    return nuova_lista;
}
```

Esercizio 3

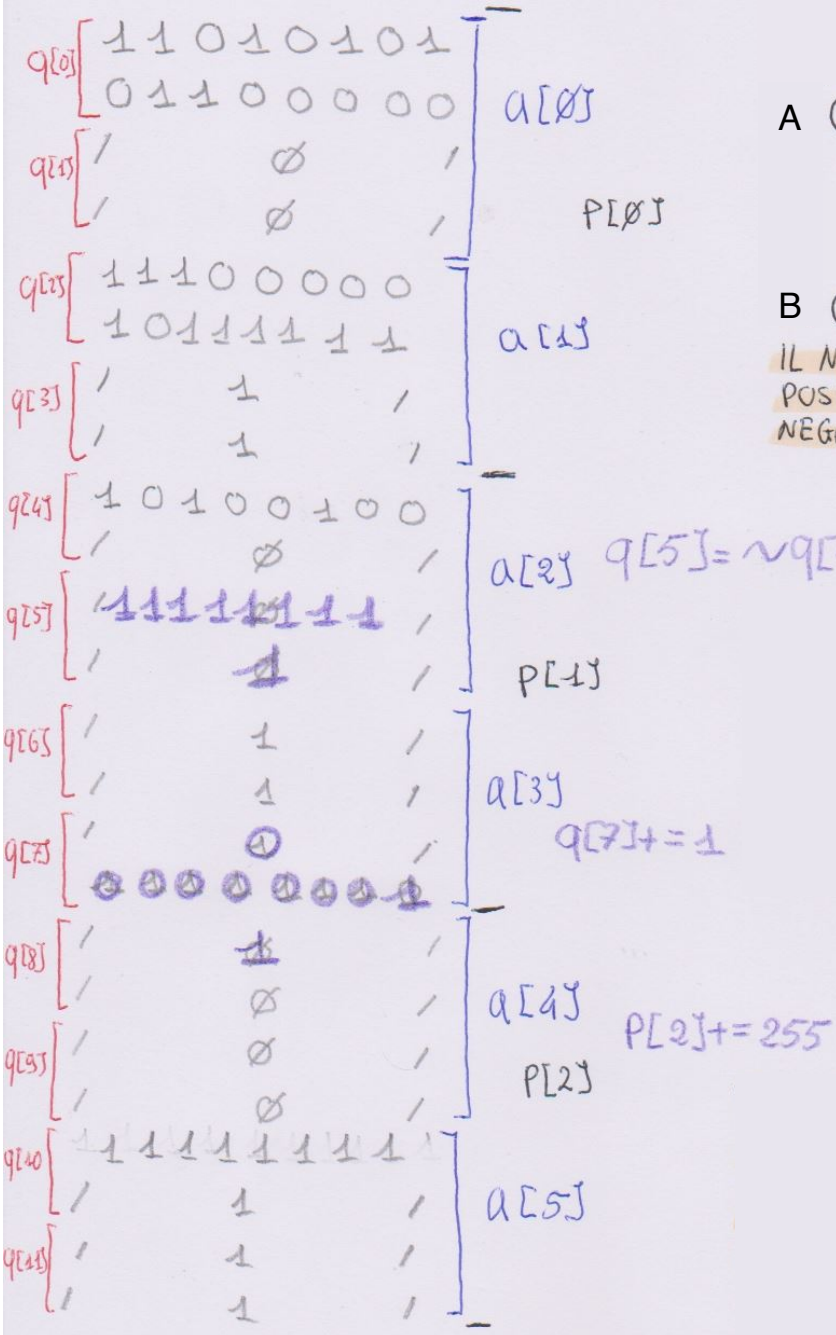
```
int* unisci(int* array1, int l1, int array2[], int l2) {
  if (l1 != l2) {
    printf("Errore lunghezze \n");
    return NULL;
  }
  int* array_unito = (int*) calloc(l1 + l2, sizeof(int));
  int i=0;
  for (int k= 0; i < l1; i++, k+=2) {
    array_unito[k]= array1[i];
    array_unito[k+1] = array2[i];
  }
  return array_unito;
}
```

Esercizio 5

int a[6] = {1707, -761, 37, INT_MAX, (INT_MAX+INTMIN)+1, -14}

long long *p = (long long*) a;

SHORT * q = (SHORT*) a;



A (CCHAR*)(&P[1]) < P[1]) F

37 < N° NEGATIVO

B (~a[4] < ~a[2]) T

IL NEGATO DI UN NEGATIVO È POSITIVO E DI UN POSITIVO È NEGATIVO, BANALMENTE VERA