Prova scritta Programmazione I - 17 Aprile 2019 - FILA A

Nome e Cognome: _____ Matricola: _____

1. [3 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).

```
1 unsigned int x = UINT_MAX, k= 10U, z
      = -1,;
2 int a= -2.5l;
3 long long y = LLONG_MAX;
4 long j= 2LL;
5 x = y - x;
6 k = a < k;
```

z == (UINT_MAX +1) - 1 == 4294967295

x == 0 (identico ad esempio su slide 21 conversioni)

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. [4 punti] Su foglio protocollo, scrivere un programma che prende da input un valore intero n, e stampa tutti i numeri primi tra 0 e n: per esempio, se $n == 14$, stampare $2, 3, 5, 7, 11, 13$. Stampare come in figura.

```
2
3 5
7 11 13
17 19 23 29
31 37 41 43 47
```

3. [3 punti] Scrivere cosa stampa il seguente programma, sapendo che b si trova all'indirizzo 0x1047e601c.

```
1 int a = 0x19, b= 017;
2
3 void f(void) {
4   while (b > 7? b-=2 : b) {b--; ++a;} }
5
6 int main() {
7   printf("%p\n", &b);
8   for (int i= 1 ; f(), b= b+i; i++, printf("%d
      %d\n", a, b))
9     if (i > 4) break;
10    else do {(*&a)++;} while (!a && a++);
11  printf("%p", ((long long*) &b) +1);
12 }
```

0x1047e601c
35 1
37 2
40 3
44 4
0x1047e6024

4. [4 punti] Data la seguente struttura che rappresenta ogni nodo di una lista, definire una funzione di nome *duplicate* che prende come parametro *int infoToDuplicate* ed il puntatore al puntatore di una lista (cioè ad un *Node**). La funzione aggiunge un nuovo elemento con valore *infoToDuplicate* in testa alla lista solo se la lista contiene già un elemento con valore identico a *infoToDuplicate*.

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

5. [3 punti] Su foglio protocollo, scrivere una funzione ricorsiva di nome *palindrome* che controlla se una stringa, passata come parametro alla funzione è palindroma oppure no (*return* 1 oppure 0). Non utilizzare variabili *static* o globali. Per esempio, *itopinonavevanonipoti* è palindroma (destra-sinistra = sinistra-destra).

https://www.sanfoundry.com/c-program-string-palindrome-using-recursion/

Esercizio 2

```
#include<stdio.h>

int isPrime(int v) {
   for (int k = v - 1; k > 1; k--)
      if (v % k == 0)
         return 0;
   return 1;
}

int main () {

   int n = 0;
   printf("Passami n: ");
   scanf("%d", &n);

   if (n < 2) {
      printf("n deve essere maggiore strettamente di 1\n");
      return 0;
   }

   int primeVector[n];
   int count = 0;

   for (int i= 2, j=0; i < n; i++)
      if (isPrime(i)) {
         primeVector[j] = i;
         count = ++j;
      }


   int row = 1, index = 0;
   while (1) {
      for (int j = 1; j <= row && count > 0; j++) {
         printf("%d ", primeVector[index]);
         count--;
         index++;
      }
      printf("\n");
      row++;
      if (count == 0)
         break;
   }
}
```

Esercizio 5

```
void duplicate(Node** pFirst, int infoToDuplicate) {
  if (*pFirst == NULL) {
    printf("Empty list\n")
    return;
  }

  // Controllo se infoToDuplicate è dentro la lista
  isIn = 0;
  Node* pScan = *pFirst;
  do {
    if (pScan -> info == infoToDuplicate) {
      isIn = 1;
      break;
    }
    pScan = pScan->pNext;
  } while(pScan!= NULL);

  if (isIn) {
    Node *pNew = (Node*) malloc(sizeof(Node));
    pNew->info = infoToDucplicate;
    pNew->pNext= NULL;

    if(*ppFirst == NULL)
    *ppFirst = pNew;
    else {
      pNew-> pNext= *ppFirst;
      *pFirst= pNew;
    }
  }
}
```

1

2

1 è esattamente uguale allo scorrimento di una lista (slide 16 su liste)

2 è esattamente uguale all'inserzione in testa (slide 21 su liste)

6. [3 punti] Su foglio protocollo, scrivere il contenuto di due file, *main.c* e *secondary.c*. I due file devono essere compilati insieme producendo con successo l'eseguibile di nome *myexe*: scrivere il comando per fare ciò. I due file devono poter essere compilati anche indipendentemente per produrre un file oggetto; scrivere il comando per fare ciò. In *main.c* ci devono essere esattamente un tentativo di definizione che rimane dichiarazione, una definizione di variabile e una di funzione entrambi con *external linkage*, una dichiarazione di funzione con *external linkage*. In *secondary.c* ci devono essere esattamente una definizione di funzione con *external linkage*, una variabile con *internal linkage* e due variabili con *no linkage*.

7. [4 punti] Scrivere cosa stampa il seguente programma.

```
1  int a, r= 2;
2
3  int f(int* y, int *x, int z) {
4    extern int a;
5    a = a + 1;
6    if (a < 4) {
7      r += ++(*x) + ++(*y), (*y+= z);
8      r= r + f(x, y, z);
9      printf("%d %d %d %d\n", *x, *y, z, r);
10     return (r/a);
11     printf("%d\n", r++); }
12   else
13     return r/a; }
14
15 int main(void) {
16   int a= 3, b= 2, c= 5;
17   b= f(&b, &c, a);
18   printf("%d %d %d\n", a, b, c); }
```

11 11 3 55
11 11 3 38
11 11 3 20
3 5 11

8. [4 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]= ∼ 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.) $(((int)(q+13) - (int)(a+1)) - q[1])\,\%\,3$ D. $(*(p+7)\,\&\,p[9])\,\%\,5$ (E.) $q[9] + q[11] > -120$ (F.) $\sim (\&p[5] - \&p[1] - q[9] * 2)$

9. [2 punti] Quali affermazioni sono vere? ◯ le espressioni contengono comandi; ◯ *i++;* è una espressione; ⊗ *break* è un comando. ⊗ *a= 4* è una espressione; ◯ *return* ha come argomento un comando. ◯ i comandi sono separati da virgole.

10. [1 punti] Data l'espressione $b = a + + + c + +$ riportare qui il numero di effetti collaterali presenti: **3** .

a++
c++
b= ...

Esercizio 6

main.c

```
int a;
int a= 10;

void fun(int);

int main(void) {
 fun(a);
}
```

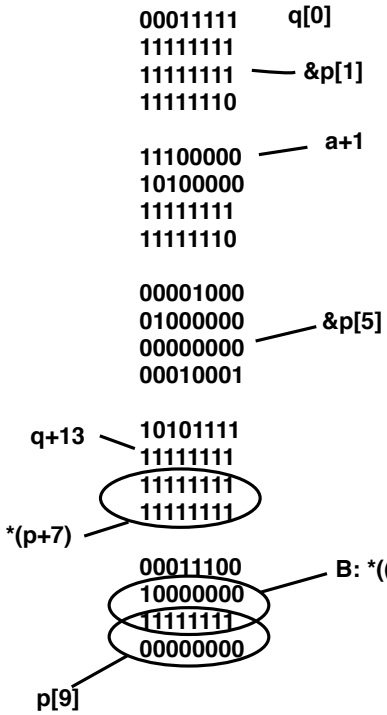
secondary.c

```
static int c= 4;

void fun(int);

void fun(int b) {
  int d= 4;
  c+= b + d;
}
```

```
gcc -c main.c
gcc -c secondary.c
gcc -o myexe main.o secondary.o (o .c)
```



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

C: (int)(q+13) - (int) (a+1) == 9
q[1] == -1
9 - (-1) % 3 == 1 VERA

D: 11111111 11111111 == *(p + 7)
11111111 00000000 == p[9]
in and bit a bit il risultato è p[9] che equivale a 255%5 == 0 FALSA

E: 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 VERA

F: &p[5] - &p[1] == 4 (restituisce il numero di short int tra i due puntatori)
q[9] * 2 == 4
&p[5] - &p[1] - (q[9] * 2) == 0, che negato bit a bit porta a VERA