

c vale 0.0

d vale 4294967292

Nome e Cognome: _____

Matricola: **rappresentabile con un double che ha precisione doppia, rappresenta 15 cifre**

1. **3 punti** Riportare le conversioni di tipo implicite, scrivere il valore di finale di c e d , dato $USHRT_MAX = 65535$ e $UINT_MAX = 4294967295$. In caso non sia possibile stabilire il valore di d , spiegare perché.

```
1 double f(float a) {return (a - 2);}
2
3 int main(void) {
4     unsigned short a = -65534;
5     unsigned int b = -1;
6     float c= f((b < a)? b : a);
7     double d= UINT.MAX - 3;
8 }
```

a convertito a uint vale $(65535 + 1) - 65534 == 2$

b convertito a uint $(4294967295 + 1) - 1 == 4294967295$

linea 4: -65534 da int a unsigned int

linea 5: -1 convertito da int a unsigned int

linea 6: in $b < a$, a da unsigned short a unsigned int (risultato comparazione convertito a int)

linea 6: la condizione è falsa, a convertito da ushort a uint.

(l'espressione condizionale ritorna un solo tipo)

linea 6: a convertito poi da uint a float

linea 1: -2 convertito da int a float

linea 1: il risultato convertito da float a double

linea 6: $f(a)$ convertito da double a float

linea 7: $UINT_MAX - 3$ convertito da long oppure long long a double

2. **3 punti** Scrivere cosa stampa la seguente porzione di codice.

```
1 int n= 5, k = 1, b, i, j;
2 for(i=0; i<n; i++) {
3     for(b=0; b <= n-i; b++)
4         printf("-");
5     for(j=0; j < i; j++) {
6         if (j==0 || i==0) k = 1;
7         else
8             k*= (i-j+2)/j-1;
9         printf("%d", k); }
10    printf("\n"); }
```

```
-----
-----1
-----1-2
----1-3-0
---1-4-4-0
```

3. **3 punti** Scrivere cosa stampa la seguente porzione di codice. Che valore contiene la variabile b alla fine del programma?

```
1 int a= 0x3d, i= -5, *b= &a;
2 for (int* p= &i; (a++, (*p)++) ?
    (++(*p), (a--)-1) : ((*p)+=3, a-1)
    ; (*p)++) {
3     --a;
4     printf("%d %d \n", a, *p);
5     if (*p > 3) {
6         a= (!(-a) && a++) ? 3 : 2;
7         break; }
8     else continue; }
9 printf("%d\n", a);
```

```
60 -3
59 0
58 3
57 6
2
```

4. **3 punti** Su foglio protocollo, scrivere la definizione di una funzione *arrayDiff* che prende due array di *int* come parametri, controlla che abbiano la stessa lunghezza, e ritorna la somma delle differenze tra elementi in posizioni corrispondenti: $a[0] - b[0] + a[1] - b[1] + \dots$. Trattare i casi particolari rispetto a input.
5. **4 punti** Data la seguente struttura, definire una funzione di nome *ins_head* che prende come parametri un puntatore a *struct Node* (il puntatore al primo elemento della lista) e un valore *int*, e inserisce un nuovo elemento in testa alla lista (cioè come primo elemento), con il campo *info* assegnato al valore passato. La funzione non crea il nuovo elemento e stampa un errore, se il campo *info* del nuovo elemento è minore del campo *info* dell'elemento successivo in lista.

Esercizio 4

Passare solo un puntatore al primo elemento non permette di modificare il puntatore al primo elemento della lista. Quindi si deve passare un puntatore a puntatore (struct Node**). Considerata comunque giusta anche la funzione con un pFirst globale.

```
void ins_head(struct Node** pFirst, int infoField) {  
    if (*pFirst != NULL)  
        if (((*pFirst) -> pInfo) < infoField) {  
            printf("Errore parametro infoField minore del successivo elemento")  
            return;  
        }  
    else  
        struct Node *pNew = (struct Node*) malloc(sizeof(struct Node));  
        pNew-> pNext= NULL;  
        pNew-> pInfo= infoField;  
        if(*pFirst == NULL)  
            *pFirst = pNew;  
        else {  
            pNew-> pNext= *pFirst;  
            *pFirst= pNew;  
        }  
    return;  
}
```

Esercizio 5

```
int arrayDiff(int n1, int a[], int n2, int b[]) {  
    if ((n1 != n2) || (n1==0)) {  
        printf("Errore sulla lunghezza degli array")  
        return -1;  
    }  
    else {  
        int sum= 0;  
        for (i= 0; i < n1; i++)  
            sum+= a[i] - b[i];  
        return sum;  
    }  
}
```

Esercizio 6

```
unsigned findMax (unsigned int num) {  
    int cifra, max;  
    if (num<10)  
        return num;  
  
    cifra = num%10;  
    num = num/10;  
  
    max = findMax(num);  
    if (cifra>max)  
        return cifra;  
    else  
        return max  
}
```

Esercizio 7

```
int a; // Tentativo (dichiarazione)  
int a= 0;  
void f1(void);  
  
int main(void {  
    extern int a; // Linkage esterno  
    f1();  
}  
  
main.c
```

```
#include<stdio.h>  
static int b= 0; // Linkage interno  
  
void f1 (void) { // f1 ha linkage esterno  
    int c= 3; // No linkage  
    printf("Ciao %d", b + c);  
}  
  
write.c
```

I due comandi sono
gcc -c main.c
gcc -c write.c

```

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

```

6. **3 punti** Su foglio protocollo, scrivere una funzione ricorsiva di nome *findMax* che ritorna la massima cifra del valore di tipo *unsigned int* passato come parametro. Per esempio, chiamata con 395 deve ritornare 9, con 54321 deve ritornare 5.
7. **3 punti** Su foglio protocollo, scrivere due file, *main.c* (che contiene la funzione *main*) e *write.c*: *main.c* non può esser compilato (ed eseguito) da solo per produrre un eseguibile, ha bisogno di essere collegato a *write.c*. In *main.c* ci deve essere un tentativo di definizione che rimane dichiarazione e una variabile locale alla funzione *main* con *linkage* esterno. Il file *write.c* deve contenere esattamente una definizione con *linkage* esterno e una con *linkage* interno; questo file deve inoltre contenere esattamente una definizione con *no linkage*. Scrivere inoltre i due comandi per creare i file oggetto dei due file.
8. **4 punti** Scrivere cosa stampa il seguente programma.

```

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

```

```

5 1 -3 6
5 2 -3 13
5 3 -3 20
5 20 -3

```

9. **4 punti** Cerchiare le affermazioni vere dato $\text{int } a[4] = \{7+4*64, \text{INT_MIN} + 39, [2] = 131002, 131072/2+111\}$; $\text{short int } *p = (\text{short}*) a$; $\text{char } *q = (\text{char}*) a$; $*(q+3) = -1$; $*((\text{short int}*) \&q[5]) = 257$; 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 $\&$ ritornano rispettivamente l'*or* e l'*and* bit-a-bit dei due operandi, \sim è la negazione bit a bit, mentre $>>$ rappresenta l'operatore di *shift* di n posizioni a destra, inserendo 0 nelle posizioni eliminate (operazione fatta nel processore).
- ☒ A. $p[3] > \text{SHRT_MIN}$ B. $q[5] - q[6] + q[10] - q[14]$ ☒ C. $((\&a[4] - a) + -p[5]) \% 2$ ☒ D. $((\text{int}) (a + 2) - (\text{int}) \&q[2]) + *(q + 14)) \% 2$ ☒ E. $(*(q + 3) \& q[1]) + q[2]$ ☒ F. $((q[12] >> 4) | q[4]) >= 35$

```

      11100000 — a
q[1] 10000000
q[2] 00000000
*(q+3) 11111111

q[4] 11100100
q[5] 10000000 — p[3]
q[6] 10000000
      00000001

01011101 — a+2
11111111
q[10] 10000000
      00000000

q[12] 11110110
      00000000
q[14] 10000000
      00000000

xxxxxxx — &a[4]
xxxxxxx
xxxxxxx
xxxxxxx

```

A: p[3] vale SHRT_MIN + 1, quindi VERA

B: $1 - 1 + 1 - 1 == 0$ quindi FALSA

C: $\&a[4] - a == 4$ (ci sono 4 interi tra i due indirizzi) + -1 ($-q[5]$). Il risultato è $3\%2$ che equivale a 1, quindi VERA

D: Tra a+2 e &q[2] la differenza è 6 (byte), q[q4] vale 1, la loro somma è 7 che modulo 2 fa 1, quindi VERA

con $*((short\ int*) \&q[5]) = 257$, prima si controlla l'indirizzo di &q[5], poi se si converte a puntatore a short int, si inserisce 257 nei successivi due byte

E: $*(q+3) == 11111111$ in and a bit a bit con $q[1] == 10000000$ il risultato è 10000000 (che equivale a 1), sommato a q[1] (che vale 0), il risultato è 1, quindi VERA

F: q[12] nel processore 01101111
 $q[12] \gg 4 == 0000110$ in or bit a bit
con q[4] == 00100111

00000110 |
00100111 ==
00100111 che in big endian (siamo nei registri del processore) vale 39. Quindi VERA