Esercizi Programmazione procedurale con laboratorio - 31 dicembre 2020

1. facile Il codice seguente stampa il risultato desiderato? (10) In caso negativo spiegare il motivo e descrivere un modo per poter correggere il problema.

```c
#include <stdio.h>

int *buildArray() {
  int vettore[100];
  vettore[0] = 10;
  return vettore;
}

int main() {
  int *ptrVettore = buildArray();
  printf("%d", ptrVettore[0]);
  return 0;
}
```

Il codice non stampa 10. La funzione buildArray() restituisce l'indirizzo di una variabile locale che potrebbe non esistere in memoria quando la chiamata a funzione è terminata. Un modo intuitivo per correggere il problema consiste nel dichiarare il vettore nel metodo main() e passarlo alla funzione in qualità di puntatore a intero. Si veda codice soluzione.

2. medio Cosa stampa il seguente codice?

```c
#include <stdio.h>

void swap(int a, int b) {
    int tmp; tmp = a; a = b; b = tmp;
}

void sumValue(int x, int *pc) {
    *pc = x + 10;
}

int main() {
    int a = 1, b = 5, *pc = &b;
    swap(a, b);
    sumValue(a, pc);
    printf("%d", b);
    return 0;
}
```

11

Spiegazione: la funzione swap() è fittizia perché i parametri vengono passati per valore, quindi nel main, le variabili a e b, non vengono sostituite. sumValue() riceve quindi 1 come primo parametro e, come secondo parametro, il puntatore a intero pc, il quale punta ad una locazione di memoria a cui viene assegnato il valore della somma (1+10=11) tramite l'operatore di dereferenziazione. A questo punto, poiché al puntatore pc era stato precedentemente assegnato lo stesso indirizzo di b, printf stampa il valore 11, ovvero lo stesso valore della locazione di memoria puntata da pc.

3. difficile Cosa stampa il seguente codice?

```c
#include <stdio.h>
static int *d = NULL;

int f(int *a, int b) {
    d = &b;
    return (--(*a) * b++ + ++(*d));
}

int main(void) {
    static int a = -2, b = 2;
    b = f(&a, b);

    printf("%d\n%d\n%d", b, a, *d);
}
```

-2
-3
4

La funzione prende in input due parametri: il primo è un puntatore a intero, il secondo è un intero. Viene assegnato al puntatore d l'indirizzo di b. A questo punto viene ritornata da f un'espressione calcolata attraverso l'uso di vari operatori, tra cui quello di dereferenziazione:
b = -3 * 2 + 4 = -2
a = -3
d = 4

4. difficile Si consideri una funzione matematica F(0) = 1, F(i) = F(i-1) + i+1 Il seguente codice, alloca dinamicamente la memoria e inizializza l'array con i valori della funzione data. Completare le parti mancanti.

```c
int *buildDynamicArray(int *arr, int size) {
    arr = (int *) malloc(                    );

    int count = 0, preValue = 0;
    for (int i = 0; i < size; i++) {
            preValue = *(       ) = preValue + ++count;
    }

    return arr;
}
```

riga 27:
arr = (int *) malloc(size * sizeof(int));

riga 31:
preValue = *(arr + i) = preValue + ++count;

5. difficile Si consideri una funzione per la generazione di una matrice di dimensioni nr * nc, con memoria allocata dinamicamente. Tale matrice conterrà valori di tipo float.

   (a) completare le parti mancanti affinché sia allocata la giusta quantità di memoria nell'heap e sia gestita correttamente l'inizializzazione di tutti i valori attraverso l'aritmetica dei puntatori.

   (b) Descrivere, inoltre, una soluzione alternativa che faccia uso di puntatori doppi e metta in luce la stretta correlazione, presente nel linguaggio C, tra puntatori ed array.

```
1  float *buildSinglePointerMatrix(float
       *arr, int nr, int nc) {
2      arr = (float *) malloc(nr * nc *
           sizeof(float));
3
4      int i, j, count = 0, preValue = 0;
5      for (i = 0; i < nr; i++) {
6          for (j = 0; j < nc; j++) {
7              *(arr + i * nc + j) =
                   (float) count++/125;
8          }
9      }
10
11     return arr;
12 }
```

(a) riga 14: arr = (float *) malloc(nr * nc * sizeof(float));
riga 19: *(arr + i * nc + j) = (float) count++/125;

(b) se si usa un array doppio di puntatori a float, è possibile allocare la memoria per costruire un vettore di dimensioni nr. Ogni riga i di tale vettore, punterà a sua volta ad un vettore di dimensione nc. A questo punto, supposto che i sia l'indice della riga e j l'indice della colonna, si potrà accedere ad ogni cella della matrice attraverso arr[i][j] oppure tramite l'aritmetica dei puntatori nel seguente modo *(*(arr+i)+j).
Si veda codice soluzione.

6. facile Definire una struct cellula per rappresentare un automa cellulare le cui caratteristiche risiedono nel numero di vicini e nell'esistenza (vivo o morto). Scrivere una semplice funzione transizione che, data una cellula viva, restituisce la cellula morta, qualora il numero di vicini sia maggiore di 3 o minore di 2. Altrimenti restituisce la cellula in input.

Si veda code soluzione.

7. facile Cosa stampa il seguente codice?

   (a) 4

   (b) 8

   (c) produce errore in compilazione

   (d) produce errore a runtime

```
1 #include <stdio.h>
2
3 struct test {
4     static int var;
5 };
6
7 int main() {
8     printf("%d", sizeof(struct test));
9 }
```

Una variabile static int misura 4 byte, tuttavia la risposta giusta è la c), in quanto, nel linguaggio C, i tipi struct e union non possono avere membri statici.

8. medio Dati int a = 2, *pa = &a, b = 0; Indicare, per ogni espressione, se produce un risultato verso (diverso da zero) o falso (zero). Si assume che, tali espressioni, siano indipendenti tra loro (es. l'esecuzione della seconda non influenza la terza).

```
(4>5 && 5>4 || !(3, 2, 1, 0)) && !!!0

(a--, --a, *pa)

a == pa

(!a ? 0 : 1) && !b
```

vero

falso

falso

vero

9. medio Cosa stampa il seguente codice?

```
1  #include <stdio.h>
2
3  int main() {
4      int i = 0xF - 5;
5      for(; i < 0x10; i++) {
6          printf("%d\n", i);
7      }
8
9      printf("%d\n", !i ^ 33);
10 }
```

10
11
12
13
14
15
33

10. facile Dato il seguente codice, indicare le affermazioni vere.

(a) la riga 7 è corretta, i è un lvalue;

(b) la riga 8 non è corretta, 7 non è un un lvalue;

(c) la riga 9 è corretta in quanto j *4 è un rvalue;

(d) la riga 10 produce un errore a runtime poiché ci è un rvalue;

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int i, j;
6      const int ci = 7;
7
8      i = 7;
9      7 = i;
10     j * 4 = 7;
11     ci = 9;
12 }
```

(a) vera

(b) vera

(c) falsa

(d) falsa: l'errore viene prodotto a compile time

11. medio Il seguente codice è corretto? In caso negativo, spiegare come può essere corretto per ottenere il risultato intuitivamente desiderato.

```
1  #include <stdio.h>
2  int main(void) {
3      char s[] = "stampa la stringa";
4      for (; *s != '\0'; ++s) {
5          printf("%c", *s);
6      }
7  }
```

L'operazione di incremento ++s equivale a s = s + 1 e quindi richiede un lvalue. La variabile s, in questo caso, è un array e quindi non si può modificare l'indirizzo di un array ma solo i singoli elementi di esso. Nella soluzione riportata (si veda il codice fornito), è stato inizializzato un puntatore a char (ptr) con l'indirizzo di s. Diventa così possibile gestire la variabile ptr attraverso l'artimetica dei puntatori.

12. medio Il seguente frammento definisce una struct di un nodo utile per implementare una linked list e una funzione per l'attraversamento e la stampa dei valori. Descrivere l'implementazione di una nuova funzione buildList() che, dato un numero k in input, crei una lista FIFO di k nodi.

```
1  #include <stdio.h>
2  struct Node {
3      int data;
4      struct Node* next;
5  };
6
7  void printList(struct Node* n) {
8      while (n != NULL) {
9          printf("%d ", n->data);
10         n = n->next;
11     }
12 }
```

Si veda soluzione allegata.

13. difficile Elencare tutte le conversioni di tipo presenti nel seguente codice.

```
1  long int f(double a, long double b) {
2      int val1 = 10;
3      short int val2 = 5;
4      if (a < val1 || b < val1) {
5          return (a > b ? val1 : val2);
6      }
7  }
8  int main(void) {
9      int a = 4.5;
10     int b = 2LL;
11     b = f(b, a);
12 }
```

riga 9: 4.5 viene convertito da double a int
riga 10: 2LL viene convertito da long long a int
riga 11: b da int a double
riga 11: a da int a long double
riga 11: il risultato della funzione viene convertito da long int a int
riga 4: val1 da int a double
riga 4: la seconda val1 da int a long double
riga 5: a viene convertito da double a long double
riga 5: il risultato dell'espressione condizionale viene convertito in long int da int se val1 o da short int se val2

14. facile Il risultato della divisione è 5.200? Motivare la risposta e, in caso negativo, descrivere l'eventuale correzione.

```c
#include <stdio.h>
int main(void) {
    int a = 5, b = 2;
    float division =  a / b;
    printf("%.3f", division);
    return 0;
}
```

Il risultato stampato è 2 poiché entrambi gli operandi (a e b) sono di tipo int, quindi avviene il troncamento della parte frazionaria, nonostante il valore sia assegnato ad una variabile float. Per ottenere il risultato desiderato, è necessario applicare la conversione forzata attraverso il casting:

float division = (float) a / (float) b;

15. difficile Il seguente frammento definisce una struct di un nodo utile per implementare una linked list. Ogni nodo punta al successore attraverso il puntaore a Node *next*, ad eccezione dell'ultimo nodo che punterà a NULL. Supponiamo inoltre che gli elementi, siano ordinati in ordine crescente sulla base del valore della variabile *data*. Implementare una funzione *int elimina(int key, struct Node **radice);* che elimini l'elemento avente chiave key. Se quest'ultimo è presente, dovrà deallocarlo e restituire 1. Se non presente, la funzione dovrà restituire 0.

```c
struct Node {
    int data;
    struct Node* next;
};
```

Si veda soluzione allegata.

SOL. ESERCIZIO 1

```c
#include <stdio.h>

void fillArray(int *ptrVettore) {
    ptrVettore[0] = 10;
}

int main() {
    int vettore[100];
    fillArray(vettore);
    printf("%d", vettore[0]);
    return 0;
}
```

SOL. ESERCIZIO 5

```c
float *buildDoublePointerMatrix(float **arr, int nr, int nc) {
    arr = (float *) malloc(nr * sizeof(float));
    for (int i = 0; i < nr; i++) {
        arr[i] = malloc(nc * sizeof(float));
    }

    int count = 0;
    for (int i = 0; i < nr; i++) {
        for (int j = 0; j < nc; j++) {
            arr[i][j] = (float) count++/125;
        }
    }

    return arr;
}
```

SOL. ESERCIZIO 6

```c
struct cellula {
    int numeroVicini;
    int vivo;
};

struct cellula transizione(struct cellula c) {
    if (c.vivo && (c.numeroVicini < 2 || c.numeroVicini > 3)) {
        c.vivo = 0;
    }

    return c;
}
```

SOL. ESERCIZIO 11

```c
#include <stdio.h>
int main(void) {
    char s[] = "stampa la stringa";
    for (char *ptr = s; *ptr != '\0'; ++ptr) {
        printf("%c", *ptr);
    }
}
```

SOL. ESERCIZIO 12

```c
struct Node* buildList(int k, struct Node* head) {
    struct Node* tail = NULL;
    struct Node* tmp = NULL;
    int inputValue;

    printf("Creazione linked list con %d elementi\n", k);

    for (int i = 0; i < k; i++) {
        printf("Inserisci l'elmento %d: ", i);
        scanf("%d", &inputValue);

        if (i == 0) {
            head = (struct Node*) malloc(sizeof(struct Node));
            head->data = inputValue;
            head->next = NULL;

            tail = head;
        } else {
            tmp = (struct Node*) malloc(sizeof(struct Node));
            tmp->data = inputValue;
            tmp->next = NULL;

            tail->next = tmp;
            tail = tmp;
        }
    }

    return head;
}
```

SOL. ESERCIZIO 15

```c
int elimina(int key, struct Node **radice){
   while( (*radice != NULL) && ((*radice)->data < key)){
      radice = &( (*radice)->next );
   }
   if ((*radice != NULL) && ((*radice)->data == key)){
      struct Node *next;
      next=(*radice)->next;
      free(*radice);
      *radice=next;
      return(1);
   } else {
      return(0);
   }
}
```