Algoritmi 1 – Sperimentazioni
Quiz sul Linguaggio C

Marco Guazzone
DiSIT, Università del Piemonte Orientale
marco.guazzone@uniupo.it

1 Cosa stampa?

Per ciascuno dei seguenti frammenti di codice, dire (senza implementare) cosa viene stampato in output.

* * *

Esercizio 1:

```
#define P printf
#define NL putchar('\n')

int x = 10;

P("%d", x+1); NL;
```

* * *

Esercizio 2:

```
int x;

x = - 3 + 4 * 5 - 6; printf("%d\n", x);
x = 3 + 4 % 5 - 6; printf("%d\n", x);
x = 3 * 4 % 6 / 5; printf("%d\n", x);
x = (7 + 6) % 5 / 2; printf("%d\n", x);
```

* * *

Esercizio 3:

```c
int x, y, z;

x = 2; y = 1; z = 0;
x = x && y || z; printf("%d\n", x);
printf("%d\n", x || ! y && z);

x = y = 1;
z = x ++ - 1; printf("%d\n%d\n", x, z);
z += - x ++ + ++ y; printf("%d\n%d\n", x, z);
```

* * *

Esercizio 4:

```c
double d = 3.2, x;
int i=2, y;

x = (y=d/i)*2; printf("x=%g, y=%g\n", x, (double) y);
y = (x=d/i)*2; printf("x=%g, y=%g\n", x, (double) y);

y = d * (x=2.5/d); printf("y=%g\n", (double) y);
x = d * (y = ((int) 2.9 + 1.1)/d); printf("x=%g, y=%g\n", x,
    (double) y);
```

* * *

Esercizio 5:

```c
int x, y, z;

x = y = 0;
while (y < 10) ++y; x += y;
printf("x=%d, y=%d\n", x, y);

y = x = 0;
while (y < 10) x += ++y;
printf("x=%d, y=%d\n", x, y);

y = 1;
while (y < 10)
{
    x = y++;
    z = ++y;
}
printf("x=%d, y=%d, z=%d\n", x, y, z);

for (y = 1; y < 10; ++y) ;
printf("x=%d, y=%d\n", x, y);
```

```
21
22 for (y = 1; (x=y) < 10; y++) ;
23 printf("x=%d, y=%d\n", x, y);
24
25 for (x = 0, y = 1000; y > 1; ++x, y /= 10)
26     printf("x=%d, y=%d\n", x, y);
27
28 y = 0;
29 do
30 {
31     printf("y=%d\n", y);
32     --y;
33 }
34 while (y > 0);
```

* * *

Esercizio 6:

```
1  int reset();
2  int next(int);
3  int last(int);
4  int new(int);
5
6  int i = 1;
7
8  int main()
9  {
10     int i, j;
11
12     i = reset();
13     for (j = 1; j <= 3; ++j)
14     {
15         printf("i = %d, j = %d\n", i, j);
16         printf("next = %d\n", next(i));
17         printf("last = %d\n", last(i));
18         printf("new = %d\n", new(i+j));
19     }
20 }
21
22 int reset()
23 {
24     return i;
25 }
26
27 int next(int j)
28 {
29     return j = i++;
30 }
```

```
31
32 int last(int j)
33 {
34     static int i = 10;
35     return j = i--;
36 }
37
38 int new(int i)
39 {
40     int j = 10;
41     return i= j += i;
42 }
```

* * *

Esercizio 7:

```
1  int a[] = {0,1,2,3,4};
2  int i, *p;
3
4  for (i = 0; i <= 4; ++i) printf("a[%d]=%d\n", i, a[i]);
5  putchar('\n');
6  for (p = &a[0]; p <= &a[4]; ++p)
7      printf("*p=%d\n", *p);
8  printf("\n\n");
9
10 for (p = &a[0], i = 1; i <= 5; ++i)
11     printf("p[%d]=%d\n", i, p[i]);
12 putchar('\n');
13 for (p = a, i = 0; p+i <= a+4; ++p,++i)
14     printf("*(p+%d)=%d\n", i, *(p+i));
15 printf("\n\n");
16
17 for (p = a+4; p >= a; --p) printf("*p=%d\n",*p);
18 putchar('\n');
19 for (p = a+4, i = 0; i <= 4; ++i) printf("p[%d]=%d\n", -i,
       p[-i]);
20 putchar('\n');
21 for (p = a+4; p >= a; --p ) printf("a[%ld]=%d\n", p-a,
       a[p-a]);
22 putchar('\n');
```

* * *

Esercizio 8:

```
int a[] = {0,1,2,3,4};
int *p[] = {a, a+1, a+2, a+3, a+4};
int **pp=p;

printf("*a=%d\n", *a);
printf("**p=%d\n", **p);
printf("**pp=%d\n", **pp);
putchar('\n');

pp++; printf("pp-p=%ld, *pp-a=%ld, **pp=%d\n", pp-p, *pp-a,
    **pp);
*pp++; printf("pp-p=%ld, *pp-a=%ld, **pp=%d\n", pp-p, *pp-a,
    **pp);
*++pp; printf("pp-p=%ld, *pp-a=%ld, **pp=%d\n", pp-p, *pp-a,
    **pp);
++*pp; printf("pp-p=%ld, *pp-a=%ld, **pp=%d\n", pp-p, *pp-a,
    **pp);
putchar('\n');

pp = p;
**pp++; printf("pp-p=%ld, *pp-a=%ld, **pp=%d\n", pp-p,
    *pp-a, **pp);
*++*pp; printf("pp-p=%ld, *pp-a=%ld, **pp=%d\n", pp-p,
    *pp-a, **pp);
++**pp; printf("pp-p=%ld, *pp-a=%ld, **pp=%d\n", pp-p,
    *pp-a, **pp);
putchar('\n');
```

* * *

Esercizio 9:

```
char *p = "abcdefgh";
int *p2 = (int*) p;

p2 +=1;
p = (char*) p2;

printf("%s\n", p);
```

* * *

Esercizio 10:

```
void incr1(int);
void incr2(int*);
void incr3(int**);

```

```c
int main()
{
    int v = 1;
    int *p = &v;

    incr1(v); printf("v=%d, p-&v=%ld\n", v, p-&v);
    incr1(p); printf("v=%d, p-&v=%ld\n", v, p-&v);
    incr2(&v); printf("v=%d, p-&v=%ld\n", v, p-&v);
    incr2(p); printf("v=%d, p-&v=%ld\n", v, p-&v);
    incr3(&p); printf("v=%d, p-&v=%ld\n", v, p-&v);

    return 0;
}

void incr1(int x)
{
    ++x;
}

void incr2(int *x)
{
    ++*x;
}

void incr3(int **x)
{
    *++*x;
}
```

2 Qual è il problema?

Per ciascuno dei seguenti frammenti di codice, individuare le problematiche e proporre una soluzione per risolvere.

* * *

Esercizio 11:
Verifica se due numeri sono uguali.

```c
int x = 10, y = 5;

if (x = y)
    printf("x e y sono uguali\n");
else
    printf("x e y sono diversi\n");
```

* * *

Esercizio 12:
Verifica un valore.

```
int x;
int y = x+1;

if (y == 1)
    printf("OK\n");
else
    printf("KO\n");
```

* * *

Esercizio 13:
Limita a una soglia massima.

```
int x = 5;
int max = 10;

if (x > max);
    x = max;

printf("x: %d\n", x);
```

* * *

Esercizio 14:
Stampa il colore giusto.

```
enum colors_t
{
    red,
    green,
    blue
};

enum colors_t color = green;

switch (color)
{
    case red:
        printf("Red\n");
    case green:
        printf("Green\n");
    case blue:
        printf("Blue\n");
    default:
        printf("Unknown\n");
}
```

* * *

Esercizio 15:
Itera da *n* a 0 (estremi inclusi).

```
void itera(unsigned int n)
{
    unsigned int i;

    for (i = n; i >= 0; --i)
        printf("i: %u\n", i);
    }
}
```

* * *

Esercizio 16:
Itera da *n* a 0 (estremi esclusi).

```
void itera(unsigned int n)
{
    unsigned int i;

    for (i = n-1; i > 0; --i)
        printf("i: %u\n", i);
}
```

* * *

Esercizio 17:
Incrementa una variabile.

```
int v = 1;
int *ptr1 = NULL, ptr2 = NULL;

ptr1 = &v;
ptr2 = ptr1;
*ptr2 += 1; *@/printf("
```

* * *

Esercizio 18:
Stringhe costanti.

```
char ary[] = "Hello";
char *ptr = "hello";

strcpy(ary, "World");
strcpy(ptr, "world");

printf("%p -> %s\n", ary, ary);
printf("%p -> %s\n", ptr, ptr);
```

* * *

Esercizio 19:
Operatore **sizeof** su array e puntatori.

```
int ary[] = {0,1,2,3,4,5,6,7,8};
int *ptr = ary;

size_t len_ary = sizeof ary/sizeof ary[0];
size_t len_ptr = sizeof ptr/sizeof *ptr;

if (len_ary == len_ptr)
{
    printf("OK\n");
}
else
{
    printf("ary -> %lu\n", len_ary);
    printf("ptr -> %lu\n", len_ptr);
}
```

* * *

Esercizio 20:
Lunghezza di una stringa e operatore **sizeof**.

```
char ary[] = "Hello, World!";
size_t szof = sizeof ary/sizeof ary[0];
size_t len = strlen(ary);

if (szof == len)
{
    printf("OK\n");
}
else
{
    printf("sizeof -> %lu\n", szof);
    printf("strlen -> %lu\n", len);
}
```

* * *

Esercizio 21:
Valutazione dell'operatore **sizeof**.

```
int x = 0;
size_t size = sizeof(++x);
printf("size: %lu, x: %d\n", size, x);
```

* * *

Esercizio 22:
Visibilità delle variabili.

```
int *get_handle()
{
    int hnd = 10;

    return &hnd;
}
int main()
{
    int *hnd = get_handle();

    printf("Handle: %d\n", *hnd);

    return 0;
}
```

* * *

Esercizio 23:
Copia di una stringa.

```
void my_strcpy(char *dest, const char *src)
{
    if (!dest || !src)
        return;

    while (*src)
    {
        dest = src;
        ++src;
        ++dest;
    }
}
```

* * *

Esercizio 24:
Allocazione dinamica della memoria.

```
struct user_t
{
   char name[100];
};
typedef struct user_t user_t;

user_t *user = malloc(sizeof(user));

strcpy(user->name, "John Doe");
printf("Name: %s\n", user->name);

free(user);
```

3 Strutture dati di base

Implementare le seguenti strutture dati di base. Non è consentito utilizzare variabili globali.

* * *

Esercizio 25:
Implementare il tipo *lista concatenata*, in cui l'informazione memorizzata in ogni elemento è un valore intero, e le seguenti operazioni:

- inserimento di un elemento in testa alla lista (sono ammessi elementi duplicati),
- ricerca di un elemento nella lista (la funzione ritorna il puntatore al nodo della lista che contiene il valore cercato se il valore è presente nella lista, o `NULL` altrimenti),
- cancellazione di un elemento dalla lista (in caso di elementi duplicati, si rimuove la prima occorrenza che s'incontra),
- stampa il contenuto della lista su un file (passato come un parametro di tipo `FILE*`), utilizzando il seguente formato di output: `[valore1, valore2, ...]`.

Nella funzione `main()` effettuare le seguenti operazioni:

1. Inserire nella lista i valori della sequenza `[1,2,3,4,5,4,3,2,1]`.
2. Stampare il contenuto della lista.
3. Rimuovere dalla lista il primo, il secondo e l'ultimo valore della suddetta sequenza, e il valore `-10` (che non è memorizzato nella lista). Ogni volta che si cancella un elemento ricercare nella lista se è ancora presente.
4. Stampare il contenuto della lista.

5. Rimuovere tutti gli elementi.

6. Stampare il contenuto della lista.

L'output prodotto dovrebbe essere simile al seguente:

```
[1,2,3,4,5,4,3,2,1]
After removal -> Element 1 found
After removal -> Element 2 found
After removal -> Element 1 not found
After removal -> Element -10 not found
[3,4,5,4,3,2]
[]
```

* * *

Esercizio 26:

Implementare il tipo *lista concatenata ordinata*, in cui l'informazione memorizzata in ogni nodo è un valore intero e tale per cui i nodi sono ordinati rispetto al valore memorizzato (quindi un nodo non può contenere un valore inferiore a quelli memorizzati nei nodi precedenti), e le seguenti operazioni:

- inserimento di un elemento nella lista (sono ammessi elementi duplicati),
- ricerca di un elemento nella lista (la funzione ritorna il puntatore al nodo della lista che contiene il valore cercato se il valore è presente nella lista, o `NULL` altrimenti),
- cancellazione di un elemento dalla lista (in caso di elementi duplicati, si rimuove la prima occorrenza che s'incontra),
- stampa il contenuto della lista su un file (passato come un parametro di tipo `FILE*`), utilizzando il seguente formato di output: `[valore1, valore2, ...]`.

Nella funzione `main()` effettuare le seguenti operazioni:

1. Inserire nella lista i valori della sequenza `[1,2,3,4,5,4,3,2,1]`.
2. Stampare il contenuto della lista.
3. Rimuovere dalla lista il primo, il secondo e l'ultimo valore della suddetta sequenza, e il valore `-10` (che non è memorizzato nella lista). Ogni volta che si cancella un elemento ricercare nella lista se è ancora presente.
4. Stampare il contenuto della lista.
5. Rimuovere tutti gli elementi.
6. Stampare il contenuto della lista.

L'output prodotto dovrebbe essere simile al seguente:

```
[1,1,2,2,3,3,4,4,5]
After removal -> Element 1 found
After removal -> Element 2 found
After removal -> Element 1 not found
After removal -> Element -10 not found
[2,3,3,4,4,5]
[]
```

* * *

Esercizio 27:

Implementare il tipo *lista concatenata ordinata* di interi come l'esercizio precedente, in cui però non sono ammessi duplicati (cioè in cui non è possibile avere due o più nodi in cui è memorizzato lo stesso valore). In particolare, l'operazione d'inserimento non dovrà inserire nella lista un valore già presente in un suo nodo.

Nella funzione `main()` effettuare le seguenti operazioni:

1. Inserire nella lista i valori della sequenza `[1,2,3,4,5,4,3,2,1]`.
2. Stampare il contenuto della lista.
3. Rimuovere dalla lista il primo, il secondo e l'ultimo valore della suddetta sequenza, e il valore `-10` (che non è memorizzato nella lista). Ogni volta che si cancella un elemento ricercare nella lista se è ancora presente.
4. Stampare il contenuto della lista.
5. Rimuovere tutti gli elementi.
6. Stampare il contenuto della lista.

L'output prodotto dovrebbe essere simile al seguente:

```
[1,2,3,4,5]
After removal -> Element 1 not found
After removal -> Element 2 not found
After removal -> Element 1 not found
After removal -> Element -10 not found
[3,4,5]
[]
```

* * *

Esercizio 28:

Si consideri il tipo *pila* in cui gli inserimenti e le cancellazioni seguono la politica *Last-In First-Out* (LIFO; cioè l'ultimo elemento inserito è anche il primo a essere rimosso), e l'unico elemento a cui è possibile accedere è quello in cima alla pila. Implementare il tipo *pila* tramite lista contenata, in cui l'informazione memorizzata in ogni nodo è un valore intero, e le seguenti operazioni:

- inserimento (“push”) di un elemento in cima alla pila;
- cancellazione (“pop”) di un elemento dalla cima della pila;
- stampa il contenuto della pila su un file (passato come un parametro di tipo `FILE*`), utilizzando il seguente formato di output in cui l’elemento più a sinistra è quello in cima alla pila: `[valore1, valore2, ...]`.

Nella funzione `main()` effettuare le seguenti operazioni:

1. Inserire nella pila i valori della sequenza `[1,2,3,4,5,6,7,8,9]`.
2. Stampare il contenuto della pila.
3. Rimuovere i primi 3 elementi dalla cima della pila.
4. Stampare il contenuto della pila.
5. Rimuovere tutti gli elementi.
6. Stampare il contenuto della pila.

L’output prodotto dovrebbe essere simile al seguente:

```
[9,8,7,6,5,4,3,2,1]
[6,5,4,3,2,1]
[]
```

* * *

Esercizio 29:

Si consideri il tipo *coda* in cui gli inserimenti e le cancellazioni seguono la politica *Fist-In First-Out* (FIFO; cioè il primo elemento inserito è anche il primo a essere rimosso), e gli unici elementi a cui è possibile accedere sono quelli all’inizio (“front”) e alla fine (“back”) della coda. Implementare il tipo *coda* tramite lista contenata, in cui l’informazione memorizzata in ogni nodo è un valore intero, e le seguenti operazioni:

- inserimento (“enqueue”) di un elemento nella coda;
- cancellazione (“dequeue”) di un elemento dalla coda;
- stampa il contenuto della coda su un file (passato come un parametro di tipo `FILE*`), utilizzando il seguente formato di output in cui l’elemento più a sinistra è l’emento “front” e quello più a destra è l’elemento “back”: `[valore1, valore2, ...]`.

Nella funzione `main()` effettuare le seguenti operazioni:

1. Inserire nella coda i valori della sequenza `[1,2,3,4,5,6,7,8,9]`.
2. Stampare il contenuto della coda.
3. Rimuovere i primi 3 elementi dalla coda.
4. Stampare il contenuto della coda.

5. Rimuovere tutti gli elementi.
6. Stampare il contenuto della coda.

L'output prodotto dovrebbe essere simile al seguente:

```
[1,2,3,4,5,6,7,8,9]
[4,5,6,7,8,9]
[]
```