PROGRAMMAZIONE I (A,B) - a.a. 2016-17
Primo Appello – 19 gennaio 2017

Esercizio 1

Si scriva una funzione **C** che, dato un array a di dimensione dim e un numero naturale n tale che $1 \leq n \leq dim$, restituisce il valore di verità della seguente formula

$$\forall j \in [n, dim).\, (a[j] = \Sigma i : i \in [j-n, j).a[i])$$

Soluzione Il testo chiede in sostanza di scrivere una procedura che verifica se ogni elemento dell'array a dall'indice n in poi ha un valore pari alla somma degli n elementi precedenti. Ad esempio, per $n = 3$ il seguente array soddisfa la condizione

```
int a[7] = {1,2,3,6,11,20,37};
```

Seguendo la struttura della formula logica, una prima soluzione corretta prevede l'uso di due cicli annidati, o in maniera più modulare, mediante l'uso di una funzione ausiliaria

```
int somma(int a[], int init, int end) {
  int somma = 0;
  for (int k = init; k < end; k++)
    somma += a[k];
  return somma;
}

int check(int a[], int dim, int n) {
  if ((n < 1) || (n > dim)) return 0;
  for (int i = n; i < dim; i++)
    if (a[i] != somma(a, i-n, i))
      return 0;
  return 1;
}
```

Osservando che ad ogni iterazione del ciclo i valori da sommare sono quasi gli stessi rispetto a quelli sommati nell'iterazione precedente, una soluzione alternativa più efficiente è la seguente

```
int check(int a[], int dim, int n) {
  int i, somma = 0;
  for (i = 0; i < n; i++)
    somma += a[i];
  for (i = n; i < dim; i++) {
    if (somma != a[i])
      return 0;
    somma = somma - a[i-n] + a[i];
  }
  return 1;
}
```

Esercizio 2

Si dica qual è il linguaggio generato dalla seguente grammatica libera sull'alfabeto $\Lambda = \{a, b, c\}$

```
S -> aSb | aAb
A -> aaA | aBb
B -> Bbb | c
```

e se tale linguaggio è o meno regolare, fornendone la prova.

Soluzione Seguendo la struttura della grammatica, non è difficile ottenere il linguaggio

$$\{a^n a a^{2m} a c b^{2p} b b b^n \mid n, m, p \geq 0\}$$

Cercare di verificare la regolarità o meno del linguaggio così formulato può essere difficile. Conviene invece notare che il linguaggio può essere espresso come

$$\{a^2 a^n a^{2m} c b^{2p} b^n b^2 \mid n, m, p \geq 0\} \text{ e ancora come } \{a^2 a^q c b^r b^2 \mid q, r \geq 0 \wedge p\%2 = q\%2\}$$

A questo punto è immediato descrivere un automa deterministico che lo riconosce.

Esercizio 3

Si suppongano predefiniti i tipi

```
struct el {int info; struct el *next;};
typedef struct el ElementoDiLista;
typedef ElementoDiLista *ListaDiElementi;
```

Si scriva in **C** una procedura che, presi una lista e un numero naturale n, crea una copia dei primi n elementi della lista e aggiunge tali nuovi elementi in coda alla lista stessa. Se la lista contiene meno di n elementi, tutti gli elementi vengono copiati e aggiunti in coda.

Soluzione Una possibile soluzione consiste nel copiare un elemento per volta, andando via via ad aggiungere la copia in fondo alla lista.

```
void copia (ListaDiElementi l, int n)
{
  ListaDiElementi tail = l;
  ListaDiElementi oldtail;
  ListaDiElementi tmp;

  if (l!=NULL && n>0)
  {
    while (tail->next!=NULL)
       tail=tail->next;
    oldtail=tail;
    while (n>0 && l!=oldtail->next)
    {
      tmp = malloc(sizeof(ElementoDiLista));
      tmp->info = l->info;
      tmp->next = NULL;
      tail->next=tmp;
      tail = tail->next;
      n--;
      l=l->next;
    }
  }
}
```

Una soluzione alternativa consiste nel creare una nuova lista costituita dalle copie dei primi n elementi, da "appendere" in un colpo solo in coda alla lista originaria.

```
void copia (ListaDiElementi l, int n)
{
  ListaDiElementi copiati = NULL;
  ListaDiElementi last_copiati = NULL;
  ListaDiElementi tmp;
  ListaDiElementi curr = l;

  if (l!=NULL)
  {
```

```
    while (n>0 && curr!=NULL)
    {
      tmp = malloc(sizeof(ElementoDiLista));
      tmp->info = curr->info;
      tmp->next = NULL;
      if (copiati==NULL)
      {
        copiati=tmp;
        last_copiati=tmp;
      }
      else
      {
        last_copiati->next=tmp;
        last_copiati=tmp;
      }
      n--;
      curr=curr->next;
    }
    while (l->next!=NULL)
      l = l->next;
    l->next = copiati;
  }
}
```

Lo stesso approccio lo si poteva seguire relegando le oparazioni di copia e di aggiunta in fondo in procedure ausiliarie

```
void append (ListaDiElementi l1, ListaDiElementi l2)
{
  // si assume l1 non vuota
  while (l1->next!=NULL)
    l1 = l1->next;
  l1->next= l2;
}

ListaDiElementi copiaN (ListaDiElementi l, int n)
{
  ListaDiElementi ris = NULL;
  ListaDiElementi last = NULL;
  while (n>0 && l!=NULL)
  {
    ListaDiElementi tmp = malloc(sizeof(ElementoDiLista));
    tmp->info = l->info;
    tmp->next = NULL;
    if (ris==NULL)
      ris = tmp;
    else
      last->next = tmp;
    last = tmp;
    l = l->next;
```

```
    n--;
  }
  return ris;
}

void copia3 (ListaDiElementi l, int n)
{
  if (l!=NULL)
    append(l,copiaN(l,n));
}
```

Esercizio 4

Definire in CAML, senza usare la ricorsione esplicita, una funzione

```
multiset : 'a list -> ('a * int) list
```

che, data una lista `lis` di elementi ordinati in modo non decrescente, restituisce una lista di coppie in cui ciascuna coppia contiene un elemento di `lis` e il numero di volte in cui tale elemento occorre in `lis`. Nella lista risultante tutti i primi elementi delle coppie devono essere distinti tra loro. Ad esempio:

```
multiset [2;2;2;3;3;5;6;6;7;7;7] = [(2,3);(3,2);(5,1);(6,2);(7,3)]
```

Soluzione Una possibile soluzione è la seguente

```
let multiset lis =
  let f x y =
    match y with
        [] -> [(x,1)]
      | (x1,y1)::xs -> if (x1=x) then (x1,y1+1)::xs
                                 else (x,1)::(x1,y1)::xs
    in
      foldr f [] lis;;
```