PROGRAMMAZIONE I (A,B) - a.a. 2016-17
Appello Straordinario – 3 Novembre 2017

Esercizio 1

Data la seguente grammatica sull'alfabeto $\Sigma = \{a, b\}$

$$S \to aSc \mid aAc$$
$$A \to aAb \mid ab$$

si dia il linguaggio generato dalla grammatica e si verifichi se tale linguaggio è regolare o meno.

SOLUZIONE Il linguaggio generato dalla grammatica è il seguente

$$L = \{\ a^k b^m c^p\ \mid\ k, m, p > 0\ \wedge\ k = m + p\ \}$$

Il linguaggio non è regolare. Per dimostrarlo usando il Pumping Lemma scegliamo, per un generico $n \in I\!N$, la stringa

$$w = a^{2n} b^n c^n$$

Con questa scelta, considerando tutte le possibili suddivisioni di w in xyz tali che $|xy| \leq n$ e $y \neq \epsilon$ abbiamo:

$$x = a^r$$
$$y = a^s$$
$$z = a^t b^n c^n$$

con $s > 0$, $r + s \leq n$ e $r + s + t = 2n$. A questo punto, considerando la stringa $xy^i z$ con $i = 0$ otteniamo $a^r a^t b^n c^n$, con $r + t < 2n$, che quindi non appartiene al linguaggio L.

Esercizio 2

Si scriva una funzione **C** che, dato un array a di dimensione dim, restituisca il valore di verità della seguente formula:

$$\exists i \in [0, dim).\big(a[i] = \#\{\ j \mid j \in [0, dim)\ \wedge\ a[j] = a[i]\ \}\ \big)$$

dove $\#\{\ldots\}$ rappresenta il numero degli elementi (cardinalità) dell'insieme.

SOLUZIONE Modularmente, seguendo la struttura della formula possiamo definire una funzione per calcolare, dati a, dim e i, il valore di

$$\#\{\ j \mid j \in [0, dim)\ \wedge\ a[j] = a[i]\ \}$$

come segue:

```
int contauguali(int a[], int dim, int i) {
  int cont=0;
  for (int j=0; j<dim; j++)
    if (a[j]==a[i]) cont++;
  return cont;
}
```

A questo punto, la formula proposta dall'esercizio risulta essere equivalente alla seguente:

$$\exists i \in [0, dim).\big(a[i] = contauguali(a, dim, i)\ \big)$$

il cui valore di verità può essere calcolato come segue:

```
int check(int a[], int dim) {
  int i=0;
  int trovato=0;
  while (i<dim && !trovato) {
    if (a[i]==contauguali(a,dim,i))
      trovato=1;
    i++;
  }
  return trovato;
}
```

Una soluzione alternativa, non modulare, è la seguente:

```
int check(int a[], int dim) {
  int i=0;
  int trovato=0;
  while (i<dim && !trovato) {
    int cont=0;
    for (int j=0; j<dim; j++)
      if (a[j]==a[i]) cont++;
    if (a[i]==cont))
        trovato=1;
      i++;
    }
  return trovato;
}
```

Un'altra soluzione ancora, un po' più efficiente, può prevedere di interrompere il ciclo interno quando la variabile *cont* supera il valore di $a[i]$:

```
int check(int a[], int dim) {
  int i=0;
  int trovato=0;
  while (i<dim && !trovato) {
    int cont=0;
    int j=0;
    int superato=0;
    while (j<dim && !superato) {
      if (a[j]==a[i]) cont++;
      if (cont>a[i]) superato=1;
      j++;
    }
    if (a[i]==cont))
      trovato=1;
    i++;
  }
  return trovato;
}
```

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, presa una lista `l`, sposta in testa alla lista l'elemento contenente la prima occorrenza del valore massimo.

SOLUZIONE Una possibile soluzione è la seguente:

```
ListaDiElementi cercamax(ListaDiElementi l) {
  ListaDiElementi massimo = l;
  while (l!=NULL) {
    if (l->info > massimo->info)
      massimo = l;
    l = l->next;
  }
  return massimo;
}

void sposta(ListaDiElementi *l) {
  if (*l!=NULL) {
    ListaDiElementi max = cercamax(l);
    if (max != *l) {
      ListaDiElementi corr = (*l)->next;
      ListaDiElementi prec = *l;
      while (corr!=max) {
        prec = corr;
        corr = corr->next;
      }
      prec->next = corr->next;
      corr->next = *l;
      *l = corr;
    }
  }
}
```

Un'altra possibile soluzione un po' più efficiente (perché scorre la lista una volta sola) è la seguente:

```
void sposta(ListaDiElementi *l) {
  if (*l!=NULL) {
    ListaDiElementi curr = *l;
    ListaDiElementi prec = NULL;
    ListaDiElementi max = *l;
    ListaDiElementi precmax = NULL;
    while (curr!=NULL) {
      if (curr->info > max->info) {
```

```
        max = curr;
        precmax = prec;
      }
      prec = curr;
      curr = curr->next;
    }
    if (max != *l) {
      precmax->next = max->next;
      max->next=*l;
      *l=max;
    }
  }
}
```

Esercizio 4

Si definisca in CAML, senza usare la ricorsione esplicita, una funzione

```
contamax : int list -> int * int
```

che, data una lista `lis` di interi, restituisce la coppia `(max,n)` dove `max` è il valore massimo in `lis` e `n` è il numero di elementi che precedono la prima occorrenza di `max` nella lista.

SOLUZIONE Dal momento che non è possibile trovare un valore massimo in una lista vuota la funzione non sarà definita per la lista vuota.

```
let contamax lis =
  match lis with
    w::ws -> let f x (max,n) =
               if (x>=max) then (x,0) else (max,n+1)
             in
               foldr f (w,0) lis;;
```

Nota: è importante passare `lis` alla `foldr` altrimenti se `w` fosse il valore massimo la funzione darebbe come risultato `(w,n)` con `n` pari alla lunghezza della lista meno uno, invece che dare `(w,0)`.