**Fondamenti di Informatica T-1 (A.A. 2014/2015) - Ingegneria Informatica**
**Prof.ssa Mello**
**Prova Parziale d'Esame di Martedì 13 Febbraio 2015 – durata 1h**
**Totale 12 punti, sufficienza con 7**

## Compito A

### ESERCIZIO 1 (6 punti)

Date due liste ORDINATE di interi POSITIVI `a` e `b`, si realizzi una funzione RICORSIVA

```
list unify(list a, list b);
```

che restituisca una lista ORDINATA contenente gli elementi di `a` e di `b`, doppioni inclusi. Le due liste possono contenere dei valori errati, identificati dal valore `-1`, che devono essere scartati durante l'unione delle due liste. Per esempio, se `a= {-1,4,90,112,140,-1,180}` e `b= {1,4,42,-1,55}`, la funzione `unify()` deve restituire la lista `r= {1,4,4,42,55,90,112,140,180}`.

A tal fine, si può ricorrere ad una funzione RICORSIVA ausiliaria

```
list clean(list a);
```

che, data in ingresso una lista `l`, restituisca la stessa lista `l` priva dei termini `-1`. Qualora si scelga di utilizzare questa funzione, è necessario fornirne l'implementazione.

Le funzioni `unify()` e `clean()` dovranno essere implementate utilizzando le primitive dell'ADT lista. Si realizzi inoltre una semplice funzione `main()` di prova che invochi correttamente la funzione `unify()` creata.

### ESERCIZIO 2 (2 punti)

Si consideri la seguente grammatica G con scopo S, simboli non terminali {N, E, O, T} e simboli terminali { **+**, **-**, **2**, **4**, **9** }

```
S ::= TE | SE | E

N ::= NO | O | E

E ::= OT | TN | N

O ::= 2 | 4 | 9

T ::= + | -
```

La stringa **"-9+42"** appartiene al linguaggio di tale grammatica?

In caso affermativo se ne mostri la derivazione left-most.

**ESERCIZIO 3** (3 punti)

Il seguente programma C compila correttamente? In caso affermativo, quali sono i valori stampati a tempo di esecuzione? (si motivi opportunamente la risposta data)
Nota: si ricorda che al carattere `'\0'` corrisponde il valore numerico `0`.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char * fun(char * a, char * b){

	char * r;
	int h, stop, la, lb;

	la = strlen(a);
	lb = strlen(b);
	h = la<lb ? lb : la;

	r = (char*)malloc((h+1)*sizeof(char));

	for(;h>=0;--h)
		*(r+h) = la<lb ? b[h] : *(a+h);

	stop = la<lb ? la : lb;
	for(;h<=stop;++h)
		r[h] = *(a+h)>b[h] ? *(b+h) : a[h];

	return r;

}

int main(){

	char * s = "CURA";
	char * t = "IO";
	char * x;

	x = fun(s,t);

	printf("%s\n",x);

	return 0;

}
```

**ESERCIZIO 4** (1 punto)

Si definiscano i concetti di variabili globali e variabili locali, sottolineando in cosa differiscono e specificando in quali casi può o deve essere usato un tipo piuttosto che l'altro.

# Soluzioni

## ESERCIZIO 1

Versione senza `clean(list a)`

```
list unify(list a, list b){

	if(empty(a)&&empty(b))
		return emptyList();
	if(empty(a))
		if(head(b)!=-1)
			return cons(head(b),unify(a,tail(b)));
		else
			return unify(a,tail(b));
	if(empty(b))
		if(head(a)!=-1)
			return cons(head(a),unify(tail(a),b));
		else
			return unify(tail(a),b);
	if(head(a)==-1)
		return unify(tail(a),b);
	if(head(b)==-1)
		return unify(a,tail(b));
	if(head(a)<head(b))
		return cons(head(a),unify(tail(a),b));
	else
		return cons(head(b),unify(a,tail(b)));
}

int main(){

	list l1,l2,l3;

	l1=cons(-1,cons(4,cons(90,cons(112,cons(140,cons(-
1,cons(180,emptyList()))))))));
	l2=cons(1,cons(4,cons(42,cons(-1,cons(55,emptyList())))));

	l3=unify(l1,l2);

	while(!empty(l3)){
		printf("%d\n",head(l3));
		l3=tail(l3);
	}

	return 0;

}
```

Versione con `clean(list a)`

```
list clean(list a){

      if(empty(a))
            return emptyList();
      if(head(a)!=-1)
            return cons(head(a),clean(tail(a)));
      else
            return clean(tail(a));
}

list unify(list a, list b){
      if(empty(a))
            return b;
      if(empty(b))
            return a;
      if(head(a)<head(b))
            return cons(head(a),unify(tail(a),b));
      else
            return cons(head(b),unify(a,tail(b)));
}

int main(){
      list l1,l2,l3;

      l1=cons(-1,cons(4,cons(90,cons(112,cons(140,cons(-
1,cons(180,emptyList()))))))));
      l2=cons(1,cons(4,cons(42,cons(-1,cons(55,emptyList())))));

      l3=unify(clean(l1),clean(l2));

      while(!empty(l3)){
            printf("%d\n",head(l3));
            l3=tail(l3);
      }

      return 0;
}
```

## ESERCIZIO 2

La frase appartiene al linguaggio. In particolare, la si può ottenere tramite la seguente derivazione left-most:
S → SE → TEE → -EE → -OTE → -9TE →-9+E → -9+N → -9+NO → -9+OO → -9+4O → -9+42

## ESERCIZIO 3

Il programma è corretto sintatticamente e la sua esecuzione produce la stampa:

```
CO
```

La funzione `main()` dichiara due puntatori a carattere `s` e `t`, inizializzati rispettivamente con le stringhe `"CURA"` e `"IO"` e un terzo puntatore a carattere `x`. Invoca la funzione `fun()` passando come argomento le stringhe `s` e `t` e assegnando il risultato alla variabile `x`.

La funzione `fun()` dichiara un puntatore a carattere `r` e quattro variabili intere: `h`, `stop`, `la` e `lb`. Alla variabile `la` associa la lunghezza della stringa puntata dall'argomento `a` e alla variabile `lb` associa la lunghezza della stringa puntata dall'argomento `b`. Alla variabile `h` associa, tramite la valutazione di un'espressione condizionale, il valore di `lb` se quello di `la` ne è inferiore, altrimenti associa quello di `la`. Il puntatore `r` viene fatto puntare ad un'area di memoria allocata dinamicamente capace di contenere un numero di variabili di tipo `char` pari al valore di `h` più `1`. Esegue un primo ciclo `for` iterando sulla variabile `h`, decrementandola di `1` ad ogni step di esecuzione, finché questa è positiva. Ad ogni passo, copia nella posizione `h` dell'area di memoria indicizzata partendo dal puntatore a carattere `r` il carattere nella posizione corrispondente della stringa `b`, se la variabile `la` ha valore minore della variabile `lb`, o della stringa `a`. Terminato il ciclo, lo stato dell'area di memoria puntato da `r` visti gli argomenti passati dal `main()` sotto esame è il seguente:

```
['C','U','R','A','\0']
```

ed è priva di celle di memoria non inizializzate. Viene assegnata alla variabile `stop` il valore contenuto dalla variabile `la`, se questa è minore di `lb`, altrimenti assegna il valore contenuto da `lb`. L'assegnamento avviene attraverso la valutazione di un'altra espressione condizionale. Viene eseguito un secondo ciclo `for` iterando sulla variabile `h`, il cui valore è `-1` a seguito della terminazione del ciclo precedente, incrementandola di `1` ad ogni step di esecuzione fino a che è minore o uguale al valore contenuto nella variabile `stop`. Ad ogni passo, copia nella posizione `h` dell'area di memoria indicizzata partendo dal puntatore a carattere `r` il carattere nella posizione `h` della stringa `b`, se il valore del corrispondente carattere della stringa `a` ha codice maggiore (nella codifica ASCII) rispetto a questo, o della stringa `a`. Alla prima esecuzione di questo ciclo, viene fatto un accesso fuori dalle aree di memoria indicati dalle stringhe `r` e `a`: questo può provocare una terminazione del programma, se l'accesso avviene ad aree di memoria riservate. In caso contrario, viene sporcata la memoria del programma (cosa, nello specifico, non è prevedibile) e l'esecuzione prosegue. Terminato il ciclo, lo stato dell'area di memoria puntato da `r` visti gli argomenti passati dal `main()` sotto esame è il seguente:

```
['C','O','\0','A','\0']
```

ed è priva di celle di memoria non inizializzate. La funzione `fun()` restituisce infine l'indirizzo contenuto nella variabile `r`.

La funzione `main()` stampa sullo standard output il valore della stringa `x` (`CO`).