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 B

ESERCIZIO 1 (6 punti)

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

```
list consolidate(list l1, list l2);
```

che restituisca una lista ORDINATA (senso decrescente) contenente gli elementi di `l1` e di `l2`, doppioni inclusi. Le due liste possono contenere dei valori errati, identificati dal valore `-1`, che devono essere scartati durante l'operazione di consolidamento delle due liste. Per esempio, date le liste `l1= {55,-1,42,4,1}` e `l2= {180,-1,140,112,90,4,-1}`, la funzione `consolidate()` deve restituire la lista `l3= {180,140,112,90,55,42,4,4,1}`.

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

```
list refine(list l);
```

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 `consolidate()` e `refine()` dovranno essere implementate utilizzando le primitive dell'ADT lista. Si realizzi inoltre una semplice funzione `main()` di prova che invochi correttamente la funzione `consolidate()` 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 **"-42-9"** 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 * x, char * y){

	char * r;
	int h, halt, lx, ly;

	lx = strlen(x);
	ly = strlen(y);
	h = ly>lx ? ly : lx;

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

	for(;h>=0;h--)
		*(r+h)=ly>lx ? *(y+h) : x[h];

	halt = ly>lx ? lx : ly;
	for(;h<=halt;h++)
		r[h]=*(y+h)<x[h]? y[h] : *(x+h);

	return r;

}

int main(){

	char * s = "AMBO";
	char * t = "RE";
	char * x;

	x= fun(s,t);

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

	return 0;

}
```

ESERCIZIO 4 (1 punto)

Si discuta la differenza tra le due istruzioni seguenti:

```
#define X 5

int x = 5;
```

Soluzioni

ESERCIZIO 1

Versione senza `refine(list l)`

```
list consolidate(list l1, list l2){

	if(empty(l1)&&empty(l2))
		return emptyList();
	if(empty(l1))
		if(head(l2)!=-1)
			return cons(head(l2),consolidate(l1,tail(l2)));
		else
			return consolidate(l1,tail(l2));
	if(empty(l2))
		if(head(l1)!=-1)
			return cons(head(l1),consolidate(tail(l1),l2));
		else
			return consolidate(tail(l1),l2);
	if(head(l1)==-1)
		return consolidate(tail(l1),l2);
	if(head(l2)==-1)
		return consolidate(l1,tail(l2));
	if(head(l1)>head(l2))
		return cons(head(l1),consolidate(tail(l1),l2));
	else
		return cons(head(l2),consolidate(l1,tail(l2)));
}

int main(){

	list l1,l2,l3;

	l1=cons(55,cons(-1,cons(42,cons(4,cons(1,emptyList())))));
	l2=cons(180,cons(-1,cons(144,cons(112,cons(90,cons(4,cons(-
1,emptyList()))))))));

	l3=consolidate(l1,l2);

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

	return 0;

}
```

Versione con `refine(list l)`

```
list refine(list l){

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

list consolidate(list a, list b){

      if(empty(a)&&empty(b))
            return emptyList();
      if(empty(a))
            return b;
      if(empty(b))
            return a;
      if(head(a)>head(b))
            return cons(head(a),consolidate(tail(a),b));
      else
            return cons(head(b),consolidate(a,tail(b)));
}

int main(){

      list l1,l2,l3;

      l1=cons(55,cons(-1,cons(42,cons(4,cons(1,emptyList())))));
      l2=cons(180,cons(-1,cons(144,cons(112,cons(90,cons(4,cons(-
1,emptyList()))))))));

      l3=consolidate(refine(l1),refine(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 → -NE → -NOE → -OOE → -4OE → -42E → -42TN → -42-N → -42-O → -42-9

ESERCIZIO 3

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

AE

La funzione `main()` dichiara due puntatori a carattere `s` e `t`, inizializzati rispettivamente con le stringhe "AMBO" e "RE" 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`, `halt`, `lx` e `ly`. Alla variabile `lx` associa la lunghezza della stringa puntata dall'argomento `x` e alla variabile `ly` associa la lunghezza della stringa puntata dall'argomento `y`. Alla variabile `h` associa, tramite la valutazione di un'espressione condizionale, il valore di `ly` se è maggiore di quello di `lx`, altrimenti associa quello di `lx`. 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 `y`, se la variabile `ly` ha valore maggiore della variabile `lx`, o della stringa `x`. Terminato il ciclo, lo stato dell'area di memoria puntato da `r` visti gli argomenti passati dal `main()` sotto esame è il seguente:

['A','M','B','O','\0']

ed è priva di celle di memoria non inizializzate. Viene assegnata alla variabile `halt` il valore contenuto dalla variabile `lx`, se `ly` ne è maggiore, altrimenti assegna il valore contenuto da `ly`. 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 `halt`. 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 `y`, se il suo valore ha codice minore (nella codifica ASCII) del corrispondente carattere della stringa `x`, o della stringa `x`. Alla prima esecuzione di questo ciclo, viene fatto un accesso fuori dalle aree di memoria indicati dalle stringhe `r` e `x`: 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:

['A','E','\0','O','\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` (`AE`).