	Politecnico di Milano Scuola di Ingegneria Industriale e dell'Informazione **FONDAMENTI DI INFORMATICA** Appello 22 Febbraio 2016	COGNOME E NOME
		MATRICOLA
		Spazio riservato ai docenti

- Il presente plico contiene 5 esercizi e **deve essere debitamente compilato con cognome e nome, numero di matricola**.
- Il tempo a disposizione è di 1 ora e 45 minuti.
- Non separate questi fogli. Scrivete la soluzione solo sui fogli distribuiti, utilizzando il retro delle pagine in caso di necessità. Cancellate le parti di brutta con un tratto di penna.
- Ogni parte non cancellata a penna sarà considerata parte integrante della soluzione.
- È possibile scrivere a matita (e non occorre ricalcare al momento della consegna!).
- **È vietato utilizzare telefoni o pc**. Chi tenti di farlo vedrà annullata la sua prova.
- È ammessa la consultazione di libri e appunti.
- **Qualsiasi tentativo di comunicare con altri studenti comporta l'espulsione dall'aula**.
- È possibile ritirarsi senza penalità.
- **Non è possibile lasciare l'aula conservando il tema della prova in corso**.

Esercizio 1 (10 punti)

Si consideri una lista dinamica di interi definita come:

```
struct nodo
{
      int val;
      struct nodo *next;
};

typedef struct nodo *lista;
```

A. Implementare la seguente funzione in modo iterativo (cioè senza ricorsione):

   ```
   void separa(lista a, lista *pari, lista* dispari);
   ```

   che riceve in ingresso la testa di una lista di interi, **a**, e i puntatori a due liste vuote di interi, **pari** e **dispari**; la funzione, quindi popola la lista puntata dal parametro **pari** tutti gli elementi pari di **a** e popola la lista puntata a dal parametro **dispari** con tutti gli elementi dispari di **a**.

B. Implementare una versione ricorsiva della funzione separa (senza cambiare il prototipo).

Nello svolgimento dei punti precedenti, è possibile utilizzare (senza bisogno di implementarla) la funzione

```
void inserisci (lista *l, int val)
```

che inserisce l'elemento `val` in testa alla lista `l`.

Note. Si può supporre che le due liste vuote puntate dai parametri pari e dispari passati alla funzione contengano il valore NULL. Se necessario, è possibile implementare ulteriori funzioni di supporto da utilizzare all'interno della funzione separa. Non è importante l'ordine degli elementi inseriti nelle liste pari e dispari.

Soluzione

```
void separa_iter(lista a, lista *pari, lista* dispari)
{
      while (a!=NULL)
      {
            if (a->val % 2 == 0)
                  inserisci(pari,a->val);
            else
                  inserisci(dispari,a->val);

            a = a->next;
      }
}

void separa_ric(lista a, lista *pari, lista* dispari)
{

      if (a!=NULL)
      {
            if (a->val % 2 == 0)
                  inserisci(pari,a->val);
            else
                  inserisci(dispari,a->val);

            separa_ric(a->next,pari,dispari);
      }
}
```

Esercizio 2 (10 punti)

Un modo molto semplice per *criptare* un messaggio di testo è quello di modificare ogni carattere contenuto nel messaggio applicando la seguente funzione:

$$C' = (C+K)\%255$$

dove C è il carattere originale del messaggio, C' è il carattere *criptato*, K è un intero detto *chiave* che serve a *criptare* il messaggio, e % è l'operatore matematico che fornisce il resto della divisione intera.

Scrivere un programma che legge da tastiera un intero K, apre il file di testo "messaggio.txt" e ne riscrive il contenuto nel file "codice.txt" dopo aver applicato la funzione di *criptazione* descritta in precedenza, utilizzando la chiave K letta da tastiera.

Nota. Si ipotizzi che l'apertura dei due file di testo vada sempre a buon fine.

Soluzione

```
#include <stdio.h>

 int main()
 {
      FILE *in, *out;
      int k;
      char c;

      in = fopen("messaggio.txt","r");
      out = fopen("codice.txt","w");

      printf("Inserisci k: ");
      scanf("%d",&k);

      while( (c = fgetc(in)) !=EOF )
            fputc((c+k)%255,out);

      fclose(in);
      fclose(out);

      return 0;
 }
```

Esercizio 3 (5 punti)

Si consideri il seguente frammento di codice C e la sua implementazione in linguaggio macchina:

```
int n,i,x,sum=0;
scanf("%d", &n);

for (i=0; i<n; i++) {
      scanf("%d",&x);
      sum = sum + x;
}
printf("%d",sum);
```

```
01.   READ
02.   STORE 101
03.   LOAD= 0
04.   STORE 102
05.   READ
06.   ADD 102
07.   STORE 102
08.   LOAD 101
09.   SUB= 1
10.   STORE 101
11.   BNE 5
12.   LOAD 102
13.   WRITE
14.   END
```

A. L'implementazione in linguaggio macchina è corretta? Giustificare la risposta data.
B. In caso la risposta precedente non sia affermativa proporre una possibile correzione dell'implementazione.

Soluzione

A. L'implementazione non è corretta: se l'utente inserisce 0 (o un numero negativo) come valore di n, l'implementazione in linguaggio macchina continua a leggere un valore dal nastro di input, decrementa n e ricomincia da capo (dal momento che il contatore, memorizzato nella cella 101, sarà sempre diversa da zero).
B.

```
01.   READ
02.   STORE 101
03.   LOAD= 0
04.   STORE 102
05.   LOAD 101
06.   SUB= 1
07.   STORE 101
08.   BL 13
09.   READ
10.   ADD 102
11.   STORE 102
12.   BR 5
13.   LOAD 102
14.   WRITE
15.   END
```

Esercizio 4 (3 punti)

Si considerino i seguenti numeri A e B in virgola mobile rappresentati con la codifica IEEE 754-1985 a 32 bit:

A
S:0 E:10000000 M:11100000000000000000001

B
S:1 E:01111111 M:11100000000000000000001

Calcolare il risultato (in base 10) dell'operazione A/B. Giustificare la risposta (eventualmente riportando i calcoli effettuati).

Soluzione

A/B = -2

Infatti A e B hanno la stessa mantissa (M):

A: $M*2^{EA}$
B: $-M*2^{EB}$

Dato che EA=128 ed EB=127 (EA-1) → $A/B = -2^{EA}/2^{EA-1} = -2$

Esercizio 5 (4 punti)

Dichiarare in C il tipo smartphone, in grado di rappresentare le seguenti informazioni: nome modello (una sequenza di 15 caratteri), quantità di memoria (espressa in GByte), tipo di sistema operativo (che può avere uno dei seguenti valori: Android, iOS, Win, Altro) e un insieme di applicazioni installate (il numero di applicazioni installate è variabile ma non può essere maggiore di 500). Per ogni applicazione installata, sarà inoltre necessario rappresentare il nome dell'applicazione (50 caratteri massimo), la sua versione (un numero con parte frazionaria) e la quantità di memoria occupata (in Byte).

Soluzione

```
typedef struct
{
      int mem;
      char nome[51];
      float ver;
} app;

typedef enum {android, ios, win, altro} os;

typedef struct
{
      char modello[16];
      int mem;
      os myos;
      int napp;
      app apps[500];

} smartphone;
```