	Politecnico di Milano Facoltà di Ingegneria Industriale **FONDAMENTI DI INFORMATICA** Seconda prova in itinere – 1 Febbraio 2016 Pre-appello Invernale – 1 Febbraio 2016		COGNOME E NOME
	RIGA	COLONNA	MATRICOLA
			Spazio riservato ai docenti

Selezionare una delle seguenti opzioni:

☐ Seconda prova in itinere – Esercizi 2, 3, 4, 6

☐ Pre-appello Invernale – Esercizi 1, 3, 4, 5

- Il presente plico contiene 6 esercizi e **deve essere debitamente compilato con cognome e nome, numero di matricola**.
- Il tempo a disposizione è di 1 ore e 30 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 – SOLO Pre-Appello

Il file "`piani.txt`" contiene i dati riguardanti i *piani tariffari* di diversi operatori telefonici; ciascuna riga del file ha la seguente struttura:

```
<nome_operatore>      <canone>   <minuti>   <dati>
```

dove `nome_operatore` è il nome dell'operatore (senza spazi al suo interno), `canone` è il costo mensile in euro, `minuti` è il numero di minuti di traffico voce inclusi al mese e `dati` è il numero di GB di traffico dati inclusi al mese.

A. Dichiarare uno o più tipi di dato necessari a rappresentare in memoria una *lista dinamica* di *piani tariffari* (secondo quanto descritto in precedenza).
B. Implementare un programma C che legga il file "piani.txt" e ne salvi il contenuto in una *lista dinamica* (usando il tipo di dato dichiarato al punto precedente) in modo tale che i piani tariffari contenuti nella lista siano ordinati (in ordine crescente) sulla base del numero di minuti di traffico voce inclusi al mese.

Nota. Si consiglia di implementare una o più funzioni di supporto per gestire adeguatamente la lista dinamica.

Soluzione
A e B)

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

typedef struct
{
      char op[50];
      float canone;
      int minuti;
      int dati;
} piano;

struct nodo
{
      piano info;
      struct nodo *next;
};

typedef struct nodo *lista;

void inserisci (lista *l, piano info);

int main()
{
      FILE *f;
      piano p;
      lista l = NULL;

      f = fopen("piani.txt","r");
      while (feof(f)==0)
      {
            fscanf(f,"%s",p.op);
            fscanf(f,"%f",&p.canone);
            fscanf(f,"%d",&p.minuti);
            fscanf(f,"%d",&p.dati);
            inserisci (&l,p);
      }

      fclose(f);
      return 0;
}
```

```
void inserisci (lista *l, piano info)
{
      lista cur, pre, p;
      cur=*l;
      pre = NULL;

      while(cur!=NULL && info.dati > cur->info.dati)
      {
            pre = cur;
            cur = cur->next;
      }

      p = malloc(sizeof(struct nodo));
      p->info = info;
      p->next = cur;

      if (pre!=NULL)
            pre->next = p;
      else
            *l = p;
}
```

Esercizio 2 – SOLO Seconda Prova in Itinere

Si consideri una lista dinamica di interi definita come:

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

typedef struct nodo *lista;
```

Implementare una funzione `somma` che riceve in ingresso due liste di interi **a** e **b**, e restituisce una nuova lista di interi **c**, i cui elementi sono la somma degli elementi di **a** e di **b** in posizione corrispondente. Nel caso **a** e **b**, non abbiano la stessa lunghezza, gli elementi in eccesso (nella lista **a** o nella lista **b**) verranno copiati in **c** senza essere sommati a nulla.

Dichiarare il prototipo della funzione `somma` nel modo che si ritiene più opportuno e fornirne una implementazione, utilizzando (se necessario) la seguente funzione (di cui non è necessario fornire un'implementazione):

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

che aggiunge **in coda** alla alla lista `l` un elemento di valore `val`.

Esempio
Siano **a:** `3 -> 6 -> 8` e **b:** `4 -> 7 -> 4 -> 11 -> 15`
`somma (a,b)` sarà **c:** `7 -> 13 -> 12 -> 11 -> 15`

Soluzione

```
lista somma(lista a, lista b)
{
      lista sum = NULL, p;
      while (a!=NULL && b!=NULL)
      {
            inserisci (&sum,a->val+b->val);
            a = a->next;
            b = b->next;
      }

      if (a!=NULL)
            p = a;
      else
            p = b;

      while (p!=NULL)
      {
            inserisci (&sum,p->val);
            p = p->next;
      }

      return sum;
}
```

Esercizio 3

Implementare una funzione **ricorsiva** in C che riceve in ingresso un array **a** di interi e restituisce 1 se ogni elemento di a è maggiore od uguale dei precedenti, altrimenti restituisce 0.

Esempi: se `a[] = {2,4,5,28,90}`, la funzione restituisce 1; se `a[] = {5,16,2,82,99}`, la funzione restituisce 0.

Nota: La funzione dovrà avere l'array **a** come parametro in ingresso; è possibile introdurre ulteriori parametri di ingresso se necessari alla corretta implementazione della funzione

Soluzione

```
int crescente(int a[], int n)
{
      if (n==1)
            return 1;
      else
            return (a[0]<=a[1]) && crescente(a+1,n-1);
}
```

Esercizio 4

Utilizzando il linguaggio assembly introdotto a lezione, implementare un programma che continua a ripetere le seguenti due operazioni, (i) legge dal nastro di ingresso due numeri interi e (ii) ne esegue la somma finché la somma dei due numeri letti è maggiore di zero; non appena tale condizione non si verifica più, il programma scrive sul nastro di uscita la sequenza di tutte le somme che ha calcolato dall'inizio dell'esecuzione (ad eccezione dell'ultima somma che è risultata minore o uguale a zero).

Specifica in C del programma da implementare (ipotizzando stdin come nastro di ingresso e stdout come nastro di uscita):

```
do
{
 scanf("%d",&x);
 scanf("%d",&y);
 if (x+y >0)
 {
   a[k] = x+y;
   k++;
 }
} while (x+y > 0);

for (i=0; i<k; i++)
 printf("%d ", a[i]);
```

Soluzione

```
1.  LOAD= 110
2.  STORE 102
3.  READ
4.  STORE 101
5.  READ
6.  ADD 101
7.  BLE 13
8.  STORE@ 102
9.  LOAD= 1
10. ADD 102
11. STORE 102
12. BR 3
13. LOAD= 110
14. STORE 103
15. LOAD 102
16. SUB 103
17. BLE 24
18. LOAD@ 103
19. WRITE
20. LOAD= 1
21. ADD 103
22. STORE 103
23. BR 15
24. END
```

Esercizio 5 – SOLO Pre-Appello

A. Si determini la codifica del valore -126.6 secondo lo Standard IEEE 754-1985 a precisione singola, riportando i calcoli effettuati.
B. La codifica di -126.6, calcolata al punto A, è esatta? Giustificare la risposta.

Soluzione

A)

Parte intera
126 → 1111110

Parte frazionaria
0.6 * 2 = 1.2
0.2 * 2 = 0.4
0.4 * 2 = 0.8
0.8 * 2 = 1.6
0.6 …

0.100110011001…

=> 1111110.1001100110011001… => $1.11111010011001100110011001… \times 2^{6}$
=> Esponente: 6+127 = 133 → 10000101

=> S:1
=> E:10000101
=> M: 11111010011001100110011

B)

La codifica di -126.6 non è esatta dato che la mantissa è periodica.

Esercizio 6 – SOLO Seconda Prova in Itinere

Un sistema dispone di 128 Kbyte di memoria fisica e una memoria virtuale indirizzabile con paginazione, caratterizzata dai seguenti parametri: indirizzo virtuale di 18 bit e 128 pagine di memoria virtuale.
Rispondere alle seguenti domande giustificando le risposte:

A. Quale è la dimensione della memoria virtuale indirizzabile?
B. Quale è la struttura dell'indirizzo virtuale e di quello fisico, e la lunghezza dei campi che li costituiscono?
C. Un consulente afferma che raddoppiando la dimensione delle pagine di memoria e quindi dimezzando il numero di pagine di memoria virtuale si riducono possibili sprechi di memoria. Siete d'accordo con quest'affermazione? Argomentare in maniera adeguata la propria risposta.

Soluzione

A. 18 bit -> 2^18 byte di memoria virtuale -> 256 Kbyte di memoria virtuale indirizzabile
B. 128 (2^7) pagine virtuali -> 7 bit per NPV;
   128 Kbyte memoria fisica -> 17 bit indirizzo fisico.
   Quindi si ha,
   NPV: 7 bit, offset: 11 bit
   NPF: 6 bit, offset: 11 bit
C. L'affermazione è errata: raddoppiando la dimensione delle pagine di memoria, e quindi dimezzando il numero di pagine virtuali, aumenta la probabilità di dover assegnare ad un processo delle pagine di memoria solo parzialmente occupate e quindi aumentano i possibili sprechi di memoria.