	Politecnico di Milano Facoltà di Ingegneria Industriale **FONDAMENTI DI INFORMATICA** Seconda prova in itinere – 13 Febbraio 2017 Primo Appello (Laureandi) – 13 Febbraio 2017		COGNOME E NOME
	RIGA	COLONNA	MATRICOLA
			Spazio riservato ai docenti

Selezionare una delle seguenti opzioni:

☐ Seconda prova in itinere – Esercizi 2, 3, 4, 5

☐ Primo Appello (Laureandi) – Esercizi 1, 3, 4, 5

- 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 ore e 40 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 (5 punti) – SOLO Primo Appello

La società Juice Inc., per analizzare la produttività dei suoi 100 programmatori, ha raccolto per ciascuno di loro le seguenti informazioni: nome, cognome, età, le ore svolte e il numero di righe (in migliaia) di codice scritte (settimanali) nelle ultime 10 settimane. Un esempio dei dati raccolti per un dipendente è il seguente:

- Nome: Mario
- Cognome: Rossi
- Età: 35
- Ore settimanali nelle ultime 10 settimane: 40,40,42,43,45,46,50,38,40,40
- Righe di codice settimanali nelle ultime 10 settimane: 3.1, 3, 3.2, 3.3, 3.5, 3.1, 2.9, 2.7, 3.6, 3.3

A. Dichiarare un tipo di dati `programmatore` in C che contenga tutte le informazioni raccolte dalla Juice Inc. per ciascuno dei suoi programmatori.
B. Completare il seguente frammento di codice C in modo che vengano copiati nell'array `selezione` i dati dei programmatori che (i) hanno meno di 30 anni **e** (ii) nelle ultime 10 settimane, hanno prodotto un numero medio di righe di codice superiore o uguale al numero medio di righe di codice prodotte da tutti i programmatori della Juice Inc.; infine il frammento di codice deve anche stampare a video il numero di programmatori nella variabile `selezione`:

```
programmatore database[100], selezione[100];

carica(database,100); //carica in database tutti i dati dei programmatori

// inserire qui il frammento di codice
```

Note. La funzione `carica` riempe interamente la variabile database con i dati di **tutti** e 100 i programmatori; si ipotizzi inoltre che per ogni programmatore siano disponibili i dati di **tutte le 10 settimane precedenti**.

Soluzione

A.

```
typedef struct {
  char nome[50];
  char cognome[50];
  int eta;
  int ore[10];
  float linee[10];
} programmatore;
```

B.

```
int i,j;
float media=0,linee[100]={0};
for (i=0; i<100; i++) {
  for (j=0; j<10; j++){
    linee[i] += database[i].linee[j];
  }
  linee[i] /= 10;
  media += linee[i];
}
media /= 100;
j=0;
for (i=0; i<100; i++)
  if ( (linee[i] >= media) && (database[i].eta<30) ) {
    selezione[j] = database[i];
    j++;
  }
printf("%d\n",j);
```

Esercizio 2 (5 punti) – SOLO Seconda Prova in Itinere

Si consideri la seguente struttura dati dinamica per rappresentare una lista di interi:

```
struct nodo
{
      int dato;
      struct nodo *next;
      struct nodo *nextP;
      struct nodo *nextD;
};

typedef struct nodo *lista;
```

In cui `next` punta al successivo elemento della lista, `nextP` punta all'elemento successivo **pari** della lista, mentre `nextD` punta all'elemento successivo **dispari** della lista.

Implementare una funzione `inserisci` che consenta di **inserire in coda** un valore intero all'interno di una lista del tipo qui sopra descritto.

Nota. Si definisca il prototipo della funzione `inserisci` nel modo ritenuto più opportuno.

Soluzione

```
void inserisci(lista *l, int dato)
{
  lista p = malloc(sizeof(struct nodo));
  lista cur = *l;

  p->next=NULL;
  p->nextP=NULL;
  p->nextD=NULL;
  p->dato=dato;

  if (*l==NULL)
    *l = p;
  else
  {
    while (cur->next != NULL)
    {
      if (cur->nextP == NULL && dato%2 == 0)
        cur->nextP = p;
      if (cur->nextD == NULL && dato%2 == 1)
        cur->nextD = p;
      cur = cur->next;
    }
    cur->next = p;
    if (dato%2 == 0)
      cur->nextP = p;
    else
      cur->nextD = p;
  }
}
```

Esercizio 3 (5 punti)

La moltiplicazione fra due numeri interi x e y entrambi di n cifre (con $n=2^k$), si può calcolare ricorsivamente usando la seguente formula:

$$x \cdot y = 10^n \cdot x_s \cdot y_s + 10^{\frac{n}{2}} (x_d \cdot y_s + x_s \cdot y_d) + x_d \cdot y_d$$

dove x_s e y_s sono le metà (cioè le n/2 cifre) più significative di x e y, mentre x_d e y_d sono le metà (cioè le n/2 cifre) meno significative di x e y. Ad esempio $1234 \cdot 5678 = 10^n \cdot 12 \cdot 56 + 10^{\frac{n}{2}} (34 \cdot 56 + 12 \cdot 78) + 34 \cdot 78$

Quindi utilizzando la formula appena descritta, implementare la seguente funzione **ricorsiva** in C per effettuare il prodotto fra due interi con n cifre:

```
int prodotto (int x[], int y[], int n)
```

dove x e y sono due array che contengono le cifre dei numeri da moltiplicare ed n è il numero di cifre contenute negli array; la funzione dovrà ritornare il risultato del prodotto come una variabile intera.

Esempio

Per effettuare il prodotto 15*10, sarà possibile utilizzare la funzione `prodotto` nel modo seguente:

```
int x[2]={1,5}, y[2]={1,0}, ris;
ris = prodotto(x,y,2);
```

La variabile `ris`, conterrà alla fine il valore 150.

Suggerimenti: se x e y contengono una sola cifra, la loro moltiplicazione non richiede di essere ulteriormente scomposta; per calcolare le potenze del 10 (10^n e $10^{n/2}$) è possibile usare la funzione `pow(a,b)` (inclusa nella libreria `math.h`) che ritorna a^b; si ipotizzi che x e y abbiano entrambi sempre lo stesso numero di cifre.

Soluzione

```
int prod (int x[], int y[], int n)
{
  if (n==1)
    return x[0]*y[0];
  else
    return pow(10,n) * prod(x,y,n/2) + pow(10,n/2) * (prod(x,y+n/2,n/2) +
                                  prod(x+n/2,y,n/2)) + prod(x+n/2,y+n/2,n/2);
}
```

Esercizio 4 (4 punti)

Utilizzando il linguaggio macchina introdotto a lezione, implementare un programma che legge dal nastro di ingresso un numero intero **n** ed una sequenza di interi, terminata da uno zero; il programma quindi scriverà al contrario sul nastro di uscita gli ultimi **n** valori interi della sequenza letta (o l'intera sequenza, nel caso **n** sia maggiore della lunghezza della sequenza letta).

Specifica in C del programma da implementare (utilizzare il nastro di ingresso e il nastro di uscita per le operazioni di lettura e scrittura su stdin e stdout):

```
scanf("%d",&n);

i=0;
do
{
    scanf("%d",&v[i]);
    i++;
} while(v[i-1]!=0);
i = i-1;

if (n>i)
  n = i;

for (j = 0; j<n; j++)
  printf("%d ",v[i-1-j]);
```

Soluzione

```
01.     LOAD= 0
02.     STORE 101
03.     LOAD= 110
04.     STORE 103
05.     READ
06.     STORE 102
07.     READ
08.     BEQ 17
09.     STORE@ 103
10.     LOAD 103
11.     ADD= 1
12.     STORE 103
13.     LOAD 101
14.     ADD= 1
15.     STORE 101
16.     BR 7
17.     LOAD 101
18.     SUB 102
19.     BGE 22
20.     LOAD 101
21.     STORE 102
22.     LOAD 102
23.     BEQ 33
24.     LOAD 103
25.     SUB= 1
26.     STORE 103
27.     LOAD@ 103
28.     WRITE
29.     LOAD 102
30.     SUB= 1
31.     STORE 102
32.     BR 23
33.     END
```

Esercizio 5 (2 punti)

Un sistema dispone di 64 Mbyte di memoria fisica e una memoria virtuale indirizzabile con paginazione, caratterizzata dai seguenti parametri: indirizzo virtuale di 27 bit e pagine di memoria virtuale di dimensione 4Kbyte.

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 riducendo la dimensione delle pagine di memoria e quindi aumentando il numero di pagine di memoria virtuale potrebbero esserci maggiori sprechi di memoria. Siete d'accordo con quest'affermazione? Argomentare in maniera adeguata la propria risposta.

Soluzione

A. 27 bit -> 2^27 byte di memoria virtuale -> 128 Mbyte di memoria virtuale indirizzabile
B. Pagina di 4Kbyte = 2^12 byte → 12 bit offset
   64 Mbyte memoria fisica -> 26 bit indirizzo fisico
   27bit indirizzo fisico e 12 bit offset -> 15 bit NPV
   26 bit indirizzo virtuale e 12 bit offset -> 14 bit NPF
   Quindi si ha,
   NPV: 15 bit, offset: 12 bit
   NPF: 14 bit, offset: 12 bit
C. L'affermazione è errata: riducendo la dimensione delle pagine di memoria, e quindi aumentando il numero di pagine virtuali, si riduce la probabilità di dover assegnare ad un processo delle pagine di memoria solo parzialmente occupate e quindi diminuiscono i possibili sprechi di memoria.