	Politecnico di Milano Facoltà di Ingegneria Industriale **INFORMATICA B** Appello 19 Febbraio 2015		COGNOME E NOME
	RIGA	COLONNA	MATRICOLA
			Spazio riservato ai docenti

- Il presente plico contiene 4 esercizi e **deve essere debitamente compilato con cognome e nome, numero di matricola**.
- Il tempo a disposizione è di 2 ore.
- 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 (o ripudiate) 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 calcolatrici, telefoni o pc**. Chi tenti di farlo vedrà annullata la sua prova.
- È ammessa la consultazione di libri e appunti, purché con pacata discrezione e senza disturbare.
- **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**.
- L'esame orale è parte integrante dell'esame e deve essere realizzato almeno sufficientemente per il superamento dell'esame complessivo.

Esercizio 1 (8 punti)

Il kernel di un sistema operativo multiprogrammato deve gestire le code di processi pronti, in attesa e in esecuzione. Ogni processo è definito da un descrittore che contiene le seguenti informazioni:

- Identificativo di processo (PID), un valore numerico intero che identifica il processo in modo univoco;
- contenuto del Program Counter (codificato come intero);
- contenuto del registro A (codificato come intero);
- contenuto del registro B (codificato come intero);
- contenuto del registro INTR (codificato come intero);
- un vettore che, per ogni pagina virtuale, mantiene il numero della corrispondente pagina fisica; si assuma che ogni processo abbia una dimensione massima di 16 pagine virtuali, che i numeri di pagina virtuale possono essere utilizzati come indice per il vettore e che, nel caso in cui una pagina virtuale non è mappata su una pagina fisica, la cella corrispondente nel vettore conterrà il valore -1. Per esempio, se vettore[0] == 1 questo significa che la pagina virtuale 0 è mappata nella pagina fisica 1. Se vettore[5] == -1 e vettore[6] == 3, questo significa che la pagina virtuale 5 non è caricata nella memoria fisica mentre la pagina virtuale 6 è caricata nella pagina fisica 3;
- un intero che indica il numero effettivo di pagine virtuali occupate dal processo.

Per semplicità, si rappresenti il contenuto dei registri (e quindi anche del Program Counter) come degli interi.

A) Si definisca in linguaggio C il tipo **DescrittoreProcesso** che includa le informazioni elencate sopra.

B) Si definisca inoltre in linguaggio C il tipo **codaProcessi** in modo che contenga:
- Una lista di processi;
- il numero di elementi nella lista.

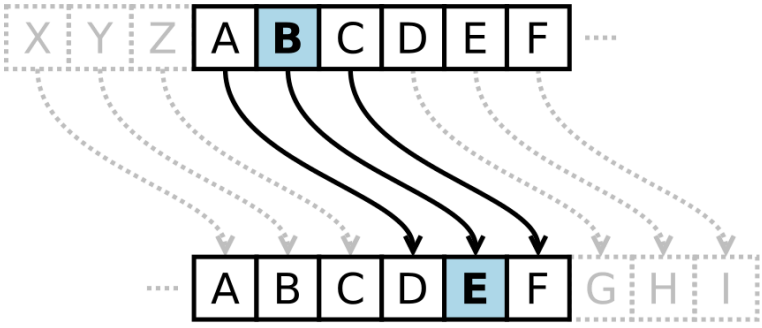
C) Si creino in linguaggio C tre variabili, **codaPronti**, **codaAttesa** e **codaEsecuzione**, che rappresentino rispettivamente le code dei processi pronti, in attesa e in esecuzione.

D) Si scriva un frammento di programma in linguaggio C che stampi il PID di tutti i processi in codaPronti che occupano più di 4 pagine fisiche. Si assuma che le code siano già state riempite da una porzione di programma di cui non viene richiesta l'implementazione.

Soluzione

```
#include <stdio.h>
#include <stdlib.h>
#define MAXMEMV 16
#define MAXPROCESSI 5

typedef struct
{
	int pid,pc,a,b,intr,countMemV;
	int memF[MAXMEMV];
} DescrittoreProcesso;

typedef struct
{
	DescrittoreProcesso processi[MAXPROCESSI];
	int countProcessi;
} CodaProcessi;

int main(int argc, char const *argv[])
{
	CodaProcessi codaPronti, codaAttesa, codaEsecuzione;
	int i, j, contFisiche;

	/* popolamento strutture dati … */

	printf("codaPronti – processi con piu` di 4 pagine fisiche: [");
	for(i=0;i<codaPronti.countProcessi;i++){
		countFisiche = 0;
		for(j=0;j<codaPronti.processi[i].countMemV;j++){
			if(codaPronti.processi[i].memF[j]!=-1)
				countFisiche = countFisiche + 1;
		}
		if (countFisiche > 4)
			printf(" PID[%d]",codaPronti.processi[i].pid);
	}
	printf("]\n");
}
```

Esercizio 2 (6 punti)
Un modo semplice di cifrare un messaggio di testo consiste nel sostituire ciascuna lettera nel messaggio con la lettera dell'alfabeto che si trova **k** posizioni dopo (considerando l'alfabeto circolare quando viene oltrepassata l'ultima lettera). Il valore **k** viene anche detto **chiave**, dal momento che è ciò che consente di cifrare/decifrare il messaggio di testo.
Ad esempio, la figura seguente mostra come vengono sostituite alcune lettere dell'alfabeto (X,Y,Z e da A a F) quando **k** è 3:



Implementare in linguaggio C un programma che legga da tastiera un **testo** (un array di caratteri di lunghezza non superiore a 1000) e la chiave **k**, controllando che **k** sia maggiore di 0 e minore di 26 (il numero di lettere dell'alfabeto). Quindi il programma dovrà stampare a video la versione cifrata del testo con chiave **k**. Nella versione cifrata, le lettere minuscole dovranno rimanere minuscole, le lettere maiuscole dovranno rimanere maiuscole e qualsiasi carattere che non sia una lettera dovrà rimanere invariato rispetto al testo originale.

Soluzione

```
#include <stdio.h>
#include <string.h>
#define NLETTERE 26

int main()
{
  char testo[1000],c;
  int k,i;

   printf("Inserire il testo: ");
   gets(testo);

  do
  {
    printf("Inserire la chiave: ");
    scanf("%d",&k);
  } while (k <= 0 || k >= 26);

  for (i = 0; i < strlen(testo); i++)
  {

    if ((testo[i] >= 'a' && testo[i] <= 'z' - k) || (testo[i] >= 'A' && testo[i] <= 'Z' - k))
       c = testo[i] + k;
    else if ((testo[i] > 'z' - k && testo[i] <= 'z') || (testo[i] > 'Z' - k && testo[i] <= 'Z'))
      c = testo[i] - NLETTERE + k;
    else
      c = testo[i];
    printf("%c",c);
  }
  printf("\n");
  return 0;
}
```

Esercizio 3 (6 punti)
Si consideri la seguente funzione di ordine superiore:

```
function r = fun(v,f)
  r = v(1);
  for i = 2:length(v)
      r = f(r,v(i));
  end
```

a) Qual è il valore ritornato dalla chiamata fun([8 9 10 9 6 0],@max)? Giustificare la risposta. max è una funzione offerta dalla libreria di Matlab. In particolare, se X e Y sono due scalari, max(X,Y) ritorna il maggiore tra i due valori.
b) Dire che cosa fa la funzione fun quando l'argomento v è un vettore numerico (di lunghezza pari almeno ad 1) ed f è @max
c) Implementare una versione ricorsiva della funzione fun che non richieda di utilizzare cicli (for o while). L'intestazione della funzione deve rimanere invariata.

Soluzione

a) La fun([8 9 10 9 6 0],@max) restituisce il valore 10.
b) La funzione restituisce l'elemento di maggior valore di v.
c)

```
function r = fun(v,f)
  n = length(v);
  if n == 1
      r = v;
  else
      r = f(v(1),fun(v(2:end),f));
  end
```

Esercizio 4 (8 punti)

Si scriva uno programma in Matlab per gestire le informazioni relative a memorie cache.

In particolare:

A) Volendo salvare in una variabile struct, `memoriaCache`, tutte le caratteristiche di una memoria cache neccesarie per determinarne il tempo medio di accesso, quali sono i campi che la variabile `memoriaCache` dovrà contenere? Fornire un esempio di inizializzazione della variabile `memoriaCache` (lo scopo dell'esempio è di illustrare quali campi deve contenere la variabile `memoriaCache`, non sono quindi importanti i valori assegnati ai campi della variabile).

B) Si scriva una funzione `sel = selezionaCache(memorie,tsoglia)`, dove `memorie` è un array i cui elementi sono struct e contengono gli stessi campi della variabile `memoriaCache` (introdotta nel punto precedente); `tsoglia` è uno scalare numerico che rappresenta un tempo medio di accesso minimo; la funzione restituisce un array `sel` che contiene soltanto gli elementi dell'array `memorie` il cui tempo medio di accesso è superiore a `tsoglia`.

C) Si scriva uno script che svolge le seguenti operazioni:

- richiede all'utente di inserire da tastiera 10 elementi dell'array `memorie`;
- richiede all'utente di inserire da tastiera la variabile `tsoglia`;
- chiama la funzione `selezionaCache` con argomenti `memorie` e `tsoglia` e ne stampa a schermo il risultato (stampa cioè tutti i campi delle memorie cache selezionate e ritornate dalla funzione)

Nota. La variabile `memoriaCache` (e quindi anche gli elementi dell'array `memorie`) non deve contenere alcun campo che rappresenti esplicitamente il tempo medio di accesso medio, ma solamente le caratteristiche delle memoria necessarie a calcolarlo.

Soluzione

A) Sono necessari tre campi numerici: hit rate, hit time e miss penalty

```
memoriaCache.hitRate = 0.9;
memoriaCache.hitTime = 20;
memoriaCache.missPenalty = 200;
```

B)

```
function sel = selezionaCache(memorie, tsoglia)
      HR = [memorie.hitRate];
      HT = [memorie.hitTime];
      MP = [memorie.missPenality];
      tempi = HR .* HT + (1 - HR) .* MP;
      indx = find(tempi > tsoglia);
      sel = memorie (indx);
```

C)

```
for ii = 1 : 10
      memorie(ii).hitRate = input('Inserire Hit Rate ');
      memorie(ii).hitTime = input('Inserire Hit Time ');
      memorie(ii).missPenality= input('Inserire MissPenality ');
end

tsoglia = input('Inserire tempo di soglia ');
sel = selezionaCache(memorie, tsoglia);

for m = sel
   disp(['HR: ', num2str(m.hitRate), …
      ' HT: ', num2str(m.hitTime), ' MP: ', num2str(m.missPenality)]);
end
```