	Politecnico di Milano Scuola di Ingegneria Industriale e dell'Informazione **INFORMATICA B** Appello 27 giugno 2016		COGNOME E NOME
			MATRICOLA
			Spazio riservato ai docenti

- Il presente plico contiene 3 esercizi e **deve essere debitamente compilato con cognome e nome, numero di matricola**.
- Il tempo a disposizione è di 1 ora 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 (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 (10 punti)

Uber2 vi ha incaricato di sviluppare in linguaggio C il loro sistema di gestione dei taxi all'interno di un'area urbana. In particolare, vi ha chiesto di progettare le strutture dati necessarie per immagazzinare le seguenti informazioni:

- **Dati relativi ai taxi**. Ogni taxi è identificato da un numero univoco ed è caratterizzato dal numero di passeggeri che può portare.
- **Dati relativi alle zone all'interno di una città**. Ogni zona è identificata da un numero univoco e dalla lista dei taxi che si trovano in quella zona, fino a un massimo di 50 taxi. Si tenga presente che il numero di taxi presenti in un certo momento in una zona può essere anche inferiore al massimo.
- **Dati relativi a tutte le città servite dal servizio taxi**. Ogni città è caratterizzata da un nome e dalle sue zone, fino a un massimo di 10. Si tenga presente che una città può avere anche meno di 10 zone. Uber2 gestisce fino a un massimo di 100 città.

A) Definire in linguaggio C tutte le strutture dati necessarie a rispondere alle richieste di Uber2.

B) Assumendo che **listaCitta** sia una variabile completamente popolata contenente la lista delle città servite da Uber2, sviluppare un frammento di programma in linguaggio C che:

- chieda all'utente di inserire: il nome di una città, il numero di una zona in quella città e il numero di passeggeri da trasportare
- stampi a schermo il codice numerico di un taxi che si trovi nella zona indicata e che abbia a disposizione un numero di posti maggiore o uguale al numero di passeggeri da trasportare.

Soluzione

A)

```
#define MAX_CITTA 100
#define MAX_ZONE 10
#define MAX_TAXI 50

typedef char stringa[30];

typedef struct {
   int numeroTaxi;
   int nPosti;
} taxi;

typedef struct {
   int numeroZona;
   Taxi listaTaxi[MAX_TAXI];
   int nTaxi;
} zona;

typedef struct {
   stringa nome;
   zona listaZone[MAX_ZONE];
   int nZone;
} citta;

citta listaCitta[MAX_CITTA];
```

B)

```
void main()
{
//…
  stringa citta;
  int nZona;
  int nPasseggeri;
  int i, j, k;

  citta listaCitta[MAX_CITTA];

//…
  printf("Inserisci il nome della citta': ");
  scanf("%s", citta);
  printf("Inserisci il numero della zona della citta': ");
  scanf("%d", &nZona);
  printf("Inserisci il numero di passeggeri: ");
  scanf("%d", &nPasseggeri);

  i = 0;
  while(i < MAX_CITTA && strcmp(listaCitta[i].nome, citta)!= 0)
    i++;
  if(i < MAX_CITTA) /* allora listaCitta[i] corrisponde alla citta` richiesta */
  {
    j = 0;
    while(j < listaCitta[i].nZone && listaCitta[i].listaZone[j].numeroZona != nZona)
      j++;
    if(j < listaCitta[i].nZone) /* allora listaCitta[i].listaZone[j] e` zona cercata */
    {
      k = 0;
      while(k < listaCitta[i].listaZone[j].nTaxi &&
            listaCitta[i].listaZone[j].listaTaxi[k].nPosti < nPasseggeri)
        k++;
      if(k < listaCitta[i].listaZone[j].nTaxi) /* allora
                  /* listaCitta[i].listaZone[j].listaTaxi[k] e` il taxi cercato */
         printf("Il taxi cercato e` %d\n",
                listaCitta[i].listaZone[j].listaTaxi[k].numeroTaxi);
      else printf("Taxi non trovato nella zona indicata\n");
    }
    else printf("Zona non trovata nella citta` indicata\n");
  }
  else printf("Citta` non trovata\n");

} /* chiude il main */
```

Esercizio 2 (10 punti)

A) Si vogliono costruire in Matlab opportune funzioni per gestire un vettore come se fosse una coda (si pensi alla coda di persone davanti allo sportello di un ufficio pubblico). In particolare, assumendo che l'elemento 1 del vettore corrisponda al primo della coda, sviluppare le seguenti funzioni:

function [nuovaCoda] = accoda(coda, valore): restituisce una nuova coda contenente tutti gli elementi di coda più il nuovo elemento valore, che viene inserito alla fine della nuova coda.

function [nuovaCoda, valore] = estraiPrimo(coda): estrae dalla coda il primo valore. Restituisce la nuova coda che non contiene più il primo valore, e il valore stesso. Per esempio, se coda = [10, 3, 4, 12, 17], dopo l'esecuzione della funzione si avrà che: nuovaCoda = [3, 4, 12, 17] e valore = 10.

function [lunghezza] = calcolaLunghezza(coda): restituisce lunghezza, che rappresenta il numero di elementi in coda.

B) Si considerino due vettori numerici che rappresentino due code, **codaNormale** e **codaPrioritaria**, ad esempio definite come:

```
codaNormale = [10, 3, 4, 12, 17, 13, 67, 45, 32, 22];
codaPrioritaria = [11, 5, 6, 8, 9, 15];
```

Si assuma che l'ordine degli elementi di una coda sia definito dall'indice della posizione degli elementi del vettore, che ciascun valore numerico nelle code corrisponda univocamente a una persona in coda agli sportelli di un ufficio pubblico, e che le persone i cui valori si trovano in codaPrioritaria debbano avere la precedenza rispetto a quelle che si trovano in codaNormale (per esempio perchè sono bambini o anziani).
Si ipotizzi che gli elementi di una coda siano estratti partendo dal primo valore del vettore che rappresenta la coda, e che la variabile **ultimoEstratto** indichi se l'ultimo valore estratto sia stato estratto da codaNormale (in questo caso ultimoEstratto assume il valore **0**) oppure da codaPrioritaria (in questo caso ultimoEstratto assume il valore **1**).

Si consideri quindi il seguente script Matlab:

```
codaNormale = input('Inserire la coda normale');
codaPrioritaria = input('Inserire la coda prioritaria');
ultimoEstratto = input('Inserire tipo di ultimo numero estratto (0 normale, 1 prioritario)');
```

Si completi lo script in modo che estragga un numero da una delle due code e ne stampi a video il valore, seguendo il seguente criterio per decidere da che coda estrarre il numero:

- se l'ultimo valore è stato estratto da codaNormale, allora il nuovo valore sarà estratto da codaPrioritaria.
- In caso contrario, se la lunghezza di codaPrioritaria è maggiore della lunghezza di codaNormale/3, allora estrae il valore da codaPrioritaria; altrimenti, estrae il valore da codaNormale.

Per lavorare con le due code si utilizzino le funzioni definite al punto precedente. Inoltre, lo script dovrà aggiornare anche il valore delle variabili codaNormale, codaPrioritaria e ultimoEstratto in modo coerente con quanto definito.

Soluzione

A)

```
function [nuovaCoda] = accoda(coda, valore)
  nuovaCoda = coda;
  nuovaCoda(end+1) = valore;
end

function [nuovaCoda, valore] = estraiPrimo(coda)
  nuovaCoda = coda;
  valore = nuovaCoda(1);
  nuovaCoda(1) = [];
end

function [lunghezza] = calcolaLunghezza(coda)
  lunghezza = length(coda);
end
```

B)

```
% Script che rimuove da una coda il prossimo elemento e lo stampa a video.
% Utilizza le variabili codaNormale, codaPrioritaria e ultimoEstratto, che
% devono essere definite prima della sua chiamata.

codaNormale = input('Inserire la coda normale');
codaPrioritaria = input('Inserire la coda prioritaria');
ultimoEstratto = input('Inserire tipo di ultimo numero estratto (0 normale, 1
prioritario)')

if ultimoEstratto == 0 || calcolaLunghezza(codaPrioritaria) >
calcolaLunghezza(codaNormale)/3
    [codaPrioritaria, v] = estraiPrimo(codaPrioritaria);
    ultimoEstratto = 1;
else
    [codaNormale, v] = estraiPrimo(codaNormale);
    ultimoEstratto = 0;
end

disp(['Numero estratto: ' num2str(v)]);
```

Esercizio 3 (6 punti)

Un sistema dispone di 256 Kbyte di memoria fisica e una memoria virtuale indirizzabile con paginazione, caratterizzata dai seguenti parametri: indirizzo virtuale di 20 bit e 256 pagine di memoria con parole di memoria di 1 byte.

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 aumentando la dimensione delle pagine di memoria e quindi riducendo il numero di pagine di memoria, aumentano i possibili sprechi di memoria. Siete d'accordo con questa affermazione? Argomentare in maniera adeguata la propria risposta.

Soluzione

A) 20 bit -> 2^20 byte di memoria virtuale -> 1 Mbyte

B)

256 (2^8) pagine virtuali -> 8 bit per NPV
256 kByte memoria fisica -> 18 bit indirizzo fisico

Quindi si ha,

NPV: 8 bit, offset: 12 bit
NPF: 6 bit, offset: 12 bit

C) L'affermazione è corretta: aumentando la dimensione delle pagine di memoria, e quindi riducendo il numero di pagine di memoria, si ha meno controllo sull'utilizzo della memoria, dovendo assegnare blocchi contigui (una pagina) più grandi a un unico processo, aumenta la probabilità di sprecare spazi durante il funzionamento del sistema.