Studente (Cognome Nome): _____ Matricola:_____

Corso di Informatica
Corso di Laurea in Ingegneria Gestionale
a.a. 2006-07
Primo Scritto – 12 Gennaio 2007

Si noti che le soluzioni ai quesiti saranno considerate valide solo se il materiale consegnato includera' anche lo svolgimento. Tale foglio deve essere consegnato insieme allo svolgimento.

Quesito 1

Un magazino è rappresentato da un array di pezzi, identificati dal nome e dal prezzo di ciascun elemento. Un preventivo richiesto da un cliente è rappresentato da un array di pezzi, identificati dal nome e dalla quantità richiesta.

a) Scrivere un metodo Java che calcoli il costo di un preventivo. Il metodo deve avere il seguente prototipo:

```
public static double costo(String [] mag_nome, double [] mag_prezzo,
String [] prev_nome, int [] prev_quantità)
```

Prerequisiti del metodo: gli array non sono null, i prezzi e le quantità sono >=0, tutti i pezzi del preventivo sono presenti in magazino.

b) Scrivere un metodo Java che ordini in modo decrescente un preventivo rispetto al costo dei singoli elementi (costo=quantità X prezzo).. Il metodo deve avere il seguente prototipo:

```
public static void ordina(String [] mag_nome, double [] mag_prezzo,
String [] prev_nome, int [] prev_quantità)
```

Prerequisiti del metodo: gli array non sono null, i prezzi e le quantità sono >=0, tutti i pezzi del preventivo sono presenti in magazino. Esempio:

se il contenuto del magazino è il seguente,

Nome	prezzo
Cinghia	50
Cerchione	350
Spazzola	10
Freni	300

e il preventivo è il seguente

Nome	Quantità
Cinghia	6
Spazzola	1
Cerchione	4

L'invocazione del metodo di cui sopra riorganizza il preventivo nel seguente modo:

Nome	Quantità
Cerchione	4
Cinghia	6
Spazzola	1

c) Scrivere un metodo Java che individua gli elementi non validi all'interno di un preventivo. Il metodo deve avere il seguente prototipo (il preventivo si suppone ordinato in modo decrescente rispetto al costo degli elementi, come indicato dal metodo b):

```
public static boolean [] validi(String [] mag_nome, double []
mag_prezzo, String [] prev_nome, int [] prev_quantità)
```

un preventivo è valido se il costo del preventivo è <= 1000 Euro. Se il preventivo è valido, il metodo deve restituire un vettore le cui componenti sono tutte true. Se il preventivo non è valido, ossia il costo è > 1000 Euro, sono considerati validi gli elementi a costo minore, (costo = prezzoXquantità), la cui somma dei costi è <= 1000 Euro.

Esempio: se il contenuto del magazino è il seguente:

Nome	prezzo
Cinghia	50
Cerchione	350
Spazzola	10
Freni	300

E l'ordine è il seguente:

Nome	Quantità
Cerchione	4
Cinghia	6
Spazzola	1

Il metodo restituisce l'array {false, true, true}.

Prerequisiti del metodo: gli array non sono null, i prezzi e le quantità sono >=0, tutti i titoli del preventivo sono presenti in magazino, il preventivo si suppone ordinato in modo decrescente rispetto al costo degli elementi, come indicato dal metodo b

Quesito 2

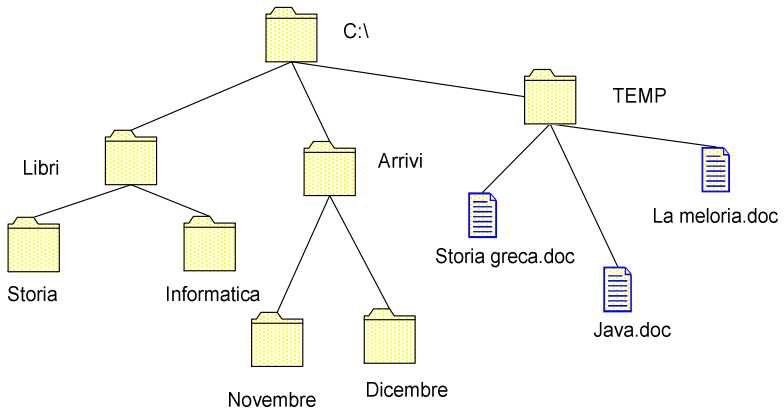
a) Rappresentare in C2 su 8 bit i seguenti numeri:

```
-127
+28
```

b) Calcolare la somma di tali numeri su 8 bit se rappresentabile.

Quesito 3

Si consideri il seguente file system, di cui esistono solo la directory C:\ e TEMP, con i relativi file contenuti (non directory). Si tenga presente che nel mese di novembre è arrivato "java.doc", nel mese di dicembre "La meloria.doc".



a) Impartire i comandi per creare le directory mancanti, supponendo che la directory corrente sia C:\.

b) Impartire la sequenza di comandi per spostare i file da C:\TEMP nelle relative cartelle/directory, eventualmente creando link, utilizzando solo path-name relativi. Si suppone che la directory corrente sia C:\TEMP. É possibile navigare fra le directory utilizzando il comando cd.

Soluzione della parte Java

```
/**
 * Soluzione scritto del 12 gennaio 07
 *
 * Magazino
 *
 * @author Nicola Serreli
 */
public class PrimoScritto_B {

	/** Funzione che calcola il costo di ciascun elemento del preventivo */
	public static double[] prev_costi(String[] mag_titolo, double[] mag_prezzo,
			String[] prev_titolo, int[] prev_quantita) {

		int i, j;
		double[] prev_costi = new double[prev_titolo.length];

		/* calcolo i vari costi (simile alla funzione costo) */
		for (i = 0; i < prev_titolo.length; i++) {

			/*
			 * Per ogni pezzo in preventivo :
			 * 1) lo cerco in magazino.
			 * 2) conoscendo il prezzo, calcolo il suo costo.
			 */
			for (j = 0; j < mag_titolo.length; j++) {
				if (prev_titolo[i].equals(mag_titolo[j])) {
					// Nota: l'indice dei costi e' lo stesso del preventivo
					prev_costi[i] = prev_quantita[i] * mag_prezzo[j];
				}
			}
		}
		return prev_costi;
	}



	public static double costo(String[] mag_titolo, double[] mag_prezzo,
			String[] prev_titolo, int[] prev_quantita) {

		double costo_totale = 0;

		/* calcolo prima i costi dei singoli elementi e poi sommo */
		double[] prev_costi = prev_costi(mag_titolo, mag_prezzo,
				prev_titolo, prev_quantita);

		int i = 0;
		for (i = 0; i < prev_costi.length; i++) {
			costo_totale += prev_costi[i];
		}

		return costo_totale;
	}

	/** Scambia due elementi appartenenti ad un vettore di double */
	public static void scambio_costi(double[] prev_costi, int i, int j) {
		double tmp_c = prev_costi[i];
		prev_costi[i] = prev_costi[j];
		prev_costi[j] = tmp_c;
	}

	/** Scambia due elementi appartenenti ad un vettore di int */
	public static void scambio_quantita(int[] prev_quantita, int i, int j) {
		int tmp_i = prev_quantita[i];
		prev_quantita[i] = prev_quantita[j];
		prev_quantita[j] = tmp_i;
	}

	/** Scambia due elementi appartenenti ad un vettore di String */
	public static void scambio_titoli(String[] prev_titolo, int i, int j) {
		String tmp_t = prev_titolo[i];
		prev_titolo[i] = prev_titolo[j];
		prev_titolo[j] = tmp_t;
	}

	public static void ordina(String[] mag_titolo, double[] mag_prezzo,
			String[] prev_titolo, int[] prev_quantita) {
```

```
        int i = 0;
        int j = 0;

        /*
         * Per ordinare il preventivo, e' utile avere un vettore contenente i
         * costi dei singoli libri.
         *
         * Notare che questo vettore e' "parallelo" ai vettori del preventivo,
         * quindi nell'ordinare dovro' fare 3 scambi.
         */
        double[] prev_costi = prev_costi(mag_titolo, mag_prezzo,
                        prev_titolo, prev_quantita);

        /* ordino rispetto al nuovo vettore */
        for (i = 0; i < (prev_costi.length - 1); i++) {

                for (j = i + 1; j < prev_costi.length; j++) {
                        if (prev_costi[i] < prev_costi[j]) {

                                // scambi
                                scambio_costi(prev_costi, i, j);
                                scambio_titoli(prev_titolo, i, j);
                                scambio_quantita(prev_quantita, i, j);

                        }
                }

        }
}


public static boolean[] validi(String[] mag_titolo, double[] mag_prezzo,
                String[] prev_titolo, int[] prev_quantita) {

        /*
         * Per risolvere questo quesito ho bisogno di conoscere il costo dei
         * singoli elementi e di sommare gli elementi a costo minore, finche' il
         * costo totale rimane <= 1000. Poiche' l'ordine e' decrescente, gli
         * elementi a costo minore stanno nelle ultime posizioni.
         */

        boolean[] prev_validi = new boolean[prev_titolo.length];


        // per calcolare il costo dei singoli elementi sfrutto la funzione prev_costi
        double[] prev_costi = prev_costi(mag_titolo, mag_prezzo,
                        prev_titolo, prev_quantita);
        double costo_totale = 0;


        int i = 0;
        // NOTA : inizio dall'ultimo elemento
        for (i = prev_costi.length-1; i >=0 ; i--) {
                costo_totale += prev_costi[i];

                // controllo se l'elemento e' valido, cioe' se non fa salire
                // il costo totale oltre i 1000
                if (costo_totale <= 1000) {
                        prev_validi[i] = true;
                } else {
                        prev_validi[i] = false;
                }
        }

        return prev_validi;
}

// soluzione alternativa
public static boolean[] validi_2(String[] mag_titolo, double[] mag_prezzo,
                String[] prev_titolo, int[] prev_quantita) {

        /*
         * Per risolvere questo quesito ho bisogno di conoscere il costo dei
         * singoli elementi e di sommare gli elementi a costo minore, finche' il
         * costo totale rimane <= 1000. Poiche' l'ordine e' decrescente, gli
         * elementi a costo minore stanno nelle ultime posizioni.
         */

        boolean[] prev_validi = new boolean[prev_titolo.length];
```

```
		// per calcolare il costo dei singoli elementi sfrutto la funzione prev_costi
		double[] prev_costi = prev_costi(mag_titolo, mag_prezzo,
				prev_titolo, prev_quantita);

		// il costo totale puo' essere calcolato con la funzione costo
		double costo_totale = costo(mag_titolo, mag_prezzo,
				prev_titolo, prev_quantita);


		int i = 0;
		// NOTA : inizio dal primo elemento e li sottraggo al preventivo,
		// finche' il totale non e' <= 1000
		while (costo_totale>1000) {
			costo_totale -= prev_costi[i];
			prev_validi[i] = false;
			i++;
		}

		// tutti gli altri elementi saranno validi
		for (; i < prev_costi.length ; i++) {
			prev_validi[i] = true;
		}

		return prev_validi;
	}

	public static boolean[] validi_3(String[] mag_titolo, double[] mag_prezzo,
			String[] prev_titolo, int[] prev_quantita) {

		/*
		 * Per risolvere questo quesito ho bisogno di conoscere il costo dei
		 * singoli elementi e di sommare gli elementi a costo minore, finche' il
		 * costo totale rimane <= 1000. Poiche' l'ordine e' decrescente, gli
		 * elementi a costo minore stanno nelle ultime posizioni.
		 */

		boolean[] prev_validi = new boolean[prev_titolo.length];

		// per calcolare il costo dei singoli elementi sfrutto la funzione prev_costi
		double[] prev_costi = prev_costi(mag_titolo, mag_prezzo,
				prev_titolo, prev_quantita);

		// il costo totale puo' essere calcolato con la funzione costo
		double costo_totale = costo(mag_titolo, mag_prezzo,
				prev_titolo, prev_quantita);


		// elimino i vari elementi, partendo dal primo, controllando di volta in
		// volta il costo totale.
		for (int i = 0; i < prev_costi.length ; i++) {
			if (costo_totale<=1000) {
				prev_validi[i] = true;
			} else {
				prev_validi[i] = false;
			}

			// in ogni caso, elimino l'elemento corrente
			costo_totale -= prev_costi[i];
		}

		return prev_validi;
	}

	// main di test, non richiesto dal quesito

	public static void main(String[] args) {
		String[] mag_titolo = new String[] {"Cinghia", "Cerchione", "Spazzola", "Freni"};
		double[] mag_prezzo = new double[] {50, 350, 10, 300};
		String[] prev_titolo = new String[] {"Cinghia", "Spazzola", "Cerchione"};
		int[] prev_quantita = new int[] {6, 1, 4};

		// costo
		double costo = costo(mag_titolo, mag_prezzo, prev_titolo, prev_quantita);
		System.out.println("Il costo totale e' " + costo);

		// ordinamento
		ordina(mag_titolo, mag_prezzo, prev_titolo, prev_quantita);
		System.out.println("Ordinato ");
		for (int i=0; i<prev_titolo.length; i++) {
```

```
                        System.out.println("  " + prev_titolo[i] + "\t"+prev_quantita[i]);
                }

                // validita'
                boolean[] valid_1 = validi(mag_titolo, mag_prezzo, prev_titolo, prev_quantita);
                System.out.print("Validita' (1) =");
                for (int i=0; i<valid_1.length; i++) {
                        System.out.print("  " + valid_1[i]);
                }
                System.out.println();

                boolean[] valid_2 = validi(mag_titolo, mag_prezzo, prev_titolo, prev_quantita);
                System.out.print("Validita' (2) =");
                for (int i=0; i<valid_2.length; i++) {
                        System.out.print("  " + valid_2[i]);
                }
                System.out.println();

                boolean[] valid_3 = validi(mag_titolo, mag_prezzo, prev_titolo, prev_quantita);
                System.out.print("Validita' (3) =");
                for (int i=0; i<valid_3.length; i++) {
                        System.out.print("  " + valid_3[i]);
                }
                System.out.println();
        }
}
```