**PROGRAMMAZIONE ORIENTATA AGLI OGGETTI (DM 270)**

**Appello del 28-02-2013 – Allievi Informatici**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Cognome:** |  | **Nome:** |  | **Matr:** |  | **Durata**: | **3 ore** |

**Esercizio 1.** Si considerano numeri interi rappresentati per complementi a 2 ad 11 bit. a) Qual è l’intervallo di rappresentazione? b) Dati i numeri n1=-35510, n2=67910, dire se sono rappresentabili e se sì fornire il loro complemento a 2 ad 11 bit. c) Determinare e mostrare il risultato delle operazioni: n1-n2, n1+n2 avendo cura di identificare e motivare (punto di vista della macchina) ogni situazione di traboccamento. d) Dato il numero complementi a 2 ad 11 bit n3=101110101002, specificare il suo valore in decimale.

**Esercizio 2.** È assegnato il numero reale r=-45.7710. Determinare e mostrare la sua rappresentazione 32 bit IEEE 754 a singola precisione. Riportare, infine, anche la versione esadecimale del risultato ottenuto.

**Esercizio 3.** Scrivere un programma Java che legga da input tre vettori di 5 interi a, b e c e scriva in uscita 1 se per ogni indice valido risulta: c[i]=b[i]2%a[i], 0 altrimenti. Quindi derivare sistematicamente e riportare una corrispondente versione RASP dell’algoritmo.

**Esercizio 4**. Data la sequenza di input 15 -4 12 -3 17 -2 0 20 16, mostrare (i) il contenuto del corrispondente albero binario di ricerca (ii) il contenuto del corrispodente heap. Nell’ipotesi di (a) eliminare la radice dell’albero, come cambia l’albero binario di ricerca ? (b) eliminare il minimo dell’heap, come cambia l’heap?

**Esercizio 5.** Su un file di tipo testo sono riportate, una per linea, le informazioni di una lista di persone, come dal seguente formato:

PERSONA=cognome *spazi* nome CODICE=intero non negativo

con la convenzione che i campi PERSONA e CODICE possono comparire in qualunque ordine. Si deve scrivere un programma interattivo che riconosca i seguenti comandi (indipendentemente dal caso delle lettere):

**INP** nome-file INVIO

**OUT** nome-file INVIO

**SBN** INVIO

**SBC** INVIO

**SHW** INVIO

**END** INVIO

INP consente di fornire il nome esterno di un file testo contenente le informazioni su una collezione di persone. Il comando verifica la consistenza del file e scrive su output “File inconsistente!” o “File caricato”. Un file è consistente se in esso mai succede che uno stesso nominativo o uno stesso codice compaiano più volte, e se le informazioni sono sempre ben formate.

OUT permette di fornire il nome esterno di un file su cui memorizzare la lista ordinata dei nominativi.

SBN richiede di ordinare la lista per nominativi per cognomi crescenti alfabeticamente ed a parità di cognome per nomi crescenti alfabeticamente.

SBC richiede di ordinare la lista per codici crescenti.

SHW richiede di visualizzare la lista ordinata ottenuta a seguito dell’ultimo comando SBN o SBC. Il comando OUT memorizza la lista ordinata su file esterno.

END permette di terminare il programma.

Il programma va modularizzato organizzandolo in classi come segue.

Classe **Nominativo** (immutabile). È provvista del confronto naturale e memorizza le informazioni di un nominativo. Il confronto naturale si riferisce alla successione alfabetica crescente per cognome ed a parità di cognome alla successione alfabetica crescente per nome. La classe deve essere dotata degli opportuni metodi accessori e certamente dei metodi equals, hashCode e toString.

Interfaccia **Lista** (iterabile). Modella una lista di nominativi ed espone metodi per aggiungere/rimuovere/cercare un nominativo, e per ordinare la lista. Sono previsti due metodi di ordinamento: sort() e sort( comparatore-di-nominativi ). Il primo realizza l’ordinamento naturale dei nominativi. Il secondo riceve un oggetto comparatore e lo utilizza per l’ordinamento. In un caso il comparatore può stabilire l’ordinamento per codice dei nominativi.

Classe astratta **ListaAstratta**. Implementa Lista e concretizza quanti più metodi è possibile e certamente i metodi equals, toString ed hashCode.

Classe **ListaConcatenata**. Estende ListaAstratta e memorizza i nominativi su una lista semplice a puntatori espliciti.

Classe **ListaSet**. Estende ListaAstratta e memorizza i nominativi su una struttura dati di tipo Set del collection framework di Java.

Classe **ListaMap**. Estende ListaAstratta e memorizza i nominativi su una struttura dati di tipo Map del collection framework di Java.

**Esercizio 6.** Mostrare una realizzazione ricorsiva del metodo toString della classe ListaConcatenata di cui all’esercizio 5.

**Bozza soluzioni – L. Nigro**

**Esercizio 1.**

a) [-2n-1, 2n-1-1]=[-1024,1023]

b) I due numeri sono rappresentabili. Segue il loro c2 ad n=11 bit.

n1=-355 => 355=00101100011 => c2(n1)=11010011101

n2=679 => 679=01010100111 => c2(n2)=01010100111

c)

n1-n2=n1+c2(-n2) =>

11010011101+

10101011001=

--------------------

1 01111110110 Esiste overflow perché da due negativi non può derivare un positivo. Il bit di riporto è ignorabile.

n1+n2=>

11010011101+

01010100111=

-------------------

1 00101000100 Nessun trabocco. Il bit di riporto è ignorabile. Il risultato corretto è negli 11 bit: 32410.

d) n3=10111010100 => n3 è negativo (bit segno 1). Il suo valore si ottiene banalmente facendo il c2 del c2:

01000101100 => n3=-55610

**Esercizio 2.**

r=-45.7710 =>

S=1

45=101101

.77≈.1100010100011110101110

mantissa=101101.1100010100011110101110 = 1.011011100010100011110101110 x 25 =>

E=127+5=132=10000100

F=01101110001010001111010 + 1 in quanto il bit di drop è 1 =>

F=01101110001010001111011 =>

1|10000100|01101110001010001111011

e in esadecimale: C237147B

**Esercizio 3.**

;int []a=new int[5], []b=new int[5], []c=new int[5];

;leggi a, b, c

;for( int i=0; i<a.length; ++i )

; if( c[i]!=(b[i]\*b[i])%a[i] ) return 0;

;return 1;

;

A: RES 5

B: RES 5

C: RES 5

LOAD# 5

STORE LENGTH

LOAD# 0

STORE I

LA: LOAD I

SUB LENGTH

JZ FLA

LOAD# A

ADD I

STORE AI

READ@ AI

LOAD I

ADD# 1

STORE I

JUMP LA

FLA:

LOAD# 0

STORE I

LB: LOAD I

SUB LENGTH

JZ FLB

LOAD# B

ADD I

STORE BI

READ@ BI

LOAD I

ADD# 1

STORE I

JUMP LB

FLB:

LOAD# 0

STORE I

LC: LOAD I

SUB LENGTH

JZ FLC

LOAD# C

ADD I

STORE CI

READ@ CI

LOAD I

ADD# 1

STORE I

JUMP LC

FLC:

LOAD# 0

STORE I

FOR:

LOAD I

SUB LENGTH

JZ ROF

LOAD# A

ADD I

STORE AI

LOAD# B

ADD I

STORE BI

LOAD@ BI

MUL@ BI

STORE BI2

DIV@ AI

STORE Q

MUL@ AI

STORE TMP

LOAD BI2

SUB TMP

STORE RESTO

LOAD# C

ADD I

STORE CI

LOAD@ CI

SUB RESTO

JZ AVANTI

WRITE# 0

JUMP FINE

AVANTI:

LOAD I

ADD# 1

STORE I

JUMP FOR

ROF:

WRITE# 1

FINE:

HALT

**Esercizio 4.**

ABR)

Utilizzando la visita anticipata e indicando con **.** un sotto albero vuoto, l’ABR originale è:

15 -4 **.** 12 -3 **.** -2 **.** 0 **...** 17 16 **..** 20 **..**

Rimuovendo il nodo radice (15) e promuovendo il minimo nel sotto albero destro della radice si ha:

16 -4 **.** 12 -3 **.** -2 **.** 0 **...** 17 **.** 20 **..**

HEAP)

[-4,-3,-2,15,17,12,0,20,16]

Dopo l'estrazione del minimo (-4), l’heap diventa:

[-3,15,-2,16,17,12,0,20]

**Esercizio 5.**