Benjamin Czaczkes PhD

The Hebrew University of Jerusalem,
Programming Instruction Unit,
Givat Ram, Jerusalem 91904, Israel

Site:pl1.mscc.huji.ac.il :אתר e-mail: msbc@mscc.huji.ac.il :דואר אלקט

ד"ר בנימין צ'צ'קס

האוניברסיטה העברית בירושלים,
היחידה להוראת התכנות,
גבעת רם, ירושלים 91904

Fax: 972-2-6586246 :פקס Tel: 972-2-6586297:טל'

תכנות C++ לכימאים מבחן מועד א' תשס"ב

יום א' 28 יוני 2002
משך המבחן 2 שעות
כל חומר כתוב מותר בשימוש

נתונה המחלקה באה:

```
01   class kuku // another name might have been more appropriate but less fun
02   }
03   Private:
04      char * name;// a rose by any other name …
05      int * data; // lots of integers
06      int size; //  number of integers in data not the last index
07   };
```

1. עבור 20 נקודות, הגדר למחלקה פונקציה בונה של ברירת המחדל ופונקציה הורסת (Default constructor, Destructor).
2. עבור 20 נקודות, הגדר למחלה פונקציה בונה המקבלת שם (עד 10 תווים לא כולל האפס), מערך של שלמים, ומספר שלם שהוא מספר האיברים במערך השלמים ומאתחלת את המחלקה בהתאם.
3. עבור 20 נקודות, הגדר למחלקה Copy Constructor, ואופרטור השמה. (operator=).
4. עבור 20 נקודות הגדר למחלקה אופרטור *
   1. מכפלה של קוקואים מוגדרת על 2 קוקו בצורה הבאה, אם גודלם אינו שווה המכפלה היא 0 אם שניהם ריקים המכפלה היא 0, אחרת המכפלה היא סכום מכפלות השלמים והיא מספר ממשי גדול long לדוגמה אם A הוא קוקו המכיל את המספרים 3,5 וB הוא קוקו המכיל את המספרים 4,3 אז A*B שווה ל4*5+4*3 שווה 32.
   2. עבור מכפלה של קוקו בשלם או שלם בקוקו מניחים שהשלם חוזר על עצמו כמספר השלמים בקוקו ומפעילים תהליך זהה למכפלת קוקו בקוקו.
5. עבור 20 נקודות העמס את אופרטור ההחדרה לקובץ כך שיוכל לעבור עם קוקו. יצר קוקו עם 10 שלמים וכתוב אותו לקובץ בשם kuku.txt בהצלחה בני

```
01   /***********************************
02     File Name: c:\My Documents\Cpp\TestA\TestA\main.cpp
03     Revision By: Benjamin Czaczkes PhD
04     Revised on 25/06/02 15:22:26
05    ***********************************/
06   #include <fstream.h>
07   #include <string.h> // strlen strcpy
08   #include <iomanip.h> // setw
09   class kuku
10   {
11   private:
12      char * name;
13      int * data;
14      int size;
15   public:
16      kuku(void) { name=NULL; data=NULL; size=0; return; }              // q1
17      ~kuku(void){ delete [] name; delete [] data; return; }            // q1
18      kuku(const char * Name, const int * Data, int Size);              // q2
19      kuku(const kuku & rhs);                                           // q3 cc
20      kuku & operator=(const kuku & rhs);                               // q3
21      friend long operator*(const kuku & lhs, const kuku & rhs);        // q4
22      friend long operator*(const kuku & lhs, long rhs);                // q4
23      friend long operator*(long lhs, const kuku & rhs);                // q4
24      friend ofstream & operator<<(ofstream & ofile, const kuku & rhs);// q5
25   };
26   void main(void)
27   {
28      kuku k;
29      int a[] = {1,1,10,10,100,100,1000,1000,10000,10000};
30      kuku kk("Ben The Man",a,10);
31      kuku kkk(kk);
32      k=kkk;
33      ofstream o("kuku.txt");
34      o<<k<<kk<<kkk << k*3 << "   " << 3*k << endl;
```

```
35     int b[]={1,2,3};
36     kuku kkkk("Man",b,3);
37     o << kkkk << endl << kkkk*kkkk*kkkk << endl;
38     o.close();
39     return;
40  }
41  kuku::kuku(const char * Name, const int * Data, int Size)    // q2
42  {
43     name = new char[strlen(Name)+1];
44     strcpy(name,Name);
45     data = new int[Size];
46     size=Size;
47     for (int i=0; i<size; i++) data[i]=Data[i];
48     return;
49  }
50  kuku::kuku(const kuku & rhs)                                 // q3 cc
51  {
52     name = new char[strlen(rhs.name)+1];
53     strcpy(name,rhs.name);
54     data = new int[rhs.size];
55     size=rhs.size;
56     for (int i=0; i<size; i++) data[i]=rhs.data[i];
57     return;
58  }
59  kuku & kuku::operator=(const kuku & rhs)               // q3
60  {
61     delete [] name; delete [] data; size=0;             // out with the old;
62     name = new char[strlen(rhs.name)+1];
63     strcpy(name,rhs.name);
64     data = new int[rhs.size];
65     size=rhs.size;
66     for (int i=0; i<size; i++) data[i]=rhs.data[i];
67     return *this;
68  }
69  ofstream & operator<<(ofstream & ofile, const kuku & rhs)    // q5
70  {
71     ofile << rhs.name << endl;
72     for (int i=0;i<rhs.size;i++)
73     {
74        ofile << setw(5) << i << ":" << setw(8) << rhs.data[i] << endl;
75     }
76     return ofile;
77  }
78  long operator*(const kuku & lhs, const kuku & rhs)           // q4
79  {
80     if ((lhs.size==0) && (rhs.size==0))     return 0;
81     else if (lhs.size!=rhs.size)            return 0;
82     else
83     {
84        long s=0;
85        for (int i=0;i<lhs.size;i++) s=(long)lhs.data[i]*(long)rhs.data[i]+s;
86        return s;
87     }
88  }
89  long operator*(const kuku & lhs, long rhs)                   // q4
90  {
91     long s=0;
92     for (int i=0;i<lhs.size;i++) s=(long)lhs.data[i]*(long)rhs+s;
93     return s;
94  }
95  long operator*(long lhs, const kuku & rhs)                   // q4
96  {
97     long s=0;
98     for (int i=0;i<rhs.size;i++) s=(long)rhs.data[i]*(long)lhs+s;
99     return s;
100 }
```