Θέμα 1.
Γράψτε τον κώδικα ενός header file που να περιέχει:
1) Ένα macro με όνομα `divides` που, αν του μεταβιβάσουμε δύο ακέραιους αριθμούς επιστρέφει 1 αν ο πρώτος αριθμός διαιρεί τον δεύτερο, αλλιώς, επιστρέφει 0.
2) Μια συνάρτηση `change` που, αν της δώσουμε δύο οποιαδήποτε διανύσματα `a` και `b` με ίσες διαστάσεις και στοιχεία τύπου `int`, να επιστρέφει (ΟΧΙ ΝΑ ΤΥΠΩΝΕΙ) ένα ανάλογο διάνυσμα `c` με στοιχεία `c(i)=a(i)+b(i)` αν το `a(i)` διαιρεί το `b(i)` και `c(i)=a(i)-b(i)` στην αντίθετη περίπτωση, εφαρμόζοντας το προηγούμενο macro.
3) Μια συνάρτηση `count` που, αν της μεταβιβάσουμε δύο οποιαδήποτε διανύσματα `a` και `b` με ίσες διαστάσεις και στοιχεία τύπου `int`, να μετρά και να επιστρέφει πόσα στοιχεία του `a` διαιρούν τα αντίστοιχα στοιχεία του `b` και πόσα όχι, εφαρμόζοντας το macro `divides`.
Φροντίστε ώστε αυτό το file να μην μπορεί να συμπεριληφθεί περισσότερες από μία φορές σε άλλο πηγαίο αρχείο. Γράψτε επίσης τον κώδικα ενός άλλου αρχείου που συμπεριλαμβάνει προς χρήση το προηγούμενο header file και περιέχει μια `main` συνάρτηση που χρησιμοποιεί τις προαναφερθείσες συναρτήσεις.

Απάντηση.
```
// We write in a header file named my_header.h
#ifndef my_header_h
#define my_header_h
#define divides(x,y) (((y)%(x)==0) ? 1 : 0)

void change (int n, int a[], int b[], int c[]) {
  for (int i=0; i<n; i++)
    if (divides(a[i],b[i])) c[i]=a[i]+b[i];
    else                    c[i]=a[i]-b[i];
}
void count (int n, int a[], int b[], int &count1, int &count2) {
  count1=count2=0;
  for (int i=0; i<n; i++)
    if (divides(a[i],b[i])) count1++;
    else                    count2++;
}
#endif

// We write in a source file named test.cpp
#include <iostream>
#include "my_header.h"
using namespace std;

int main() {
  const unsigned int n=3;
  int x[n];
  int y[n];
  int z[n];
  int c1,c2;
  for (int i=0; i<n; i++) cin >> x[i];
  for (int i=0; i<n; i++) cin >> y[i];
  change(n,x,y,z);
  for (int i=0; i<n; i++) cout << z[i] << '\n';
  count(n,x,y,c1,c2);
  cout << c1 << " " << c2 << '\n';
  return 0;
}
```

Θέμα 2.
Κατασκευάστε ένα πρόγραμμα που να αποτελείται από
1) μια ΑΝΑΔΡΟΜΙΚΗ συνάρτηση `subtract` που, αν της δίνουμε δύο οποιαδήποτε διανύσματα `a` και `b` με διάσταση `n` (διάφορη του 0) και στοιχεία τύπου `int`, να μας επιστρέφει ένα διάνυσμα `c` με στοιχεία `c(i)=a(i)-b(n-1-i)`.
2) μια ΑΝΑΔΡΟΜΙΚΗ συνάρτηση `sum` που, αν της δίνουμε ένα διάνυσμα a με στοιχεία τύπου `int`, να μας επιστρέφει το άθροισμα $\sum_{i=0}^{n-1} \frac{a(i)}{i+1}$.
3) μια `main` συνάρτηση που θα χρησιμοποιεί τις προαναφερθείσες συναρτήσεις αφού ορίσει τα απαραίτητα διανύσματα. Οι διαστάσεις αυτών των διανυσμάτων δεν θα είναι γνωστές κατά την φάση της μετάφρασης. Συνεπώς, χρειάζεσθε δυναμικά διανύσματα.
Να χρησιμοποιήσετε υποδείκτες (indices) για να κινηθείτε κατά μήκος των διανυσμάτων.

Απάντηση.
```
#include <iostream>
using namespace std;

void subtract (int n, int a[], int b[], int c[], int i) {
  if (i==0) c[0]=a[0]-b[n-1];
  else      { c[i]=a[i]-b[n-1-i]; subtract(n,a,b,c,i-1); }
}
float sum (int n, int a[], int i) {
  return (i==0) ? float(a[0]) : sum(n,a,i-1)+float(a[i])/(i+1);
}

int main() {
  unsigned int n;
  cin >> n;
  if (n==0) return -1;
  int *x= new int[n];
  int *y= new int[n];
  int *z= new int[n];
  for (int i=0; i<n; i++) cin >> x[i];
  for (int i=0; i<n; i++) cin >> y[i];
  subtract(n,x,y,z,n-1);
  for (int i=0; i<n; i++) cout << z[i] << '\n';
  cout << sum(n,x,n-1) << '\n';
  delete [] x; delete [] y; delete [] z;
  return 0;
}
```

Θέμα 3.
Υλοποιείστε το σενάριο της προηγούμενης άσκησης γράφοντας όλον τον κώδικα στη συνάρτηση `main` και χρησιμοποιώντας
1) τα κλασσικά επαναληπτικά σχήματα που διαθέτει η C++ αντί για αναδρομικές συναρτήσεις και
2) αριθμητική δεικτών (pointer arithmetic) αντί για υποδείκτες για να κινείστε κατά μήκος των διανυσμάτων.

Απάντηση.

```
#include <iostream>

using namespace std;


int main() {
  unsigned int n;
  cin >> n;
  if (n==0) return -1;
  int *x= new int[n]; int *y= new int[n]; int *z= new int[n];
  int *px; int *py; int *pz;
  for (px=x; px<x+n; px++) cin >> *px;
  for (py=y; py<y+n; py++) cin >> *py;
  px=x; py=y+n-1; pz=z;
  for ( ; px<x+n; px++) {
    *pz=*px-*py;
    py--; pz++; }
  for (pz=z ; pz<z+n; pz++)
    cout << *pz << '\n';
  float sum=0.0f;
  px=px-n;
  for (int i=0; i<n; i++) {
    sum=sum+float(*px)/(i+1);
    px++; }
  cout << sum << '\n';
  delete [] x; delete [] y; delete [] z;
  return 0;
}
```

Θέμα 4.
Κατασκευάζετε μια κλάση με όνομα `circle` που θα διαχειρίζεται κύκλους που βρίσκονται σε ένα επίπεδο Η κλάση θα περιλαμβάνει:
1) Τα απαραίτητα δεδομένα-μέλη για την αναπαράσταση ενός τέτοιου πολυωνύμου (ένα αντικείμενο τύπου char για το όνομα του κέντρου κύκλου, ένα αντικείμενο μιας δομής με δύο μέλη τύπου float για τις συντεταγμένες του και ένα αντικείμενο τύπου float για την ακτίνα του κύκλου).
2) Έναν constructor που θα επιτρέπει να του διαβιβάζετε μέσω των παραμέτρων του τα στοιχεία του δημιουργούμενου κύκλου.
3) Έναν copy constructor.
4) Έναν destructor που θα αναφέρει τα στοιχεία του κύκλου που καταστρέφει κάθε φορά.
5) Μια συνάρτηση μέλους `concentric` που θα εξετάζει αν ένας κύκλος είναι ομόκεντρος με το τρέχον αντικείμενο.
6) Μια συνάρτηση μέλους `area` που θα υπολογίζει το εμβαδόν του τρέχοντος αντικειμένου.
7) Μια συνάρτηση μέλους-υπερφόρτωση του τελεστή = που θα «αναθέτει» έναν κύκλο σε έναν άλλο.
8) Μια συνάρτηση μέλους-υπερφόρτωση του τελεστή `!=` που θα ελέγχει αν ένας κύκλος είναι διαφορετικός από έναν άλλο.
Το πρόγραμμα θα περιλαμβάνει επίσης μια συνάρτηση `print` που θα είναι φίλος της κλάσης και θα τυπώνει τα στοιχεία οποιουδήποτε μέλους της κλάσης και μια `main` συνάρτηση που θα δημιουργεί δύο στατικά και ένα δυναμικό αντικείμενο της κλάσης και θα χρησιμοποιεί όλες τις συναρτήσεις μέλους και την `print`. Γράψτε τον αντίστοιχο κώδικα.

Απάντηση.
```
#include <iostream>
#include <cmath>
using namespace std;

const float pi=4.0f*atan(1.0f);

struct point {
  float x;
  float y;
};

class circle {
  private:
    char c;
    point pos;
    float r;
  public:
    circle (char cc, point ppos, float rr) {
      c=cc;
      pos.x=ppos.x;
      pos.y=ppos.y;
      r=rr;
    }
    circle (const circle &a) {
      c=a.c;
      pos.x=a.pos.x;
      pos.y=a.pos.y;
      r=a.r;
    }
    ~circle () {
      cout<<"\ndestructing "<<c<<" "<<pos.x<<" "<<pos.y<<" "<<r<<"\n";
    }
    bool concentric (const circle &a) {
      return c==a.c;
    }
    float area () {
      return pi*r*r;
    }
    circle & operator= (const circle &a) {
```

```
      if (this!=&a) {
        c=a.c;
        pos.x=a.pos.x;
        pos.y=a.pos.y;
        r=a.r; }
        return *this;
    }
    bool operator!= (const circle &a) {
      if (c!=a.c || r!=a.r) return true;
      else return false;
    }
    friend void print (circle a);
};

void print (circle a) {
  cout << a.c << " " << a.pos.x << " " << a.pos.y << " " << a.r;
}

int main() {
  point p; p.x=3.0f; p.y=2.f;
  circle c1('A',p,3.4f);
  circle c2(c1);
  print(c1);
  if (c1.concentric(c2))
    cout << " concentric with ";
  else
    cout << " is not concentric with ";
  print(c2); cout << "\n";
  circle *c3= new circle(c2);
  cout << c3->area() << '\n';
  print(c2);
  if (c2!=c1)
    cout << " is not the same with ";
  else
    cout << " is the same with ";
  print(c1); cout << "\n";
  return 0;
}
```