# ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
# Κ10: ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ
## Εξετάσεις 2 Σεπτεμβρίου 2006

1. (α′) Πού εστιάζει ο διαδικαστικός προγραμματισμός (procedural programming) και πού ο αντικειμενοστραφής; Δώστε ένα παράδειγμα γλώσσας για τον καθένα. Δώστε, επιγραμματικά, ένα παράδειγμα προβλήματος που ενδείκνυται να υλοποιηθεί με μια γλώσσα διαδικαστικού προγραμματισμού και ένα που ενδείκνυται να υλοποιηθεί με μια γλώσσα αντικειμενοστραφούς προγραμματισμού.

   (β′) Έστω ότι έχουμε το παρακάτω πρόγραμμα:

```
#include <iostream.h>
class A{ int data;
public:
   A(int datav=0) : data(datav) { cout<< "I just constructed an A" << endl; }
   A(A& a) { cout<< "I just constructed an A by copying" << endl;
             data = a.data; }
   A& operator=(const A&)
      { cout << "I do nothing at all" << endl;  return *this; }
   ~A(){ cout << "I am destructing an A with value: " << data << endl; }
   void put(int datav) { data = datav;} };

main() { A a1; A a2(a1); A a3 = a2; a1.put(5); a3 = a1; }
```

   Τι εκτυπώνεται κατά την εκτέλεσή του; Δώστε το πρότυπο της συνάρτησης (function prototype) που καλείται κατά την εκτέλεση κάθε εντολής της `main`.

2. Αιτιολογώντας την απάντησή σας, δώστε το αποτέλεσμα της εκτέλεσης του παρακάτω προγράμματος C++, αποσχολιάζοντας τις σχολιασμένες γραμμές κώδικα μια κάθε φορά:

```
#include <iostream.h>
class Block{ int data;
 public:
   Block(int datav = 0) : data(datav)
        { cout << "Constructing a Block with " << datav << endl; }
   ~Block(){ cout << "Destructing a Block with: " << data << endl; }
   void double_it() { data = data * 2 ;} };

class B1{ Block* data;
 public:
  B1(int value) { cout << "Constructing a B1 with value " << value << endl;
                  data = new Block(value); }
  B1(const B1& b1) { data = new Block(*(b1.data)); }
  ~B1(){  cout << "Destructing a B1 with block " << endl;
              delete data; } };

class B2{ Block data;
 public:
   B2(int datav) : data(datav)
        { cout << "Contructing a B2 with value: " << datav << endl; }
   ~B2(){ cout << "Destructing a B2 with block " << endl; }
   Block safe() { return data; }
   Block& unsafe() { return data; } };
```

```
class B{ B1 data1; B2 data2;
 public:
   B(int datav) : data2(datav), data1(datav)
       { cout << "Constructing a B with: " << datav << endl; }
   ~B() { cout << "Destructing a B" << endl; }
   B2 reveal1() {return data2;}
   B2& reveal2() {return data2;} };

main(){
  B ba(10);
// B bb(ba);

//  ba.reveal1().safe().double_it();
//  ba.reveal1().unsafe().double_it();
//  ba.reveal2().safe().double_it();
//  ba.reveal2().unsafe().double_it();
}
```

3. Δίδεται το παρακάτω ημιτελές πρόγραμμα C++:

```
#include <iostream.h>
class Ingredient{ int quantity;
 public:
   Ingredient(int q) : quantity(q) { cout << "I am using :" << q << " of "; id();}
   Ingredient& mix(Ingredient& ingr)
             { cout << "Adding " << ingr.quantity << " of "; ingr.id();
               cout << " to " << endl; id();
               Ingredient* pin = new Ingredient(quantity + ingr.quantity);
               return *pin; }
   virtual void id() { cout << "an ingredient " << endl; }};

class Sugar : public Ingredient{
   public:
      Sugar(int q) : Ingredient(q) { cout<< "sugar" << endl; }
      void id() { cout << "sugar" << endl; }};
class Eggs : public Ingredient{
   public:
      Eggs(int q) : Ingredient(q) { cout<< "eggs" << endl; }
      void id() { cout << "eggs" << endl; }};
class Butter : public Ingredient{
   public:
      Butter(int q) : Ingredient(q) { cout<< "butter" << endl; }
      void id() { cout << "butter" << endl; }};
class Powder : public Ingredient{
   public:
      Powder(int q) : Ingredient(q) { cout<< "powder" << endl; }
      void id() { cout << "powder" << endl; }};
class Cocoa : public Ingredient{
   public:
      Cocoa(int q) : Ingredient(q) { cout<< "cocoa" << endl; }
      void id() { cout << "cocoa" << endl; }};
```

```
class Cheese : public Ingredient{
   public:
      Cheese(int q) : Ingredient(q) { cout<< "cheese" << endl; }
      void id() { cout << "cheese" << endl; }};

class Food{
 protected: int Mins;
 public:
   Food(int mins = 0) : Mins(mins) {}
   virtual void bake() { cout << " Put to oven for " << endl; }
   virtual void prepare() = 0;};

class Cake : public Food{
   Sugar& sugar; Eggs& eggs; Butter& butter; Powder& powder;
 public :
   Cake(Sugar& s, Eggs& e, Butter& b, Powder& pow) :
        Food(90), sugar(s), eggs(e), butter(b), powder(pow)
        { cout << " I am going to prepare a cake " << endl; }
   void mix() { powder.mix(butter.mix(sugar.mix(eggs))); }
   virtual void bake () { Food::bake(); cout << Mins << " mins" << endl; }
   virtual void prepare() { mix(); bake();}};
class ChocoCake : public Cake{ Cocoa& cocoa;
 public:
   ChocoCake(Sugar& s, Eggs& e, Butter& b, Powder& p, Cocoa& co) :
   Cake(s,e,b,p), cocoa(co) { cout << " with cocoa " << endl ;}
   void add_speciality() { cout << "adding cocoa" << endl; }
   virtual void prepare() { mix(); add_speciality(); bake();}};

class CheesePie : public Food{
   Eggs& eggs; Butter& butter; Cheese& cheese;
 public :
   CheesePie(Eggs& e, Butter& b, Cheese& ch) :
        Food(120), eggs(e), butter(b), cheese(ch)
        { cout << " I am going to prepare a cheesepie " << endl; }
   void mix() { cheese.mix(butter.mix(eggs)); }
   virtual void bake () { Food::bake(); cout << Mins << " mins" << endl; }
   void wrap() { cout << "adding pastry " << endl;}
   virtual void prepare() { mix(); wrap(); bake();}};

void prepare(Food& fagito) { fagito.prepare(); }

main(){
//   Part A
//   TYPE fagito(PARAMS);
     prepare(fagito);
}
```

Τι τιμές μπορεί να πάρει το TYPE; Για κάθε μια από αυτές αφαιρέστε τα σχόλια από το σώμα της main και αντικαταστήστε με τους κατάλληλους ορισμούς το Part A και κατάλληλες παραμέτρους το PARAMS για να μπορεί να εκτελεστεί η main. Αιτιολογώντας την απάντησή σας, δώστε το αποτέλεσμα της εκτέλεσης.

4. Έστω ότι έχουμε να υλοποιήσουμε σε C++ μία προσομοίωση της αίθουσας αναχωρήσεων ενός αεροδρομίου. Η αίθουσα αποτελείται από ένα σύνολο από "Θέσεις εξυπηρέτησης". Οι επιβάτες που πρόκειται να αναχωρήσουν εξυπηρετούνται ένας-ένας από τις "Θέσεις Εξυπηρέτησης" στις οποίες σχηματίζουν ουρές. Μια "Θέση Εξυπηρέτησης" εξυπηρετεί είτε τους επιβάτες μιας συγκεκριμένης πτήσης εξωτερικού είτε οποιαδήποτε πτήση εσωτερικού (*σημείωση*: αεροπορικές εταιρείες δεν λαμβάνονται υπόψη στην προσομοίωση). Αν ο χρόνος μιας πτήσης εσωτερικού πλησιάζει (π.χ. απομένει μισή ώρα για την αναχώρησή της), καλούνται οι επιβάτες της να προωθηθούν στις ουρές που περιμένουν. Δηλαδή, φεύγουν από τις θέσεις τους και έρχονται στις πρώτες θέσεις της ουράς που βρίσκονται, ο ένας μετά τον άλλον (με οποιονδήποτε τρόπο).

   Για την προσομοίωση να χρησιμοποιηθούν οι κλάσεις "Αναχωρήσεις" (`Departures`), "Θέση εξυπηρέτησης" (`CheckinCounter`), "Επιβάτης" (`Passenger`) και "Πτήση" (`Flight`).

   Η κλάση "Αναχωρήσεις":

   - έχει 100 Θέσεις εξυπηρέτησης (`counters`)
   - έχει ένα ρολόι που καταγράφει τον χρόνο σε λεπτά (`time`)

   Η κλάση "Αναχωρήσεις" χαρακτηρίζεται από την εξής συμπεριφορά:

   - Αρχικά ο χρόνος είναι μηδενικός και σε κάθε μια από τις θέσεις εξυπηρέτησης καταχωρείται αν εξυπηρετεί εσωτερικές πτήσεις ή τον αριθμό πτήσης που εξυπηρετεί
   - Γίνεται άφιξη επιβάτη, ο οποίος προστίθεται για να περιμένει στην θέση εξυπηρέτησης στην μικρότερη ουρά από εκείνες που εξυπηρετούν την πτήση του (`arrive`)
   - Γίνεται εξυπηρέτηση, εξυπηρετώντας επιβάτη από τη θέση εξυπηρέτησης με τους περισσότερους επιβάτες σε αναμονή (`serve`)
   - Γίνεται κλήση εσωτερικής πτήσης προκειμένου να προωθηθούν οι επιβάτες της στις θέσεις εξυπηρέτησης που περιμένουν, αν ο χρόνος της πτήσης πλησιάζει (`announce_flight`)

   Η κλάση "Θέση εξυπηρέτησης":

   - για τις πτήσεις εξωτερικού, φυλάσσει τον αριθμό της πτήσης ή το μηδέν αν πρόκειται για πτήσεις εσωτερικού (`flightno`)
   - έχει μια ουρά από επιβάτες (`passenger_queue`)
   - έχει έναν μετρητή των επιβατών που περιμένουν στην ουρά (`waiting_passengers`)

   Η κλάση "Θέση εξυπηρέτησης" χαρακτηρίζεται από την εξής συμπεριφορά:

   - Αρχικά η ουρά των επιβατών είναι κενή και θεωρείται ότι εξυπηρετεί εσωτερικές πτήσεις
   - Επιτρέπει την αλλαγή της πτήσης που εξυπηρετεί (`set_flightno`)
   - Εξυπηρετεί, αφαιρώντας επιβάτη από την ουρά και μειώνοντας τον μετρητή των επιβατών που περιμένουν (`serve`)
   - Γίνεται άφιξη επιβάτη, ο οποίος προστίθεται για να περιμένει στο τέλος της ουράς και αυξάνεται ο μετρητής των επιβατών που περιμένουν (`arrive`)
   - Αν εξυπηρετεί πτήσεις εσωτερικού, καλεί την πτήση αν πλησιάζει η ώρα αναχώρησής της για να προωθηθούν οι επιβάτες της στην ουρά (`announce_flight`)

   Οι "Πτήσεις" δομούνται από τον αριθμό τους, από την ένδειξη αν είναι εσωτερικού ή όχι και την ώρα αναχώρησής τους.

   Οι "Επιβάτες" δομούνται από τον αριθμό διαβατηρίου τους και τον αριθμό της πτήσης τους.

   Να υλοποιήσετε σε C++ τις παραπάνω κλάσεις, ορίζοντας νέες όπου δείχνει ενδεικνυόμενο.
   *Σημείωση*: Όπου θεωρείτε απαραίτητο, κάνετε παραδοχές στις προδιαγραφές, τεκμηριώνοντάς τις.