Ανάπτυξη Κώδικα σε Αντικειμενοστραφές Περιβάλλον Προγραμματισμού

Νίκος Ποθητός
pothitos@di.uoa.gr

8/5/2006

Ανάπτυξη Κώδικα σε Αντικειμενοστραφές Περιβάλλον Προγραμματισμού

- Οι αντικειμενοστραφείς γλώσσες προγραμματισμού διευκολύνουν στην καλύτερη οργάνωση του κώδικα.
- Στη C++ μπορούμε να σχεδιάζουμε με αντικειμενοστραφή τρόπο αποδοτικές εφαρμογές.
- Για να γράψουμε κώδικα C++ που επιλύει ένα συγκεκριμένο πρόβλημα, χρειάζεται προηγουμένως να δούμε προσεκτικά την περιγραφή του προβλήματος αυτού. Θα πρέπει να διακρίνουμε τις οντότητες που υπάρχουν στο πρόβλημα, τις ιδιότητές τους και τις μεταξύ τους σχέσεις. Θα σκεφτούμε επίσης τους αλγορίθμους επίλυσης του προβλήματος.

Σύστημα Κατάρτισης Ωρολογίου Προγράμματος

- Σήμερα, τα μεγάλα προβλήματα χρονοπρογραμματισμού επιλύονται με τη βοήθεια υπολογιστή. Τέτοια προβλήματα είναι π.χ. οι βάρδιες του προσωπικού μιας αεροπορικής εταιρείας, ο καθορισμός ημερομηνιών για τις επαγγελματικές συναντήσεις, το ωρολόγιο πρόγραμμα ενός εκπαιδευτικού ιδρύματος κ.ά.
- Ο βέλτιστος χρονοπρογραμματισμός και η αποδοτική κατάρτιση ωρολογίων προγραμμάτων είναι ένα πρόβλημα (και) της Τεχνητής Νοημοσύνης.
- Εκτός από εφαρμογές που επιλύουν τέτοια προβλήματα, έχουν αναπτυχθεί και αρκετές, εμπορικές κυρίως, βιβλιοθήκες επίλυσης.

Περιγραφή του προβλήματος

Για αρχή, θα ασχοληθούμε με μία απλή εκδοχή του προβλήματος κατάρτισης ωρολογίου προγράμματος:

Έστω ότι μας έχει ανατεθεί η δημιουργία του ωρολογίου προγράμματος για ένα γυμνάσιο. Το πρόγραμμα αφορά μόνο μία ημέρα της εβδομάδας. Γνωρίζουμε ποιες τάξεις –ή καλύτερα ποια τμήματα– υπάρχουν στο σχολείο αυτό. Σε κάθε τάξη διδάσκονται συγκεκριμένα μαθήματα από έναν/μία διδάσκοντα/ουσα το καθένα. Κάθε μάθημα διαρκεί μία ώρα.

Περιγραφή του προβλήματος: Σχόλια

- Η παραπάνω περιγραφή είναι σχετικά σαφής. Ωστόσο δεν αναφέρονται σε αυτήν κάποιες προφανείς παράμετροι του πραγματικού κόσμου που θα πρέπει να ληφθούν υπόψη κατά τη μοντελοποίηση του προβλήματος.
- Π.χ. ένας καθηγητής δεν μπορεί να βρίσκεται ταυτόχρονα σε δύο αίθουσες.
- Ακόμα, σε μία τάξη δεν μπορούν να γίνονται ταυτόχρονα δύο διαφορετικά μαθήματα!

Αναδρομικός αλγόριθμος επίλυσης σε ψευδοκώδικα

```
SCHEDULE(ℓ, nlesson)
  if ℓ == nlesson
    return true        // επιτυχία

  for h = 0 to nhours − 1 {
    if ο καθηγητής και η αίθουσα του μαθήματος
         είναι διαθέσιμα την ώρα h {
           Δέσμευσε τον καθηγητή και την αίθουσα για την h.
           Προγραμμάτισε το ℓ να γίνεται την h.
           if SCHEDULE(ℓ + 1, nlesson) == true
             return true
           Ακύρωσε την ανάθεση της h για το ℓ.
           Ακύρωσε τη δέσμευση καθηγητή και αίθουσας.
    }
  }
  return false         // αποτυχία εύρεσης ώρας για το μάθημα ℓ
```

Απλός Διαδικαστικός Προγραμματισμός

- Στη συνέχεια παρουσιάζεται το αρχείο κώδικα `only_main.cpp`, το οποίο κάθε άλλο παρά το δρόμο της αντικειμενοστραφούς μοντελοποίησης ακολουθεί.
- Γίνεται απόπειρα να «στριμωχτούν» μέσα σε μία `main()` όλες οι οντότητες του προβλήματος.
- Ο κώδικας αυτός θα μπορούσε κάλλιστα να γραφεί και με απλή C.

only_main.cpp (1/6)

```
#include <iostream>
#include <string>
using namespace std;

//  A constant for declaring an empty hour for a classroom
const int  EMPTY = -1;

//  How many hours exist in a day
const int  nhours = 3;

inline bool
solve_rec (int course_room[], int course_prof[], int course_hour[],
           int ncourse, bool room_unavail[][nhours],
           bool prof_unavail[][nhours]);
```

only_main.cpp (2/6)

```
int  main (void)
{
  //  The following names could be used for printing
  //  the timetable on the screen.
  string  room_name[] = {"C1", "C2"};
  string  prof_name[] = {"Einstein", "Godel", "Eco"};
  string  course_name[] = {"Physics", "Physics", "Maths",
       "Maths", "Philosophy", "Philosophy", "Literature"};

  //  Availabilities of our resources
  bool  room_unavail[][nhours] = { {false, false, false},
                                   {false, false, false} };
  bool  prof_unavail[][nhours] = { {false, false, false},
                                   {false, false, false},
                                   {false, false, false} };
```


only_main.cpp (3/6)

```
  //  Indexes of the corresponding classroom and professor
  //  for each course, and the hour it is scheduled at
  //  (currently EMPTY).
  int  course_room[] = {0, 1, 0, 1, 0, 1, 0};
  int  course_prof[] = {0, 0, 1, 1, 2, 2, 2};
  int  course_hour[] = {EMPTY, EMPTY, EMPTY, EMPTY,
                        EMPTY, EMPTY, EMPTY};

  //  If we change ncourse to 7, the scheduler
  //  will not find any solution.
  int  ncourse = 6;

  if (solve_rec(course_room,course_prof,course_hour,ncourse,
                           room_unavail,prof_unavail) == true)
     cout << "Found a solution\n";
  else
     cout << "Couldn't find a solution\n";
}
```

only_main.cpp (4/6)

```
bool
solve_rec_body (int course_room[], int course_prof[],
  int course_hour[], int curr_course, int ncourse,
  bool room_unavail[][nhours], bool prof_unavail[][nhours])
{
  //  "Renaming" (for better readability)
  int  l = curr_course;

  //  Check if all the courses have been scheduled
  if (l == ncourse)
    return  true;

  for (int h=0;  h<nhours;  ++h)  {
    if ( !room_unavail[course_room[l]][h]
          &&  !prof_unavail[course_prof[l]][h] )  {

      //  Assign resources and hour
      room_unavail[course_room[l]][h] = true;
      prof_unavail[course_prof[l]][h] = true;
      course_hour[l] = h;
```


only_main.cpp (5/6)

```
      //  Continue with scheduling the next course
      if (solve_rec_body(course_room,course_prof,course_hour,
            l+1,ncourse,room_unavail,prof_unavail) == true)
            return  true;

      //  Free resources
      room_unavail[course_room[l]][h] = false;
      prof_unavail[course_prof[l]][h] = false;
      course_hour[l] = EMPTY;
    }
  }

  //  Couldn't find an hour, for course `l'
  return  false;
}
```

only_main.cpp (6/6)

```
inline bool
solve_rec (int course_room[], int course_prof[], int course_hour[],
           int ncourse, bool room_unavail[][nhours],
           bool prof_unavail[][nhours])
{
  return  solve_rec_body(course_room,course_prof,course_hour,
                         0,ncourse,room_unavail,prof_unavail);
}
```


only_main.cpp: Αποτελέσματα

```
% g++ -o only_main only_main.cpp
% ./only_main
Found a solution
%
```

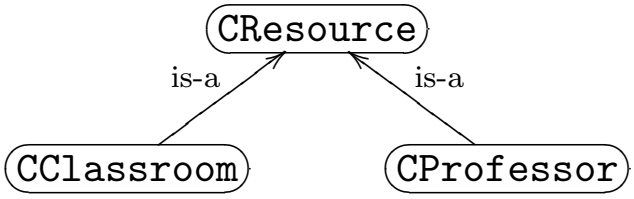
- Για απλότητα, το only_main.cpp δεν εκτυπώνει το ωρολόγιο πρόγραμμα· αναφέρει μόνο αν υπάρχει κάποια λύση στο πρόβλημα.
- Παρατηρούμε ότι χωρίς τη χρήση κλάσεων δεν είναι ξεκάθαρο το ποιες είναι τελικά οι οντότητες του προβλήματος. Υπάρχει μία «ισοπέδωση» των δεδομένων, αφού η αναπαράστασή τους περιστρέφεται γύρω από τους αλγορίθμους μόνο.

Αντικειμενοστραφής Μοντελοποίηση (1/2)

- Για να προχωρήσουμε στην αντικειμενοστραφή μοντελοποίηση, θα πρέπει να εντοπίσουμε τις οντότητες του προβλήματος, καθώς και τις μεταξύ τους σχέσεις.
- Όσον αφορά το πρόβλημά μας, θα δημιουργήσουμε μία κλάση για τα *μαθήματα*, μία για τους *καθηγητές* και άλλη μία για τις *τάξεις*.
- Κάθε μάθημα περιέχει *αναφορά* (reference) στον/στην καθηγητή/τρια που το διδάσκει και στην τάξη στην οποία θα γίνει.

Αντικειμενοστραφής Μοντελοποίηση (2/2)

- Στον χρονοπρογραμματισμό, ένας *πόρος* (resource) είναι κάτι το οποίο ανά πόσα χρονική στιγμή μπορεί να ανατεθεί σε το πολύ μία δραστηριότητα.
- Ένας καθηγητής μπορεί να θεωρηθεί ότι *είναι-ένας* (is-a) πόρος: καθεμία ώρα μπορεί να διδάσκει ένα μόνο μάθημα. Όμοια, σε μία αίθουσα μπορεί να λαμβάνει χώρα ένα το πολύ μάθημα την ώρα, οπότε και η αίθουσα κληρονομεί τις ιδιότητες ενός πόρου



- Όσον αφορά τον αλγόριθμο επίλυσης, μπορούμε να υλοποιήσουμε δύο εναλλακτικές τεχνικές αναζήτησης:
α) με αναδρομή (έμμεση χρήση στοίβας) και
β) με εμφανή χρήση στοίβας.

timetable.h (1/8)

```
#ifndef TIMETABLE_H
#define TIMETABLE_H

#include <string>
#include <iostream>


namespace timetable  {

 class  CResource  {
   private:
     bool  *slot;
     int  nhours;
     //  slot[h]==true means that the resource is unavailable at
     //   hour ‘h’.  The length of the array ‘slot’ is ‘nhours’.
```


timetable.h (2/8)

```
   public:
     CResource (int nhours_init)
      : nhours(nhours_init)
     {
       slot = new bool[nhours];
       for (int h=0;  h<nhours;  ++h)
         slot[h] = false;
       //  initially the resource is always available
     }

     ~CResource (void)
     {
       delete [] slot;
     }
```

timetable.h (3/8)

```
    void  reserve (int hour)
    {
      slot[hour] = true;
    }

    void  release (int hour)
    {
      slot[hour] = false;
    }

    bool  is_free (int hour)  const
    {
      return  ( !slot[hour] );
    }
};
```


timetable.h (4/8)

```
class CClassroom : public CResource  {
  private:
    const  int  id;
    static int  count;
    std::string  name;

  public:
    CClassroom (std::string name_init, int nhours_init)
     : CResource(nhours_init), id(++count),
       name(name_init)  {   }

    friend std::ostream&
    operator << (std::ostream& os, const CClassroom& room);

    int  get_id (void)  const
     {  return  id;  }
};
```

timetable.h (5/8)

```
class CProfessor : public CResource  {
  private:
    const  int  id;
    static int  count;
    std::string  name;

  public:
    CProfessor (std::string name_init, int nhours_init)
     : CResource(nhours_init), id(++count),
       name(name_init)  {   }

    friend std::ostream&
    operator << (std::ostream& os, const CProfessor& prof);

    int  get_id (void)  const
     {  return  id;  }
};
```


timetable.h (6/8)

```
class  CCourse  {
  private:
    std::string  name;
    int  hour_scheduled;
    static const int  EMPTY = -1;
    //  hour_scheduled==EMPTY means that
    //   the course has not been scheduled.
    CClassroom&  classroom;
    CProfessor&  professor;

  public:
    CCourse (std::string name_init, CClassroom& cl_init,
             CProfessor& pr_init)
     : name(name_init), hour_scheduled(EMPTY),
       classroom(cl_init), professor(pr_init)  {   }

    friend std::ostream&
    operator << (std::ostream& os, const CCourse& les);
```

timetable.h (7/8)

```
        CClassroom&  get_class (void)  const
         {  return  classroom;  }

        CProfessor&  get_prof (void)  const
         {  return  professor;  }

        int  get_hour (void)  const
         {  return  hour_scheduled;  }

        bool  schedule (int hour)
        {
          if (classroom.is_free(hour)
              && professor.is_free(hour))  {
                classroom.reserve(hour);
                professor.reserve(hour);
                hour_scheduled = hour;
                return  true;
          } else  {
                return  false;
          }
        }
```


timetable.h (8/8)

```
        void  cancel_schedule (void)
        {
          classroom.release(hour_scheduled);
          professor.release(hour_scheduled);
          hour_scheduled = EMPTY;
        }
    };

    bool
    solve_rec (CCourse course[], int ncourse, int nhours);

    void
    print_timetable (CCourse course[], int ncourse,
          CClassroom room[], int nroom, int nhours);
  } // end namespace

  #endif  // TIMETABLE_H
```

timetable.cpp (1/6)

```
#include "timetable.h"
using namespace timetable;
using namespace std;

//  Initializations of the counters of the instances
//   of the classes
int  CClassroom::count = 0;
int  CProfessor::count = 0;
```


timetable.cpp (2/6)

```
//  Overloaded operators for printing
ostream& timetable::operator <<
  (ostream& os, const CClassroom& room)
{
  os << room.name;
  return  os;
}

ostream& timetable::operator <<
  (ostream& os, const CProfessor& prof)
{
  os << prof.name;
  return  os;
}

ostream& timetable::operator <<
  (ostream& os, const CCourse& c)
{
  os << c.name;
  return  os;
}
```

timetable.cpp (3/6)

```
namespace {
  bool  solve_rec_body (CCourse course[], int ncourse,
                        int curr_course, int nhours)
  {
    if (curr_course == ncourse)
      return  true;  // success: every course has been scheduled

    //  variable renaming (for readability)
    CCourse&  c = course[curr_course];

    for (int h=0;  h<nhours;  ++h)  {
      if (c.schedule(h) == true)  {
        if (solve_rec_body(course,ncourse,
             curr_course+1,nhours) == true)
                return  true;
        c.cancel_schedule();
      }
    }
    return  false;   // failed to find an appropriate hour
  }
} // end namespace
```


timetable.cpp (4/6)

```
//  Recursive search for a solution (timetable)
bool
timetable::solve_rec (CCourse course[], int ncourse, int nhours)
{
  return  solve_rec_body(course,ncourse,0,nhours);
}
```

```
#include <iostream>
#include <iomanip>
```

timetable.cpp (5/6)

```
namespace {
  //  Serial search for the course scheduled to take place
  //   in room 'room' at hour 'hour'
  int  search_course (CCourse course[], int ncourse,
                      CClassroom& room, int hour)
  {
    for (int c=0;  c<ncourse;  ++c)  {
      if (course[c].get_class().get_id() == room.get_id()
          && course[c].get_hour() == hour)
              return  c;
    }
    return  -1;
  }

  inline void  print_bar (int columns)
  {
    for (int i=0;  i<columns;  ++i)
      cout << "+------------";
    cout << "+\n";
  }
} // end namespace
```


timetable.cpp (6/6)

```
void
timetable::print_timetable (CCourse course[], int ncourse,
                 CClassroom room[], int nroom, int nhours)
{
  int  h, r, c;
  print_bar(nroom);
  for (r=0;  r<nroom;  ++r)  {
    cout << "| ";
    cout << setiosflags(ios::left) << setw(10) << room[r] << " ";
  }
  cout << "|\n";
  print_bar(nroom);
  for (h=0;  h<nhours;  ++h)  {
    for (r=0;  r<nroom;  ++r)  {
      c = search_course(course,ncourse,room[r],h);
    cout << "| ";
    cout << setiosflags(ios::left) << setw(10) << course[c] << " ";
    }
    cout << "|\n";
  }
  print_bar(nroom);
}
```

main.cpp (1/2)

```
#include "timetable.h"
using namespace timetable;
#include <iostream>
using namespace std;
int  main (void)
{ int  nhours = 3;          // hours of the timetable
  CClassroom  room[] = {
    CClassroom("C1", nhours),
    CClassroom("C2", nhours)  };
  CProfessor  prof[] = {
    CProfessor("Einstein", nhours),
    CProfessor("Godel",    nhours),
    CProfessor("Eco",      nhours)  };
  CCourse  course[] =  {
    CCourse("Physics",    room[0], prof[0]),
    CCourse("Physics",    room[1], prof[0]),
    CCourse("Maths",      room[0], prof[1]),
    CCourse("Maths",      room[1], prof[1]),
    CCourse("Philosophy", room[0], prof[2]),
    CCourse("Philosophy", room[1], prof[2]),
    CCourse("Literature", room[0], prof[2])  };
```


main.cpp (2/2)

```
  cout << "First case, with 6 courses:\n";
  if (solve_rec(course,6,nhours) == true)  {
     cout << "Found a solution\n\n";
     print_timetable(course, 6, room, 2, nhours);
  } else  {
     cout << "Couldn't find a solution\n";
  }

  //cout << "Second case, with 7 courses:\n";
  //if (solve_rec(course,7,nhours) == true)  {
  //   cout << "Found a solution\n\n";
  //   print_timetable(course, 7, room, 2, nhours);
  //} else  {
  //   cout << "Couldn't find a solution\n";
  //}
}
```

Αποτελέσματα

```
% g++ -o timetable timetable.cpp main.cpp
% ./timetable
First case, with 6 courses:
Found a solution

+------------+------------+
| C1         | C2         |
+------------+------------+
| Physics    | Philosophy |
| Maths      | Physics    |
| Philosophy | Maths      |
+------------+------------+
%
```

Σημείωση: Αν μεταγλωττιστεί ο κώδικας στα μηχανήματα Sun του Τμήματος με την προκαθορισμένη έκδοση του g++, τα αποτελέσματα θα εμφανιστούν κανονικά, αλλά χωρίς στοίχιση. Αν χρησιμοποιήσουμε την πιο καινούργια έκδοση του g++, δεν θα υπάρχει αυτό το πρόβλημα. Το μόνο που χρειάζεται για να γίνει αυτό, είναι να τρέξουμε, πριν τις εντολές του παραπάνω πλαισίου, την εντολή

```
alias g++ '/usr/sfw/bin/g++'
```


Αναζήτηση λύσης με στοίβα

- Αν αντί της συνάρτησης `solve_rec()` (αναζήτηση με αναδρομή), χρησιμοποιήσουμε την `solve_with_stack()` (αναζήτηση με εμφανή χρήση στοίβας) του `solve_stack.h`, θα πάρουμε τα ίδια ακριβώς αποτελέσματα.
- Εξάλλου, οι δύο αυτές συναρτήσεις υλοποιούν τον ίδιο αλγόριθμο, που λέγεται αναζήτηση λύσης με *οπισθοδρόμηση* (backtracking).

solve_stack.h

```
#ifndef SOLVE_STACK_H
#define SOLVE_STACK_H

#include "timetable.h"


namespace timetable  {

 bool
 solve_with_stack (CCourse course[], int ncourse, int nhours);

} // end namespace

#endif  // SOLVE_STACK_H
```


solve_stack.cpp (1/3)

```
#include "solve_stack.h"
using namespace timetable;

#include "my_stack.h"

bool
timetable::solve_with_stack(CCourse course[],int ncourse,int nhours)
{
  my_stack  stack_choices;
```

solve_stack.cpp (2/3)

```
int  c=0, h=0;
while (c < ncourse)  {
  while (h < nhours)  {
    if (course[c].schedule(h) == true)  {
        // Course No ‘c’ has been scheduled at hour ‘h’
        stack_choices.push(h);
        ++c;
        h = 0;
        break;
    } else  {
        ++h;
    }
  }
  if (h == 0)
      continue;
  // Search for course No ‘c’ failed
  if (c == 0)
      return  false;
```


solve_stack.cpp (3/3)

```
  // Canceling previous assignment for course No ‘c-1’
  --c;
  course[c].cancel_schedule();
  h = stack_choices.top() + 1;  // proceed to the next hour
  stack_choices.pop();
}
return  true;  // success: every course has been scheduled
}
```

Υλοποιήσεις στοίβας

Έχουμε τρεις εναλλακτικές υλοποιήσεις του τύπου δεδομένων στοίβα:

1. Στα αρχεία `my_stack.h` και `my_stack.cpp` υλοποιείται μία απλή στοίβα ακεραίων (κλάση `my_stack`).
2. Στο αρχείο `my_stack_templ.h` υλοποιείται μία template στοίβα (κλάση `my_stack_templ`), δηλαδή μία στοίβα της οποίας ο τύπος των στοιχείων μπορεί να είναι ο οποιοσδήποτε – οπότε μπορεί να είναι μία στοίβα ακεραίων, ή μια στοίβα συμβολοσειρών, ή μια στοίβα στοιβών κ.ο.κ.
3. Τέλος υπάρχει η έτοιμη λύση της STL (κλάση `stack` του ομώνυμου αρχείου επικεφαλίδας της πρότυπης βιβλιοθήκης).

my_stack.h (1/2)

```
#ifndef MY_STACK_H
#define MY_STACK_H

class  my_stack  {
  private:
    struct stacknode {
      int  thedata;
      struct stacknode  *next;
    };

    stacknode  *stack;


  public:
    my_stack (void)
     : stack(0)  {  }
```

my_stack.h (2/2)

```
    ~my_stack (void)
    {
      while (stack != 0)
          pop();
    }

    int  top (void)  const
    {
        return  stack->thedata;
    }

    void  pop (void);

    void  push (int newdata);
};

#endif  // MY_STACK_H
```


my_stack.cpp

```
#include "my_stack.h"

void  my_stack::pop (void)
{
  stacknode  *current = stack;
  stack = current->next;
  delete current;
}

void  my_stack::push (int newdata)
{
  stacknode  *newnode = new stacknode;
  newnode->thedata = newdata;
  newnode->next = stack;
  stack = newnode;
}
```

my_stack_templ.h (1/3)

```
#ifndef MY_STACK_TEMPL_H
#define MY_STACK_TEMPL_H

template <class TemplType>
class  my_stack_templ  {
  private:
    struct stacknode {
      TemplType  thedata;
      struct stacknode  *next;
    };

    stacknode  *stack;


  public:
    my_stack_templ (void)
     : stack(0)  {  }
```


my_stack_templ.h (2/3)

```
    ~my_stack_templ (void)
    {
      while (stack != 0)
          pop();
    }

    TemplType&  top (void)
    {
        return  stack->thedata;
    }

    void  pop (void);

    void  push (TemplType& newdata);
};
```

my_stack_templ.h (3/3)

```
template <class TemplType>
void  my_stack_templ<TemplType>::pop (void)
{
  stacknode  *current = stack;
  stack = current->next;
  delete current;
}

template <class TemplType>
void  my_stack_templ<TemplType>::push (TemplType& newdata)
{
  stacknode  *newnode = new stacknode;
  newnode->thedata = newdata;
  newnode->next = stack;
  stack = newnode;
}

#endif  // MY_STACK_TEMPL_H
```


Υλοποιήσεις στοίβας: Σχόλια (1/2)

- Το σώμα μίας template συνάρτησης μπορεί να ορισθεί και σε ένα αρχείο επικεφαλίδας. Ο λόγος είναι ότι ο μεταγλωττιστής χρειάζεται να γνωρίζει όχι μόνο τη δήλωση, αλλά και τον ορισμό μιας τέτοιας συνάρτησης, για να φτιάχνει κάθε φορά διαφορετικές «εκδόσεις» της (σε γλώσσα μηχανής), ανάλογα με τον τύπο δεδομένων για τον οποίο καλείται.
- Μπορούμε να ορίζουμε δομές (π.χ. την `struct stacknode`) –ακόμα και άλλες κλάσεις– εντός ενός ορισμού μιας κλάσης. Αυτό γίνεται για την καλύτερη οργάνωση του κώδικα, δηλαδή για να δηλώνεται και να ορίζεται καθετί στο επίπεδο που χρειάζεται και για να μην υπάρχει (αχρείαστη) καθολική πρόσβαση στην οποιαδήποτε δήλωση.

Υλοποιήσεις στοίβας: Σχόλια (2/2)

- Αντίθετα από την `my_stack::top()`, η `my_stack_templ::top()` επιστρέφει *αναφορά* (reference) στο αντικείμενο που βρίσκεται στην κορυφή της στοίβας. Ο λόγος που γίνεται αυτό είναι για να μην καλείται ο copy-constructor του τύπου δεδομένων που περιέχεται στη στοίβα. Για την κλάση `my_stack` δεν υπάρχει αντίστοιχο πρόβλημα, αφού γνωρίζουμε εκ των προτέρων ότι ο τύπος δεδομένων που περιέχεται σε αυτήν τη στοίβα είναι *ακέραιος* (`int`). Η ανάθεση ενός ακεραίου σε έναν άλλο είναι προφανώς μία πράξη που δεν κοστίζει.

Νέα περιγραφή προβλήματος (με περισσότερα ζητούμενα)

Σε αυτό το σημείο, θα κάνουμε αλλαγές και προσθήκες στον κώδικα που έχει ήδη παρουσιασθεί, έτσι ώστε –μεταξύ των άλλων– να καλυφθούν και τα εξής ζητούμενα:

- Κατάρτιση ωρολογίου προγράμματος για *εβδομάδα*.
- Ένα μάθημα τώρα θα μπορεί να αποτελείται από πολλές παραδόσεις. Δύο παραδόσεις του ιδίου μαθήματος δεν μπορούν να γίνονται την ίδια ημέρα.
- Κάθε παράδοση έχει συγκεκριμένη διάρκεια σε ώρες.
- Απομόνωση των οντοτήτων εκείνων που αφορούν καθαρά το αλγοριθμικό κομμάτι της κατάρτισης ωρολογίου προγράμματος. Παρακάτω θα δούμε πώς θα αντιμετωπίσουμε αυτό το ζητούμενο, αναφορικά με την κλάση για τους καθηγητές/ριες.

Απομόνωση κλάσεων ωρολογίου προγράμματος (1/2)

- Η κλάση `CProfessor` αναλύεται σε `CProfessor` και `CProfessorSched`.
- Στη `CProfessorSched` περιέχονται τα γνωρίσματα εκείνα ενός καθηγητή, που χρειάζεται να ξέρει ένα σύστημα κατάρτισης ωρολογίου προγράμματος όπως το χαρακτηριστικό `MaxHoursPerDay`, που είναι οι ώρες ανά ημέρα που μπορεί το πολύ να κάνει μάθημα ένας καθηγητής.[1]
- Από την άλλη, στη `CProfessor` (η οποία κληρονομεί τα χαρακτηριστικά της `CProfessorSched`) περιέχονται γνωρίσματα ενός καθηγητή όπως το όνομά του, που δεν είναι απαραίτητο να τα ξέρει ένας αλγόριθμος κατάρτισης ωρολογίου προγράμματος.

[1] Π.χ. στη `main()` που ακολουθεί ορίζεται ότι ο Einstein μπορεί να διδάξει το πολύ 2 ώρες/ημέρα, λόγω π.χ. σύμβασης.

Απομόνωση κλάσεων ωρολογίου προγράμματος (2/2)

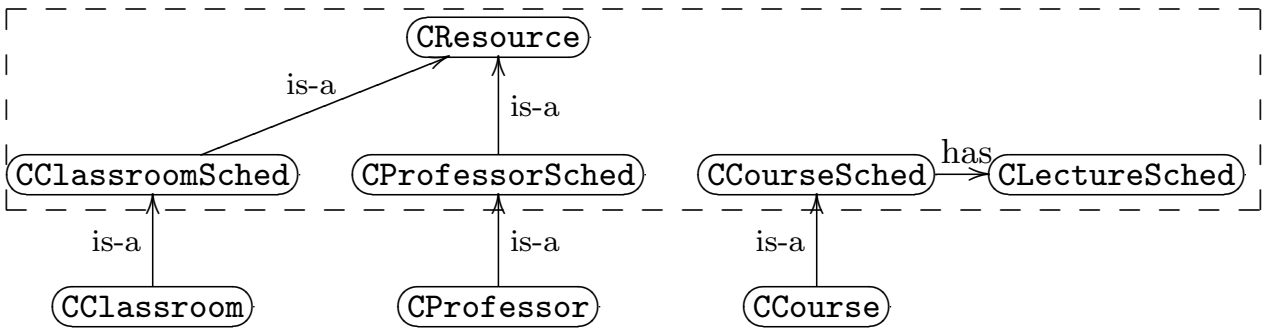
- Αφού αναλύσαμε την αρχική κλάση, σε `CProfessor` και `CProfessorSched`, γίνεται τώρα να προχωράμε σε αλλαγές στην υλοποίηση της `CProfessorSched`, χωρίς να μας ενδιαφέρει η «εσωτερική» υλοποίηση της `CProfessor` και αντίστροφα.

Και οι υπόλοιπες οντότητες του προβλήματος αντιμετωπίζονται με ανάλογο τρόπο.

Σχέσεις μεταξύ των κλάσεων

Τέλος, κάθε μάθημα (CCourseSched) *περιλαμβάνει* (has) έναν συγκεκριμένο αριθμό παραδόσεων.

Οι σχέσεις κληρονομικότητας και σύνθεσης μεταξύ των κλάσεων φαίνονται στο παρακάτω σχήμα. Στο πλαίσιο με τη διακεκομμένη γραμμή εντάσσονται οι κλάσεις που αφορούν άμεσα το σύστημα κατάρτισης ωρολογίου προγράμματος (αλγοριθμικό μέρος).



timetableV2.h (1/16)

```
#ifndef TIMETABLEV2_H
#define TIMETABLEV2_H

#include <string>
#include <iostream>


namespace timetable  {

 extern int  ndays;    // days  of the timetable
 extern int  nhours;   // hours of the timetable

 class  CResource  {
   private:
     bool  **slot;
     //  slot[d][h]==true means that the resource is unavailable on
     //   day ‘d’ and hour ‘h’.
```

timetableV2.h (2/16)

```
public:
  CResource (void)
  {
    int  d, h;
    slot = new bool*[ndays];
    for (d=0;  d<ndays;  ++d)  {
      slot[d] = new bool[nhours];
      for (h=0;  h<nhours;  ++h)
        slot[d][h] = false;
      //  initially the resource is always available
    }
  }

  ~CResource (void)
  {
    for (int d=0;  d<ndays;  ++d)
       delete [] slot[d];
    delete [] slot;
  }
```


timetableV2.h (3/16)

```
  void  reserve (int day, int hour, int dur=1)
  //  reserve the resource for 'dur' hours
  //  starting at ('day','hour')
  {
    while (dur-- > 0)
      slot[day][hour++] = true;
  }

  void  release (int day, int hour, int dur=1)
  //  release the resource for 'dur' hours
  //  starting at ('day','hour')
  {
    while (dur-- > 0)
      slot[day][hour++] = false;
  }
```

timetableV2.h (4/16)

```
    bool  is_free (int day, int hour, int dur=1)  const
    //  check whether the resource is availalbe
    //  for ‘dur’ hours, starting at (‘day’,‘hour’)
    {
      if (hour + dur > nhours)
            return false;
      while (dur-- > 0)  {
        if (slot[day][hour++] == true)
              return false;
      }
      return true;
    }
};
```


timetableV2.h (5/16)

```
class CClassroomSched : public CResource  {
  private:

  public:
    CClassroomSched (void)  {   }
};
```

timetableV2.h (6/16)

```
class CProfessorSched : public CResource  {
  private:
    int  MaxHoursPerDay;

  public:
    CProfessorSched (int MaxHoursPerDay_init=0)
     : MaxHoursPerDay(MaxHoursPerDay_init)  {   }

    bool  is_free (int day, int hour, int dur=1)  const
    {
      if (MaxHoursPerDay != 0)  {
        int  HoursPerDay=0;
        for (int h=0;  h<nhours;  ++h)
             HoursPerDay += !CResource::is_free(day,h);
        if (HoursPerDay + dur > MaxHoursPerDay)
             return  false;
      }
      return  CResource::is_free(day,hour,dur);
    }
};
```


timetableV2.h (7/16)

```
class  CLectureSched  {
  private:
    CClassroomSched&  classroom;
    CProfessorSched&  professor;
    int  duration;
    int   day_scheduled;
    int  hour_scheduled;
    static const int  EMPTY = -1;
    //  hour_scheduled==EMPTY means that
    //   the course has not been scheduled.

  public:
    CLectureSched (CClassroomSched& cl_init, CProfessorSched& pr_init,
                   int duration_init)
     : classroom(cl_init), professor(pr_init),
       duration(duration_init),
       day_scheduled(EMPTY), hour_scheduled(EMPTY)  {   }

    int  get_duration (void)  const
     {  return  duration;  }
```

timetableV2.h (8/16)

```
int  get_day  (void)  const
 {  return   day_scheduled;  }

int  get_hour (void)  const
 {  return  hour_scheduled;  }

bool  schedule (int day, int hour)
{
  if (classroom.is_free(day,hour,duration)
      && professor.is_free(day,hour,duration))  {
        classroom.reserve(day,hour,duration);
        professor.reserve(day,hour,duration);
        day_scheduled  = day;
        hour_scheduled = hour;
        return  true;
  } else  {
        return  false;
  }
}
```


timetableV2.h (9/16)

```
void  cancel_schedule (void)
{
  classroom.release(day_scheduled,hour_scheduled,duration);
  professor.release(day_scheduled,hour_scheduled,duration);
  day_scheduled  = EMPTY;
  hour_scheduled = EMPTY;
}
};
```

timetableV2.h (10/16)

```
class  CCourseSched  {
  private:
    CLectureSched  **Lecture;
    int  nLecture;

  public:
    CCourseSched (CClassroomSched& cl_init, CProfessorSched& pr_init,
                  int duration[], int nLecture_init)
     : nLecture(nLecture_init)
     {
        Lecture = new CLectureSched*[nLecture];
        for (int l=0;  l<nLecture; ++l)  {
             Lecture[l] = new CLectureSched(cl_init,pr_init,
                                            duration[l]);
        }
     }
```


timetableV2.h (11/16)

```
    ~CCourseSched (void)
    {
      for (int l=0;  l<nLecture; ++l)
           delete Lecture[l];
      delete [] Lecture;
    }

    int  get_nLecture (void)  const
     {  return  nLecture;  }

    const CLectureSched&
    getLecture (int LectIndex)  const
     {  return  *Lecture[LectIndex];  }

    bool  schedule (int lectIndex, int day, int hour)
    {
      for (int l=0;  l<nLecture;  ++l)  {
           if (l!=lectIndex  &&  Lecture[l]->get_day()==day)
               return  false;
      }
      return  Lecture[lectIndex]->schedule(day,hour);
    }
```

timetableV2.h (12/16)

```
    void  cancel_schedule (int LectIndex)
    {
      Lecture[LectIndex]->cancel_schedule();
    }
};
```


timetableV2.h (13/16)

```
//  The following classes inherit the behaviour of the
//   corresponding *Sched classes

class CClassroom : public CClassroomSched  {
  private:
    const  int  id;
    static int  count;
    std::string  name;

  public:
    CClassroom (std::string name_init)
     : id(++count), name(name_init)  {   }

    friend std::ostream&
    operator << (std::ostream& os, const CClassroom& room);

    int  get_id (void)  const
     {  return  id;  }
};
```

timetableV2.h (14/16)

```
class CProfessor : public CProfessorSched  {
  private:
    const  int  id;
    static int  count;
    std::string  name;

  public:
    CProfessor (std::string name_init, int MaxHoursPerDay_init=0)
     : CProfessorSched(MaxHoursPerDay_init), id(++count),
       name(name_init)  {   }

    friend std::ostream&
    operator << (std::ostream& os, const CProfessor& prof);

    int  get_id (void)  const
     {  return  id;  }
};
```


timetableV2.h (15/16)

```
class CCourse : public CCourseSched  {
  private:
    std::string  name;
    std::string  code;
    CClassroom&  classroom;
    CProfessor&  professor;

  public:
    CCourse (std::string name_init, std::string code_init,
             CClassroom& cl_init, CProfessor& pr_init,
             int duration[], int nLecture_init)
     : CCourseSched (cl_init,pr_init, duration, nLecture_init),
       name(name_init), code(code_init),
       classroom(cl_init), professor(pr_init)  {   }

    friend std::ostream&
    operator << (std::ostream& os, const CCourse& les);

    CClassroom&  get_class (void)  const
     {  return  classroom;  }
```

timetableV2.h (16/16)

```
        CProfessor&  get_prof (void)  const
         {  return  professor;  }
    };

    bool
    solve_rec (CCourse **course, int ncourse);

    void
    print_timetable (CCourse **course, int ncourse,
          CClassroom *room, int nroom);
} // end namespace

#endif  // TIMETABLEV2_H
```


timetableV2.cpp (1/8)

```
#include "timetableV2.h"
using namespace timetable;
using namespace std;

//  Initializations of the counters of the instances
//   of the classes
int  CClassroom::count = 0;
int  CProfessor::count = 0;
```

timetableV2.cpp (2/8)

```
//  Overloaded operators for printing
ostream& timetable::operator <<
  (ostream& os, const CClassroom& room)
{
  os << room.name;
  return  os;
}

ostream& timetable::operator <<
  (ostream& os, const CProfessor& prof)
{
  os << prof.name;
  return  os;
}

ostream& timetable::operator <<
  (ostream& os, const CCourse& c)
{
  os << c.name;
  return  os;
}
```


timetableV2.cpp (3/8)

```
namespace {
  bool
  solve_rec_body (CCourse **course, int ncourse,
                  int curr_course, int curr_lect)
  {
    if (curr_lect   == course[curr_course]->get_nLecture())  {
      ++curr_course;  // every lecture has been scheduled, so...
      curr_lect = 0;  //  we continue to the next course
    }
    if (curr_course == ncourse)
      return  true;   // success: every course has been scheduled

    //  variable renaming (for readability)
    CCourse&  c = *course[curr_course];
```

timetableV2.cpp (4/8)

```
        int  d, h;
        for (d=0;  d<ndays;  ++d)  {
          for (h=0;  h<nhours;  ++h)  {
            if (c.schedule(curr_lect,d,h) == true)  {
              if (solve_rec_body(course,ncourse,
                   curr_course,curr_lect+1) == true)
                       return  true;
              c.cancel_schedule(curr_lect);
            }
          }
        }
        return  false;   // failed to find an appropriate hour
      }
    } // end namespace

    //  Recursive search for a solution (timetable)
    bool
    timetable::solve_rec (CCourse **course, int ncourse)
    {
      return  solve_rec_body(course,ncourse,0,0);
    }
```


timetableV2.cpp (5/8)

```
    #include <iostream>
    #include <iomanip>

    namespace {
      //  Serial search for the course scheduled to take place
      //   in room 'room' on (day,hour)
      int  search_course (CCourse **course, int ncourse,
                          CClassroom& room, int day, int hour)
      {
        int  c, l;
        for (c=0;  c<ncourse;  ++c)  {
          for (l=0;  l < course[c]->get_nLecture();  ++l)  {
            if (course[c]->get_class().get_id() == room.get_id()
                && course[c]->getLecture(l).get_day()  == day
                && course[c]->getLecture(l).get_hour() <= hour
                && hour <   course[c]->getLecture(l).get_hour()
                          + course[c]->getLecture(l).get_duration())
                    return  c;
          }
        }
        return  -1;
      }
```

timetableV2.cpp (6/8)

```
  inline void
  print_bar (int columns)
  {
    for (int i=0;  i<columns;  ++i)
      cout << "+------------";
    cout << "+\n";
  }
} // end namespace

void
timetable::print_timetable (CCourse **course, int ncourse,
                            CClassroom *room, int nroom)
{
  int  i, d, h, r, c;
```


timetableV2.cpp (7/8)

```
  for (d=0;  d<ndays;  ++d)  {
    //  PRINT DAY
    cout << "\n\n+";
    for (i=0;  i<nroom-1;  ++i)
      cout << "-------------";
    cout << "------------+\n";
    cout << "| " << "DAY "
         << setiosflags(ios::left) << setw(nroom*13-7) << d+1
         << " |\n";
    print_bar(nroom);

    //  PRINT ROOM
    for (r=0;  r<nroom;  ++r)  {
      cout << "| ";
      cout << setiosflags(ios::left) << setw(10) << room[r] << " ";
    }
    cout << "|\n";
    print_bar(nroom);
```

timetableV2.cpp (8/8)

```
    //  PRINT COURSE
    for (h=0;  h<nhours;  ++h)  {
      for (r=0;  r<nroom;  ++r)  {
        c = search_course(course,ncourse,room[r],d,h);
        cout << "| ";
        cout << setiosflags(ios::left) << setw(10);
        if (c == -1)
                cout << "";
        else
                cout << *course[c];
        cout << " ";
      }
      cout << "|\n";
    }
    print_bar(nroom);
  }
}
```


mainV2.cpp (1/2)

```
#include "timetableV2.h"
using namespace timetable;
#include <iostream>
using namespace std;

int  timetable::ndays  = 3;   // days  of the timetable
int  timetable::nhours = 4;   // hours of the timetable

int  main (void)
{
  CClassroom  room[] = {
    CClassroom("C1"),
    CClassroom("C2")  };
  CProfessor  prof[] = {
    CProfessor("Einstein", 2),
    CProfessor("Godel"),
    CProfessor("Eco")        };

  int  *dur = new int[ndays];
  CCourse  *course[6];
```

mainV2.cpp (2/2)

```
    dur[0]=2;   dur[1]=2;
    course[0] = new CCourse("Physics",   "P1",room[0],prof[0],dur,2);
    dur[0]=2;
    course[1] = new CCourse("Physics",   "P1",room[1],prof[0],dur,1);
    dur[0]=3;   dur[1]=2;
    course[2] = new CCourse("Maths",     "P2",room[0],prof[1],dur,2);
    dur[0]=2;   dur[1]=2;
    course[3] = new CCourse("Maths",     "P2",room[1],prof[1],dur,2);
    dur[0]=2;
    course[4] = new CCourse("Philosophy","T1",room[0],prof[2],dur,1);
    dur[0]=2;   dur[1]=2;
    course[5] = new CCourse("Philosophy","T1",room[1],prof[2],dur,2);

    if (solve_rec(course,6) == true)  {
       cout << "Found a solution\n";
       print_timetable(course, 6, room, 2);
    } else  {
       cout << "Couldn't find a solution\n";
    }
}
```


Αποτελέσματα (1/2)

```
% g++ -o timetableV2 timetableV2.cpp mainV2.cpp
% ./timetableV2
Found a solution


+-------------------------+
| DAY 1                   |
+------------+------------+
| C1         | C2         |
+------------+------------+
| Physics    | Maths      |
| Physics    | Maths      |
| Maths      | Philosophy |
| Maths      | Philosophy |
+------------+------------+
```

Αποτελέσματα (2/2)

```
+-------------------------+
| DAY 2                   |
+------------+------------+
| C1         | C2         |
+------------+------------+
| Physics    | Maths      |
| Physics    | Maths      |
| Philosophy |            |
| Philosophy |            |
+------------+------------+


+-------------------------+
| DAY 3                   |
+------------+------------+
| C1         | C2         |
+------------+------------+
| Maths      | Physics    |
| Maths      | Physics    |
| Maths      | Philosophy |
|            | Philosophy |
+------------+------------+
%
```


Ευχαριστίες

- στην κα *Ιζαμπώ Καράλη* για τη βοήθειά της στη δημιουργία και οργάνωση αυτής της παρουσίασης.
- στον κο *Παναγιώτη Σταματόπουλο* και στον *Κυριάκο Ζερβουδάκη*. Οι εργασίες τους πάνω στην κατάρτιση ωρολογίων προγραμμάτων αποτέλεσαν το υπόβαθρο της πτυχιακής μου εργασίας, πάνω στην οποία βασίστηκε αυτή η παρουσίαση.
- και στον *Στέφανο Σταμάτη* για τον κώδικα υλοποίησης μιας στοίβας που είχε γράψει σε C και τον οποίο δανείστηκα.