ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
UNIVERSITY OF MACEDONIA

Xinogalos Stelios

Lecturer , Department of Technology Management

University of Macedonia, Greece

stelios@uom.gr

June 11-13, Novi Sad, Serbia

Research group

Professor **Satratzemi Maya**
Department of Applied Informatics, University of Macedonia

Educational Technology
Programming Environments and Techniques
Adaptive Educational Hypermedia LMSs
Distributed Pair Programming

Associate Professor **Dagdilelis Vassilios**
Department of Educational and Social Policy

Educational Technology
Didactics of Informatics

EDUCATIONAL PROGRAMMING ENVIRONMENTS - review, evaluation, design, implementation (objectKarel, Karel)

1998 - today

Motivation for this research (back to 1998)

The OOP paradigm seemed to have prevailed and its teaching was considered necessary.

The research that had been carried out and the educational tools that had been developed referred mainly to procedural programming.

The classic approach to teaching programming, which is based on

- using a general purpose programming language,
- using a professional programming environment,
- solving problems –mainly- about number and symbol processing

was/is not compliant with students' didactical needs.

There was a need to design and develop a new **educational** programming environment for the introduction to **object-oriented programming**.

Research regarding Novice Programmers

In order to design and implement the new environment research was carried out with the aim of

- reviewing **students' difficulties**, **errors** and **misconceptions** when introduced to programming
- categorizing **approaches to teaching programming** and the relevant **educational tools**
- categorizing the most important **design principles for integrated novice programming environments**

Teaching approaches

Classic teaching approach:

- use of a general purpose programming language,
- use of a professional programming environment,
- solving number and symbol processing problems

poses several didactical problems

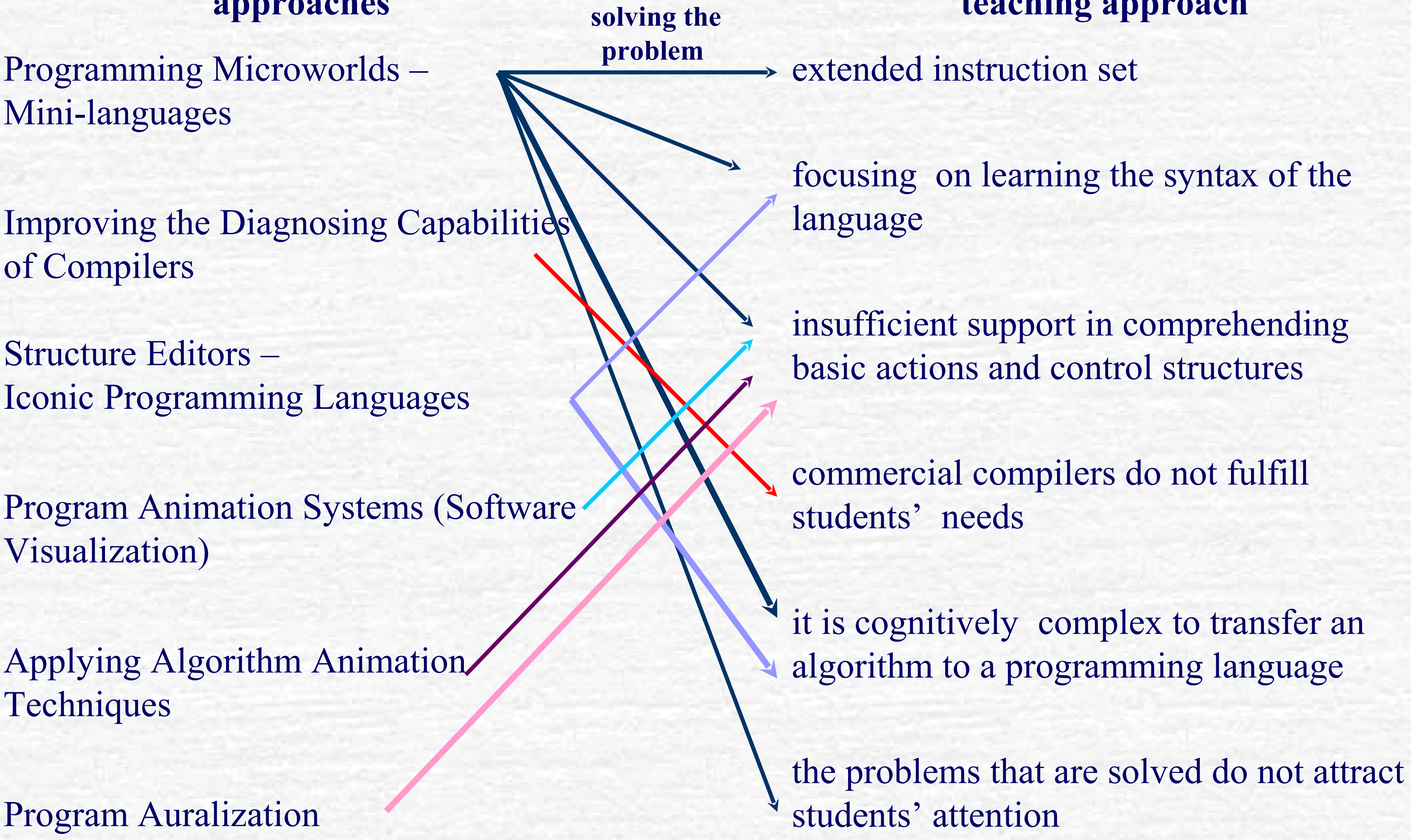
Alternative teaching approaches:

- Programming microworlds – mini-languages
- Improving the Diagnosing Capabilities of Compilers
- Structure Editors – Iconic Programming Languages
- Program Animation Systems (software visualization)
- Applying Algorithm Animation Techniques
- Software Auralization (software visualization with use of sound)

aim at dealing with the didactical problems of the classic teaching approach. However, each approach focuses on solving a specific or some of the problems considered by its proponents as the most important.

Alternative teaching approaches

Didactical problems of the classic teaching approach



Programming Microworlds – Mini-languages

Improving the Diagnosing Capabilities of Compilers

Structure Editors – Iconic Programming Languages

Program Animation Systems (Software Visualization)

Applying Algorithm Animation Techniques

Program Auralization

extended instruction set

focusing on learning the syntax of the language

insufficient support in comprehending basic actions and control structures

commercial compilers do not fulfill students' needs

it is cognitively complex to transfer an algorithm to a programming language

the problems that are solved do not attract students' attention

Educational tools

Programming Microworlds – Mini-languages:

CARISTO, Boxer, LOGOMotion, StarLogo, MultiLogo, SuperLogo

BPS Robot Simulator, Karel The Robot, Karel++ for Windows, Karel-3D, Marta, Darel

RoboPascal for Delphi, Knobby's World, Karel Genie, JKarelRobot, Jeroo, Karel J. Robot, Alice

Compilers with Improved Diagnosing Capabilities: THETIS, CAP

Structure Editors – Iconic Programming Languages: MacGNOME Genies, Alice Pascal, BACCII, BACII++, FLINT, KidSim/Cocoa

Program Animation Systems (Software Visualization) : DynaLab, AnimPascal, BlueJ
Applying Algorithm Animation Techniques: LENS

Program Auralization: CAITLIN, LogoMedia

Xinogalos, S. & Satratzemi, M. (2004) Introducing Novices to Programming: a review of Teaching Approaches and Educational Tools, *Proceedings of the 2nd International Conference on Education and Information Systems, Technologies and Applications (EISTA 2004),* Orlando, Florida, USA, July 21-25, Vol. 2, 60-65.

Didactical – Research Rationale behind objectKarel

- Selecting as the main teaching approach, **Programming Microworlds**:
  - it is the only approach that solves nearly all the didactical problems of the classic approach

- Incorporating the technology of **Program Animation**:
  - supports students in comprehending the semantics of programming concepts/structures and flow of control
  - supports students in debugging their programs

- Incorporating the technology of **Explanatory Visualization**:
  - provides an alternative means of visualization that might suit better to some students
  - deeper understanding of the meaning of concepts that are difficult for novices (such as control structures) and program flow of control

Didactical – Research Rationale behind objectKarel

- Although the programming language consists of a limited instruction set, we decided to incorporate a **Structure Editor**:
  - focus on concepts and developing problem solving abilities
  - teaching the material in less time
  - teaching programming concepts to smaller aged students

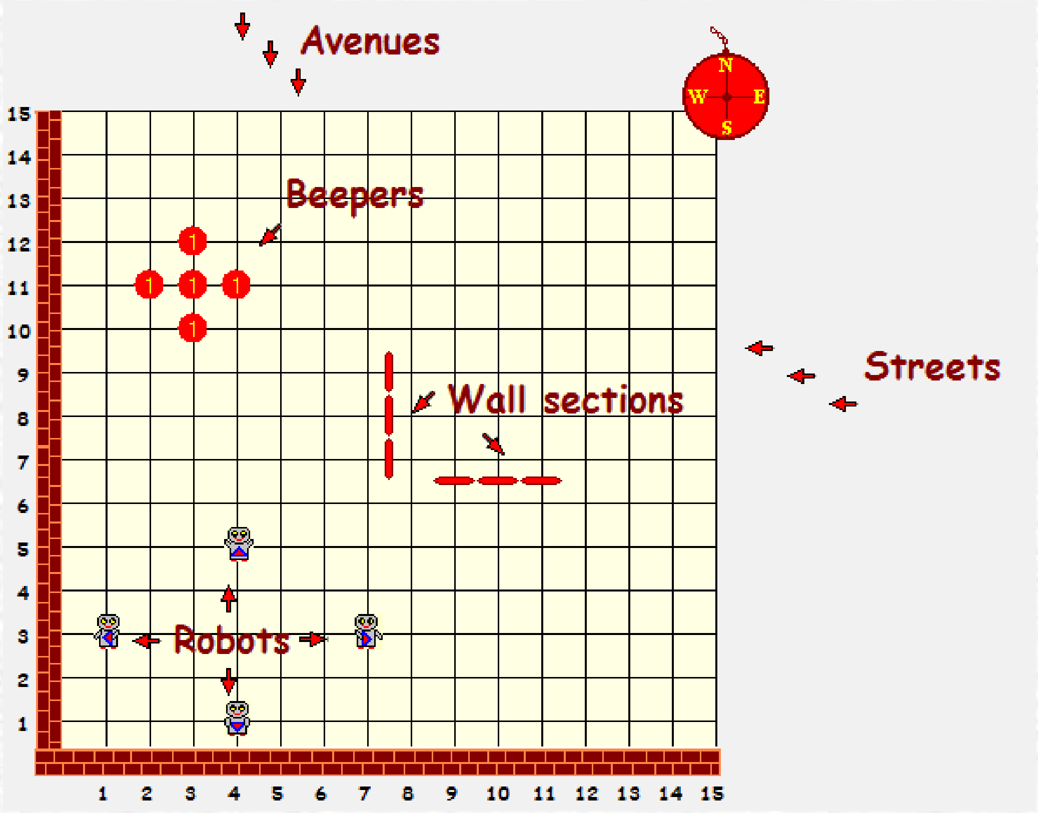
- **Reporting user-friendly error messages**:
  - revealing the misconceptions that usually account for the errors

- Incorporating a **Series of Lessons** containing theory and hands-on activities:
  - teaching the material in less time
  - familiarizing students with concepts before implementing programs

- **History of compilations**

Xinogalos, S. & Satratzemi, M. (2002) An Integrated Programming Environment for Teaching the Object-Oriented Programming Paradigm, *Lecture Notes in Computer Science (LNCS), 2510, Shafazand H A Min Tjoa (Eds.), Springer Verlag,* 544-551.

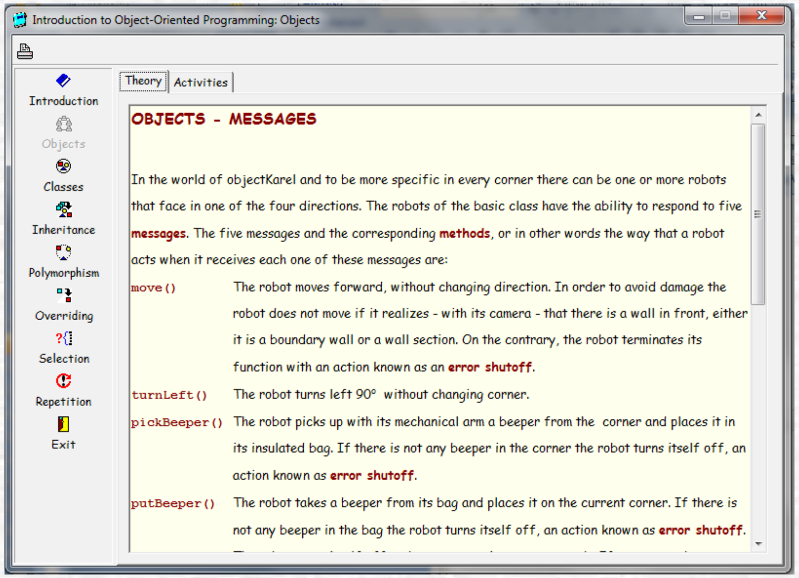
The microworld objectKarel



- is available from the home page of the creator of Karel++, Prof. Joseph Bergin: http://csis.pace.edu/~bergin/temp/findkarel.html

- was a finalist at the *“2nd International Competition of non-commercial Software Systems, Tools and Products for Technology-Based Education”* at the “9th IASTED International Conference on Computers and Advanced Technology in Education”, Lima, Peru, 4-6 October, 2006

The lessons module

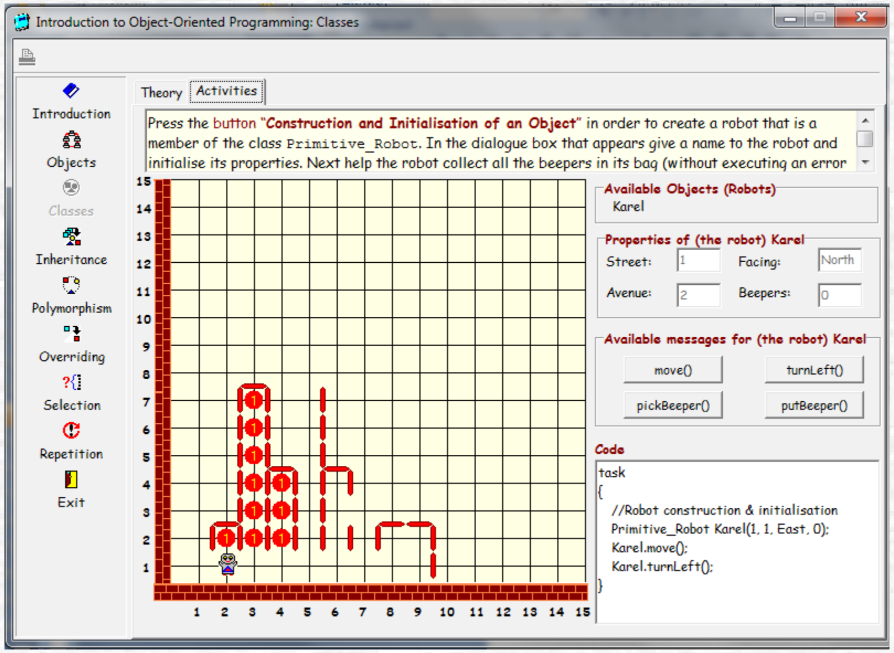


OBJECTS - MESSAGES

In the world of objectKarel and to be more specific in every corner there can be one or more robots that face in one of the four directions. The robots of the basic class have the ability to respond to five **messages**. The five messages and the corresponding **methods**, or in other words the way that a robot acts when it receives each one of these messages are:

`move()`	The robot moves forward, without changing direction. In order to avoid damage the robot does not move if it realizes - with its camera - that there is a wall in front, either it is a boundary wall or a wall section. On the contrary, the robot terminates its function with an action known as an **error shutoff**.
`turnLeft()`	The robot turns left 90° without changing corner.
`pickBeeper()`	The robot picks up with its mechanical arm a beeper from the corner and places it in its insulated bag. If there is not any beeper in the corner the robot turns itself off, an action known as **error shutoff**.
`putBeeper()`	The robot takes a beeper from its bag and places it on the current corner. If there is not any beeper in the bag the robot turns itself off, an action known as **error shutoff**.

- fundamental OOP concepts are presented prior to control structures



Press the button "**Construction and Initialisation of an Object**" in order to create a robot that is a member of the class `Primitive_Robot`. In the dialogue box that appears give a name to the robot and initialise its properties. Next help the robot collect all the beepers in its bag (without executing an error

Available Objects (Robots)
Karel

Properties of (the robot) Karel
Street: 1 Facing: North
Avenue: 2 Beepers: 0

Available messages for (the robot) Karel

Code

```
task
{
  //Robot construction & initialisation
  Primitive_Robot Karel(1, 1, East, 0);
  Karel.move();
  Karel.turnLeft();
}
```

- each activity lies in the heart of the underlying concept
- the main aim of the activities is to help students comprehend deeper the OOP concepts

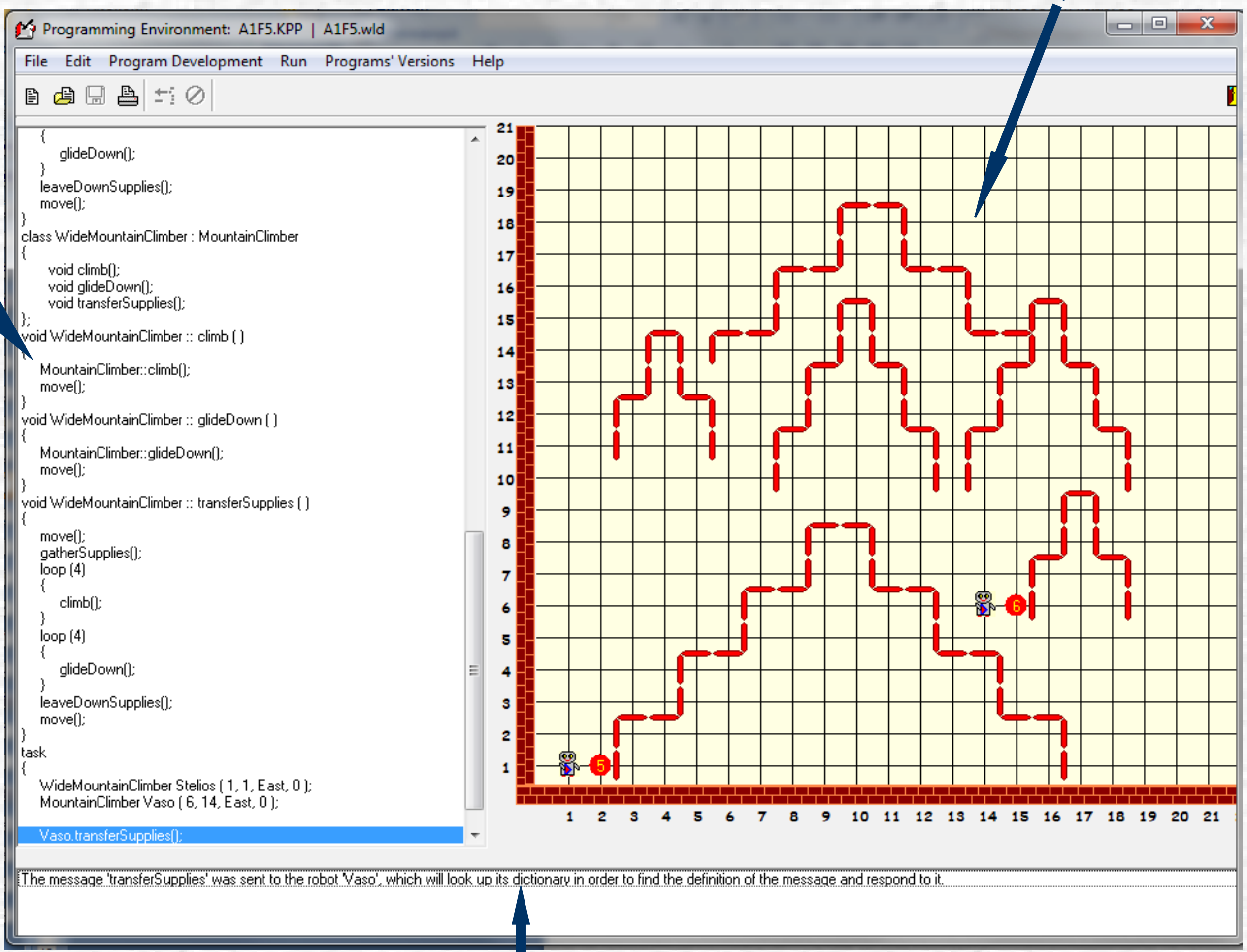
Xinogalos, S., Satratzemi, M. (2005) «Using Hands-on Activities for Motivating Students with OOP Concepts Before They Are Asked to Implement Them», *ACM SIGCSE Bulletin, Vol. 37, Number 3, September 2005,* 380.

Xinogalos, S., Satratzemi, M. (2005) «The Hands-on Activities of the Programming Microworld objectKarel», *ACM SIGCSE Bulletin, Vol. 37, Number 3, September 2005,* 384.

The main window of objectKarel

Visualization of Program Execution

Source Code

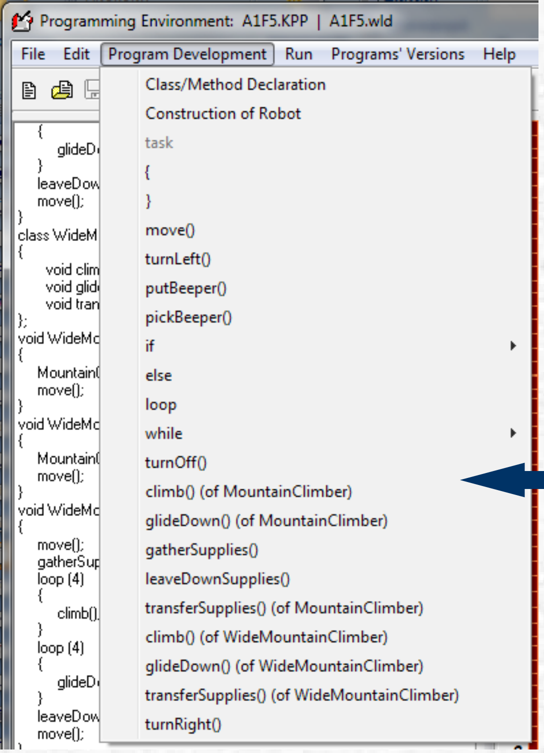

Programming Environment: A1F5.KPP | A1F5.wld
File Edit Program Development Run Programs' Versions Help

```
    {
        glideDown();
    }
    leaveDownSupplies();
    move();
}
class WideMountainClimber : MountainClimber
{
     void climb();
     void glideDown();
     void transferSupplies();
};
void WideMountainClimber :: climb ( )
{
    MountainClimber::climb();
    move();
}
void WideMountainClimber :: glideDown ( )
{
    MountainClimber::glideDown();
    move();
}
void WideMountainClimber :: transferSupplies ( )
{
    move();
    gatherSupplies();
    loop (4)
    {
        climb();
    }
    loop (4)
    {
        glideDown();
    }
    leaveDownSupplies();
    move();
}
task
{
    WideMountainClimber Stelios ( 1, 1, East, 0 );
    MountainClimber Vaso ( 6, 14, East, 0 );

    Vaso.transferSupplies();
```

```
The message 'transferSupplies' was sent to the robot 'Vaso', which will look up its dictionary in order to find the definition of the message and respond to it.
```

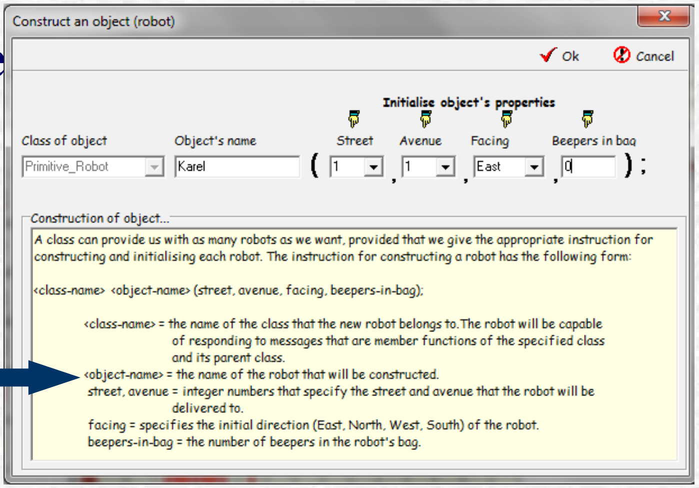
Compiler Output /Explanatory Visualization


Programming Environment: A1F5.KPP | A1F5.wld
File
Edit
Program Development
Run
Programs' Versions
Help
Class/Method Declaration
Construction of Robot
task
{
}
move()
turnLeft()
putBeeper()
pickBeeper()
if
else
loop
while
turnOff()
climb() (of MountainClimber)
glideDown() (of MountainClimber)
gatherSupplies()
leaveDownSupplies()
transferSupplies() (of MountainClimber)
climb() (of WideMountainClimber)
glideDown() (of WideMountainClimber)
transferSupplies() (of WideMountainClimber)
turnRight()

```
  {
      glideD
  }
  leaveDow
  move();
}
class WideM
{
    void clim
    void glid
    void tran
};
void WideMo
{
  Mountain(
  move();
}
void WideMo
{
  Mountain(
  move();
}
void WideMo
{
  move();
  gatherSup
  loop (4)
  {
      climb()
  }
  loop (4)
  {
      glideD
  }
  leaveDow
  move();
```

Program Development

Programs are developed with the use of a **structure editor** consisting of a **Statements Menu** And **dialog boxes**



Primitive_Robot Karel (1 , 1 , East , 0) ;

Construction of object...

A class can provide us with as many robots as we want, provided that we give the appropriate instruction for constructing and initialising each robot. The instruction for constructing a robot has the following form:

<class-name> <object-name> (street, avenue, facing, beepers-in-bag);

<class-name> = the name of the class that the new robot belongs to. The robot will be capable of responding to messages that are member functions of the specified class and its parent class.
<object-name> = the name of the robot that will be constructed.
street, avenue = integer numbers that specify the street and avenue that the robot will be delivered to.
facing = specifies the initial direction (East, North, West, South) of the robot.
beepers-in-bag = the number of beepers in the robot's bag.

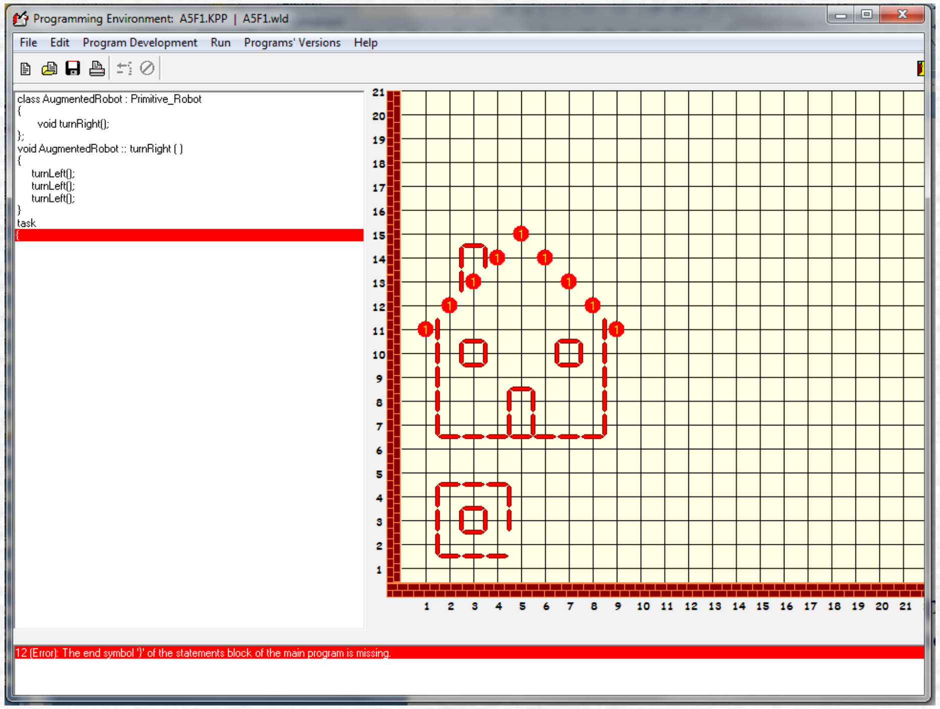
Features of the structure editor:

- the menu is automatically updated
- the source code is auto-indented

Differences in comparison with a typical structure editor:

- it allows the violation of some syntactic rules, in order to allow making changes easier
- semantic and logic errors are reported during compilation in order to familiarize students with error messages regarding the comprehension of basic concepts and the debugging process

Compilation of Programs


Programming Environment: A5F1.KPP | A5F1.wld
File Edit Program Development Run Programs' Versions Help

```
class AugmentedRobot : Primitive_Robot
{
      void turnRight();
};
void AugmentedRobot :: turnRight ( )
{
    turnLeft();
    turnLeft();
    turnLeft();
}
task
{
```

```
12 (Error): The end symbol '}' of the statements block of the main program is missing.
```

User-friendly error messages:

- in Greek (or English)
- using natural language (no codes)
- reporting the actual line of the error
- reporting the source of the error

Interaction with the results of the compilation:

- students focus on the right line of the source code

Program execution

Execution choices:

1. Running
2. Tracing
3. Step by step execution

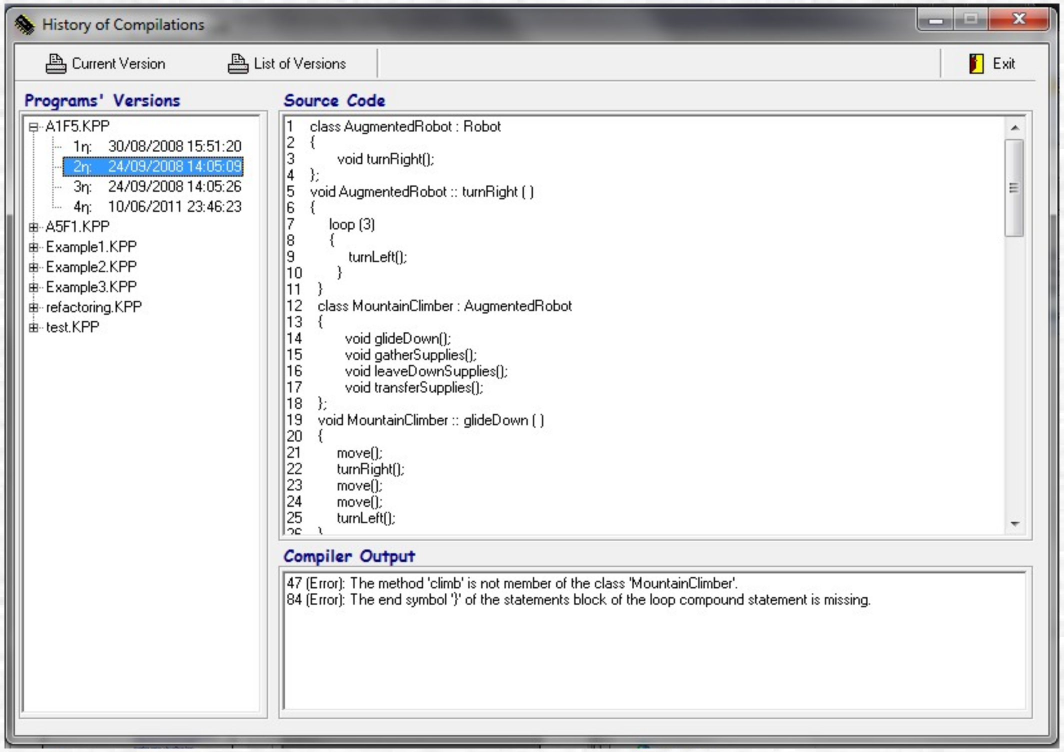
Program animation

when a program is executed with one of these ways, explanations regarding the semantics of the current statement are presented in natural language (**explanatory visualization**)

Explanatory visualization:

- ➤offers an alternative means of presenting the semantics of concepts/structures and flow of control during program execution
- ➤Helps in comprehending – dealing with misconceptions about concepts/structures that are difficult for students

History of Compilations



```
A1F5.KPP
    1η:  30/08/2008 15:51:20
    2η:  24/09/2008 14:05:09
    3η:  24/09/2008 14:05:26
    4η:  10/06/2011 23:46:23
A5F1.KPP
Example1.KPP
Example2.KPP
Example3.KPP
refactoring.KPP
test.KPP
```

```
class AugmentedRobot : Robot
{
     void turnRight();
};
void AugmentedRobot :: turnRight ( )
{
   loop (3)
   {
      turnLeft();
   }
}
 class MountainClimber : AugmentedRobot
 {
     void glideDown();
     void gatherSupplies();
     void leaveDownSupplies();
     void transferSupplies();
 };
 void MountainClimber :: glideDown ( )
 {
    move();
    turnRight();
    move();
    move();
    turnLeft();
```

```
47 (Error): The method 'climb' is not member of the class 'MountainClimber'.
84 (Error): The end symbol '}' of the statements block of the loop compound statement is missing.
```

- each time a program is compiled (or saved) it is saved along with the compilation results
- the history of compilations is presented with the form of a two level tree

Benefits:

- recording students' difficulties-misconceptions when introduced to OOP programming
- studying students' problem solving techniques
- helps the teacher adjust the lesson to students' needs

Pilot Use and Evaluation of objectKarel

Participants:

- 19 undergraduate students of the Department of Applied Informatics (working in groups of two)
- conditions for participating:
  1) students should have difficulties in programming (had failed the exams)
  2) students had to attend six 2-hour lessons, to fill-in two questionnaires at the beginning and at the end of the lessons
- «reward»: credits for the course «Introduction to Programming»

Material:

- teaching took place entirely in the lab using the environment of objectKarel
- Camtasia Recorder was used for recording (video) two of the groups

Xinogalos, S., Satratzemi, M. & Dagdilelis, V. (2006) An Introduction to object-oriented programming with a didactic microworld: objectKarel, *Computers & Education*, Volume 47, Issue 2, September 2006, 148-171, Elsevier Publishers.

Description *(continued…)*

The design of the study:

- Total number of lessons: 6
- Duration of each lesson: 2 hours
- Frequency: weekly

Lesson content	**Didactical aims**
Objects- Classes	Familiarization with: 1) the concepts object, message/method, class, 2) sequential structure and 3) the programming environment
Inheritance	1) Comprehending the concept of inheritance, the advantages of creating new classes and reusability and 2) Familiarization with declaring new classes and defining methods
Polymorphism & Overriding	Comprehending the concepts of polymorphism and overriding
Selection Structures	1) Comprehending the syntax and semantics of the selection structures if, if/else nesting them, and 2) Reference to defining predicates
Repetition structures	1) Comprehending the repetitive structure with predefined number of executions and the repetitive execution under condition, and 2) Selecting the most appropriate structure for each case
Evaluation	Evaluation of students' knowledge regarding the basic OOP concepts – basic control structures

Lesson structure – Data collection

Each one of the first 5 lessons consisted of 3 phases:
1st phase: teaching with the theory incorporated in the environment
2nd phase: using the hands-on activities incorporated in the environment
3rd phase: ***the groups of students solved one or two problems using objectKarel***

For each one of the 5 lessons:

- the instructor recorded difficulties with comprehending the taught concepts – using the programming environment
- the consecutive versions of students programs (history of compilations) were thoroughly studied
- the recorded actions and the dialogs of 2 groups of students were analyzed

In the 6th lesson students:

- ***debugged a program*** that contained all types of errors
- ***answered (in person) in open type questions***

Answered a questionnaire for the evaluation of the programming environment and the incorporated series of lessons.

Conclusions

The overall design of the lessons was successfully applied:

- the general didactical aims of the 5 lessons were achieved in a high degree
- students faced difficulties with selection structures, a fact that shows that an extra lesson is needed
- some students faced difficulties with the concepts of polymorphism and overriding

Students' performance was very good:

- students comprehended the basic OOP concepts and control structures
- several misconceptions recorded in the literature regarding OOP were not recorded in our study

The programming environment in combination with the series of lessons supported students and made it possible to teach the main principles of the OOP paradigm in a short period of time.

Conclusions *(continued…)*

All the features of the programming environment helped students:

Feature	Help provided in	Percentage of students
Incorporation of lessons	Teaching Problem solving	79%
Structure editor	Program development Program editing	100% 95%
Program Animation	Debugging programs Comprehending the semantics of statements and flow of control	100%
Explanatory Visualization	Deeper understanding of how a program functions Deeper understanding of the semantics Debugging	68%
User-friendly Error Messages	Analytical explanation of syntax errors Use of simple language	90%

Further studies based on objectKarel

- Students' **problem solving techniques** were studied using data from the "history of compilations" feature of objectKarel –
  - students do not devote much time and effort in developing algorithms
  - students do not apply effective strategies for implementing algorithms

Xinogalos, S. & Satratzemi, M. (2004) Studying Novice Programmers' Attitudes in Developing and Implementing Algorithms Using an Educational Programming Environment, *Proceedings of the 10th International Conference on Information Systems Analysis and Synthesis (ISAS 2004) jointly with the International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA 2004)*, Orlando, Florida, USA, July 21-25, Vol. 1, 198-203.

Xinogalos, S. & Satratzemi, M. (2003) Students' Practices in Developing and Implementing Algorithms: An Empirical Study, *Proceedings of the 6th Hellenic European Research on Computer Mathematics and its Applications Conference (HERCMA 2003)*, Athens, 25-27 September 2003, Vol. 2, 754-763.

Further studies based on objectKarel

- The series of lessons based on objectKarel was revised based on the results of its pilot use and applied using the same methodology
  – the positive results of the first study were confirmed

Xinogalos, S., Satratzemi, M. & Dagdilelis, V. (2006), An Objects-First Approach to Teaching Object Orientation based on objectKarel, *Proceedings of the 5th WSEAS International Conference on Education and Educational Technology, 16-18 December 2006, Tenerife, Spain,*93-98.

Xinogalos, S., Satratzemi, M. & Dagdilelis, V. (2006), Teaching Fundamental Notions of Object Oriented Programming with objectKarel, *International Journal of WSEAS Transactions on Advances in Engineering Education*, *Issue 11, Vol. 3,* 1022-1029, WSEAS Press.

Further studies based on objectKarel

- **Comparing the evaluation of objectKarel's features by two groups of students with different level of experience** in programming languages and environments, and **proposing design guidelines** for programming environments:
  - Incorporate brief and concise theory and hands-on activities to the environment
  - Use a combination of a structure – text editor
  - Use explanatory visualization in combination with program animation

Xinogalos, S., Satratzemi, M. & Dagdilelis, V. (2006), Evaluating objectKarel - an educational programming environment for object oriented programming, *In A. Mendez-Vilas et al. (eds) "Current Developments in Technology-Assisted Education", vol. 2,* 821-825, Formatex press.

Further studies based on objectKarel

- **Studying students' conceptual grasp of fundamental OOP concepts in objectKarel and BlueJ**
  - students' conceptual grasp is deeper in objectKarel
  - objectKarel can not be used for teaching the real thing
  - the combined use of the two environments is proposed

Xinogalos, S., Satratzemi, M. & Dagdilelis, V. (2007), A Comparison of Two Object-Oriented Programming Environments for Novices, Proceedings *of the 10th IASTED International Conference on Computers and Advanced Technology in Education (CATE 2007), 8 -10 October 2007, Beijing, China*, 49-54.

The microworld Karel

Karel – the procedural version of objectKarel was developed in the context of *Pythagoras II-Funding of research groups in the University of Macedonia, Priority Action 2.2.3.e, Action 2.2.3, Measure 2.2, implemented within the framework of the Operational Programme "Education and Initial Vocational Training II (EPEAEK)" and co-financed by the European Union [3rd Community Support Framework, 75% financed by the European Social Fund 25% National Resources])*

Xinogalos, S. (2010), An Interactive Learning Environment for Teaching the Imperative and Object-Oriented Programming Techniques in Various Learning Contexts, In Lytras et al. (Eds.), Knowledge Management, Information Systems, E-Learning, and Sustainability Research, *Communications in Computer and Information Science*, Vol. 111, Springer-Verlag Berling Heidelberg, (presented in WSKS 2008), 512-520.

Xinogalos, S. (2011), Teaching Programming to Secondary Education Students with a Learning Environment Based on "Karel the Robot": A Pilot Study in a Greek High School, In *Horizons in Computer Science Research*, Vol. 2, Thomas S. Clary (Ed.), New York: Nova Science, 67-92.

STUDYING STUDENTS' DIFFICULTIES WHEN INTRODUCED TO OOP

2005 - today

- Since 2005 we have been studying thoroughly students' programs developed at lab sessions, as home assignments, as well as the programs developed during the middle term and final exams. The result of this rigorous analysis of programs resulted in the formulation of **categories of students' difficulties and misconceptions**. This list of difficulties and misconceptions was used for **devising new examples, assignments, didactical interventions and even adjustments in the teaching approach**.

Xinogalos, S., Satratzemi, M. & Dagdilelis, V. (2006), Studying Students' Difficulties in an OOP Course Based on BlueJ, *9th IASTED International Conference on Computers and Advanced Technology in Education (CATE 2006)*, 4-6 October 2006, Lima, Peru, 82-87.

Xinogalos, S., Satratzemi, M. & Dagdilelis, V. (2008), An analysis of students' difficulties with ArrayList object collections and proposals for supporting the learning process, *Proceedings of the 8th IEEE International Conference on Advanced Learning Technologies (IEEE ICALT 2008),* 18-20 July 2007, Niigata, Japan, 180-182.

Xinogalos, S. (2010), DIFFICULTIES WITH COLLECTION CLASSES IN JAVA – The Case of the ArrayList Collection, *Proceedings of the 2nd International Conference on Computer Supported Education (CSEDU)*, 7-10 April, Valencia, Spain, 120-125.

OBJECT-ORIENTED PROGRAMMING COURSE DESIGN

2005 - today

- Since 2005 a **long-term evaluation and reformation of the OOP course** takes place based on the results from studying students' difficulties and evaluating various programming environments.

Xinogalos, S. & Satratzemi, M. (2009), A Long-Term Evaluation and Reformation of an Object Oriented Design and Programming Course, *Proceedings of the 9th IEEE International Conference on Advanced Learning Technologies (IEEE ICALT)*, July 2009, Riga, Latvia, IEEE Computer Society Press (best short paper award), 64-66.

Xinogalos, S. (2009), Guidelines for Designing and Teaching an Effective Object-Oriented Design and Programming Course, In *Advanced Learning*, Raquel Hijón-Neira (ed.), INTECH, 397-422.

Xinogalos, S., Satratzemi, M. & Dagdilelis, V. (2007), Teaching Java with BlueJ: a Two-Year Experience, *ACM SIGCSE Bulletin, Vol. 39, Issue 3,* 345.

Xinogalos, S., Satratzemi, M. & Dagdilelis, V. (2007), Re-designing an OOP course based on BlueJ, *Proceedings of the 7th IEEE International Conference on Advanced Learning Technologies (IEEE ICALT 2007),* 18-20 July 2007, Niigata, Japan, 660-664.

Xinogalos, S., Satratzemi, M., Dagdilelis, V. & Evangelidis, G. (2006), "Teaching OOP with BlueJ: a Case Study", *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies (IEEE ICALT 2006),* Kerkrade, 5-7 July 2006, 944-946.

TEACHING ACTIVITIES

1998- 6/2010:	Teacher of Informatics in Secondary Education
2004-2010:	adjunct Lecturer at the Departments of “Applied Informatics”, “Educational and Social Policy”, “Technology Management” at the University of Macedonia (teaching programming courses
since 6/2010:	Lecturer, Department of Technology Management

Current Courses

Department of Technology Management, University of Macedonia
- Computer Programming (2nd semester)
- Object-Oriented Design and Programming (3rd semester)
- Network-centric Software (8th semester, new elective course)

Postgraduate Programme in Information Systems, University of Macedonia
- Software Design and Development (1st semester)

Administration of the Courses

Teaching and learning is supported by an **asynchronous e-learning platform** called CoMPUs (Course Management Platform for Universities: http://compus.uom.gr).

The following **features** of CoMPUs are utilized **for supporting students in learning**:

Calendar: the calendar is kept updated with the lessons carried out (in corresponding dates), their content, and the associated educational material available in CoMPUs.

Documents: this tool presents students with a structure of folders corresponding to the lectures, enabled when a lecture is carried out and containing all the necessary educational material (presentation, sample programs, and all kinds of supporting material).

Students' assignments: this tool is used for accessing through a distinct link for each lab the weekly homework assignment. For each assignment the following information is provided: which projects from the downloadable lab leaflet (containing several projects) must be submitted; instructions for the assignments or/and supporting material (optionally); the valid names for the submitted files; the submission deadline. Students submit their assignments through this tool.

Administration of the Courses

Discussion forum: the tool is organized in sections that correspond to lectures, as well as a section for general topics. Students can post their question/comment and their colleagues and/or the teacher responds. Students are strongly encouraged to use this tool for communication and cooperation during studying and problem solving, but experience has shown that it is poorly utilized.

Administration of the Courses

The features of CoMPUs utilized mostly by the teacher for course management are:

Description of the course: this area contains a description of the didactical aims and the content of the course, the available educational material, the textbooks, the software needed, students' obligations and grading policy. This information is available from the beginning of the course and constitutes a clear **didactical contract** between the students and the teacher.

Assignments management: this tool is used for assigning homework to students, collecting their solutions and downloading them organized in a separate folder per student. An xls file with the assignments submitted by each student can be downloaded, as well as the final grade of the assignments if they are graded online.

Announcements: this tool is used for making announcements regarding the course and sending them by email to all students.

Learning design in both courses

- The course consists of 2 week hours of lectures and 2 week hours of labs (13 weeks)
- Students submit weekly assignments through CoMPUs
- Students write middle-term exams on paper (implementation of programs)
- Students write final exams on paper (implementation of programs)
- Grading policy:
    - 20% by weekly assignments
    - 20% by middle-term exams
    - 60% by final exams

Computer Programming Course

- Based on the imperative programming paradigm
- Uses C as the programming language.

  Although C might not be considered as the best choice for first programming language there are several factors that favour its usage - one of the main factors is that the curriculum gives emphasis on the areas of telecommunications and digital systems, and C is considered a useful tool in these areas.
- In order to support students in their introduction to programming with C and help them cope with some of the recorded difficulties, the libraries developed by Eric Roberts at Stanford and the accompanying textbook are used.

  These libraries tackle with some of the difficulties posed when using C for introducing novices to programming. For example: simplified functions (GetType(), i.e GetInteger()) are used for user input, instead of the scanf() function and the underlying & operator; classical data types, such as Boolean and string, that are not present in C are implemented in the libraries.

Object-Oriented Design and Programming Course

- Students are introduced to the main concepts of OOP using objectKarel (first two lessons)
- Next, the BlueJ IDE and Java are used.
  The teaching approach used is an "objects-first", iterative (important concepts are taught first and often), project-driven approach. Students begin with a predefined set of classes, create objects and invoke the available methods, in order to study the objects' behavior. Next, students extend existing classes by implementing or adding their own methods, define new classes in the context of existing projects, and finally create their own projects.
- In the middle of the course a professional IDE, usually JCreator Pro, is presented to students and they are left free to decide on their own which environment fits better to their needs.

What is missing?

- A system for automated checking of programs, assisting the teacher in grading assignments and providing immediate feedback to students.

- Supplemental material for stand-alone studying:

under development (graduation theses):

  - a web site for supporting the learning of imperative programming with the use of multiple representations of programs (pseudocode, structured flowcharts, programs in C) and the presentation of problem solving methodology

  - a course for OOP based on Java for the Adaptive Educational Hypermedia (SCORM compliant) LMS *Proper* (http://proper.uom.gr)