universität wien

Diplomarbeit

Titel der Diplomarbeit:

Musical Algorithms and Data Structures in Programming Instruction

Verfasser:

Rainer Dangl

angestrebter akademischer Titel:

Magister der Naturwissenschaften (Mag. rer. nat.)

Wien, im April 2010

Studienkennzahl lt. Studienblatt:	A 190 344 884
Studienrichtung lt. Studienblatt:	Lehramtsstudium UF English, UF Informatik und Informatikmanagement
Betreuer:	Univ.-Prof. Dr. Erich Neuwirth

Acknowledgements

First and foremost I want to thank my family for their great support throughout my studies. I am truly grateful that they always believed in my skills end expertise - without them I would have never made it that far. Thank you very much!

I also want to thank two of my colleagues, Christoph Jarosch and Christoph Grapa. We worked together for almost the entire course of our studies, and we share many ups and downs and humorous experiences. Through our cooperation and help, our sometimes very busy schedules became manageable and we benefitted greatly from each other. Thank you!

Over the course of writing this thesis, after investigating the principles and ideas for this project more and more, I was quite surprised and fascinated in which way two seemingly unrelated fields like music and programming can be linked in order to achieve a learning outcome. I am greatly thankful to Univ.-Prof. Dr. Erich Neuwirth for offering such a highly interesting topic and for his great support throughout the process of writing this thesis.

Contents

1 Introduction **11**

2 Learning theories **13**

2.1 Behaviorism 13

2.2 Cognitivism 16

2.3 Piaget's *Theory of Cognitive Development* 22

2.4 Constructivism/Constructionism 24

2.5 Summary 27

3 Theoretical foundations for CS teaching **29**

3.1 A definition of didactics for computer science 29

3.2 Principles of teaching methodology 31

3.2.1 Forms of teaching 31

3.2.2 Problem based learning 33

3.2.3 Active learning 35

3.2.4 Project based learning 36

3.3 Summary 39

4 Teaching programming **41**

4.1 General ideas 41

4.2 General approaches to teaching programming 42

4.2.1 Semiotic ladder 42

4.2.2 Cognitive objectives taxonomy 43

4.3 Music and programming - a contradiction? 44

4.3.1 An argument in favor of music 45

4.3.2 Data structures in music 46

4.3.3 Further possibilities of music in CS teaching 54
4.4 Summary 62

5 Teaching project 64
5.1 Prerequisites 64
5.2 Target group 64
5.3 Learning targets 65
5.4 Timeline 66
5.4.1 At school 66
5.4.2 At university 66
5.5 General expectations 66
5.6 Course structure 67
5.7 Lesson plans (for school) 67
5.7.1 First week 67
5.7.2 Second week 68
5.7.3 Third week 69
5.7.4 Fourth week 70
5.7.5 Fifth and final week 71

6 Project evaluation 72
6.1 Sample group 72
6.2 Teaching sequence 73
6.3 Selected tasks 74
6.4 Significant results 75
6.4.1 Naming conventions 75
6.4.2 Task 1 and 2 75
6.4.3 Tasks 3 and 4 78
6.4.4 Task 5 78
6.4.5 Task 6 80
6.4.6 Tasks 7 and 8 81
6.4.7 Task 9 86
6.4.8 Task 10 86
6.4.9 Further macro codes (task 11 on the diagram) 86

6.5 Reflection and Summary 88

7 Conclusion **90**

8 Bibliography **94**

9 Appendix **98**

9.1 Abstract - English 98
9.2 Abstract - German 99
9.3 Workbook 100
9.4 Presentation for the evaluation 122
9.5 Task sheet for the evaluation 131
9.6 Curriculum Vitae of Rainer Dangl 134

List of Figures

2.1 A finite relational graph 18
2.2 A propositional network 18
2.3 Example network for animals 20
2.4 Example network for software 21

3.1 Embedding of didactics of computer science 30
3.2 Problem solving process 33
3.3 Explorative learning 35
3.4 Categorization of project based teaching 37

4.1 Semiotic ladder 42
4.2 Cognitive objectives taxonomy 44
4.3 A simple tune 47
4.4 Objects in music 47
4.5 Phrase 1, tune #1 48
4.6 Phrase 2, tune #1 49
4.7 Scale in MidiCSD 54
4.8 UML diagram for music 59
4.9 A context-free grammar for music 60

6.1 Gender ratio 72
6.2 Study areas of sample group members 73
6.3 Choice of tasks 74
6.4 Distribution of tasks 88

List of Tables

2.1 Amplification of Pavlovian reactions 14

5.1 Lesson plan week 1 68
5.2 Lesson plan week 2 69
5.3 Lesson plan week 3 69
5.4 Lesson plan week 4 70
5.5 Lesson plan week 5 71

6.1 MIDI table of 'Oh when the saints' (I-M) 76
6.2 MIDI table including chords (A-F 2) 77
6.3 Bass line for the original phrase (I-F) 82
6.4 Bass line for *saints1* (A-M) 83
6.5 Additional chords (A-M) 84
6.6 Additional clarinet voice (A-M) 85
6.7 *init* phrase containing instrument assignments (A-M) 85
6.8 *init* phrase containing sound effects (Mixed 2) 88

1 Introduction

This diploma thesis explores the application of music in a relatively unusual environment - the teaching of computer science, specifically in programming instruction. This may seem to be an unconventional approach, but this thesis intends to prove that music can significantly facilitate learning and increase student motivation and interest in the subject.

It can be said that computer science is in many ways a very special subject at school or university in terms of setting and didactical requirements. As the first chapters of the thesis show, learning theories and methodological approaches apply in a unique way - for instance, few other subjects require a similar amount of active learning by students and emphasize the constructionist approach similarly, because learners should not only learn *about* computers, but also how to work *with* computers. For many students, this might be difficult to grasp, especially when it comes to more advanced topics of computer science, in the case of this thesis it is programming.

It seems justified to assume that the usage (i. e. the syntax) of a particular programming language does not so much pose a significant difficulty for learners. Languages like Logo are syntactically easy compared to languages like C/C++ or Java in order to be easily accessible to beginners or programming. Yet in order to develop well coded programs and to understand the fundamental concepts of programming, a certain way of algorithmic thinking needs to be developed and abstract concepts such as data structures, especially object-oriented structures are essential. A knowledge of these concepts is vital and functions as the basis on top of which one can use a programming language to develop well designed applications. The question now arises how to teach these fundamentals, which brings up the hypothesis of this thesis: the teaching of abstract concepts needed for introductory programming can be significantly improved when familiar concepts are utilized. For the purpose of the argument in this thesis, music shall serve as such familiar concept. It is assumed that the development of algorithms and data structures can be derived from arranging music and working with musical structures. Chapters 4 and 5 focus on this discussion and how to implement it in a teaching environment. The validity of the hypothesis is tested in chapter 6, where in a teaching sequence some aspects of the discussion are put into practice.

For teaching purposes a workbook is created that shall serve as an aid in teaching musical programming with MidiCSD and explains in a learner-friendly way features and possibilities of the development environment and can either be used in teaching but also for self-study.

2 Learning theories

Every theory of didactics and pedagogy obviously has its roots in the major learning theories. Thus, this chapter focuses on the three main concepts that are relevant for the development of the hypothesis later on. These concepts are behaviorism, cognitivism and constructivism (or constructionism). Each theory shall be analyzed with regard to how it could contribute to a didactical framework in the CS classroom.

2.1 Behaviorism

Behaviorism is a theory that essentially says that all psychological findings have to be verifiable by experiments.[1] Furthermore, the human mind is regarded as a black box, only observable behavior is regarded being important. Therefore, behaviorism does not attempt to look at how learning actually takes place, it rather tries to predict a certain outcome, behavior or reaction to external stimuli.[1] Clearly, one may say that behaviorism is an outdated theory, as a crucial part of the learning process - the brain - is completely ignored. Yet still, behaviorism yields a few interesting aspects for the discussion here, therefore it shall be explored briefly in more detail.

Behaviorism was first explored intensively by Ivan Pavlov.[1] He conducted experiments in order to investigate the connection between *neutral stimuli* and *uncon-*

ditional stimuli which, when applied simultaneously, create a certain *conditional reaction*.[1] Some of his experiments became very well known, for example the famous dog experiment. Pavlov discovered that the ring of a bell (a neutral stimulus) can trigger increased production of a dog's saliva (conditional reaction) when combined with a second (unconditional) stimulus, meat. After some time, the dog reacts solely when hearing the bell.[2] This form of experiment and the field of research in which Pavlov was working in is called *classical conditioning*.[1]

How are Pavlov's experiments, which were initially aimed at predicting animal behavior, relevant for didactical concepts? John Watson was the scientist who first thought that Pavlov's findings could be applied in learning psychology.[1] He tried to define human behavior in Pavlov's terminology and claimed that research into learning psychology should limit itself to observable output. He classified emotional behavior as a subcategory of classical conditioning.[1] Watson therefore laid the foundations for Thorndike and Skinner, who developed Watson's theories further.

Edward Thorndike introduced the concept of *Amplification of Pavlovian Couplings* that was developed further by Burrhus F. Skinner to his *Theory of Operant Conditioning*.[3] Thorndike and Skinner noted that Pavlovian reactions can be amplified as shown in table 2.1.[3]

	comfortable stimulus	uncomfortable stimulus
applied	positive amplification	punishment
removed	punishment	negative amplification

Table 2.1: Amplification of Pavlovian reactions

Empirical findings show that punishment is much less effective than amplification - it usually leads to suppression of behavior when the punisher is present.[3] Obviously, behaviorism is only to a small extent relevant in modern didactical concepts, yet as Hubwieser points out, one certainly can extract useful suggestions in order to improve learning:[3]

- create a comfortable learning environment with a relaxed, attention-enhancing atmosphere
- utilize continuous, but differentiated positive amplification (praise)
- avoid punishments
- avoid defense reactions and generation of fear

Although one might claim that behaviorism is an outdated theory, there are a few points that still are quite relevant even in modern didactics. While it is generally agreed that punishment is not a part of pedagogic work today, one clearly can see that positive reinforcement leads to positive achievements in terms of learning outcome. As Hubwieser stresses in the points mentioned above, a positive learning environment and praise can be significantly more effective than punishment. It is thus noted that a comfortable learning environment benefits motivation and learning outcome and shall clearly be preferred over an intimidating and overly strict teaching style.

Furthermore, it should be noted that behaviorism is closely connected to another major problem in teaching - assessment. It is a great challenge for a teacher to objectively and uniformly assess students. As the teacher can only put marks on observable output, a behaviorist concept is at work here as well. Yet still, the challenge for teachers is to find out whether students learn by heart of if they actually understand the topics discussed in class. This differentiation cannot be made on a behaviorist basis. It is therefore necessary to expand the definition of learning beyond behaviorism. It is obviously not enough to just look at the output as such; in order to really understand learning, one needs to include the brain and how it processes information. This helps to understand how learning takes place and whether the output is the result of an actual learning process. Hence the next section focuses on cognitivism.

2.2 Cognitivism

Cognitivism is the result of a counter-movement to behaviorism that started simultaneously in the USA and Europe.[4] It basically claims that learning, a complex process, cannot be sufficiently described by behaviorist theories and that the (inner) changes of brain networks and structures (caused by learning) are significantly more important when one tries to explain how learning actually takes place.[4] Cognitivism particularly focuses on the description and modeling of so called *higher mental processes* - it does not try to predict a certain outcome, as behaviorism does, it rather tries to explain the outcome by modeling the mental processes at work in the brain, which in turn gives scientists clues on how to present the input.[4] It is therefore crucial to understand how information is stored, processed and, which is particularly important, how pieces of information are connected.

Donald Hebb was one of the most important psychologists in the field of cognitivism.[5] He tried to explain learning by a model of the electro-chemical processes in the brain.[6] The most important elements of his model are *neurons* (approx. 1.5 billion nerve cells in the brain and spine) which connect *receptors* (sensory organs) and *effectors* (e. g. muscle cells).[6] The neurons transmit electro-chemical impulses that need a certain recovery time between two impulses. Therefore, in order to store impulses, *circuits* are necessary. These are defined as *neural circuits* and represent basic results of learning in the brain.[6] An example of this principle can be learning of how to write. The information stored in the brain is transmitted by neurons which deliver the information from the sensory organs (in this case the eyes) and cause the brain to process the next letter and send the information to the effectors (here the muscles of the arms and hands). The activation of the neurons is triggered by an electro-chemical process and can be altered by certain diseases (e. g. Parkinson's disease).[7]

As it has been clarified how information is stored and transmitted in the brain, the much more interesting and relevant question in terms of didactics is how information is connected. In order to explain target-based behavior in animals, psychologists introduced the cognitive inter-processes of *anticipation* and *understanding*, a concept they called *cognitive map*.[8] Jerome Bruner included these concepts and expanded them to create his theory of acquisition of concepts.[9] Possibly the best way to describe how information is linked in the brain was introduced by John Anderson, who developed the idea of *propositional networks*.[3] Propositional networks are a part of Anderson's theory of *ACT - adaptive control of thought*, a theory where Anderson attempts to simulate and understand human cognition.[10] Anderson defines a propositional network as a special case of a *finite labeled graph*, which is defined as:[11]

$$G = \langle R,\ N,\ A \rangle$$

R is a finite set of relations, N is a finite set of nodes and A is a finite set of links in the graph.[11] The members of A are represented by triples of elements denoted $\langle a\ X\ b \rangle$. Thus, in the graph the the relation X connects nodes a and b.[11] A good example of a finite relational graph is given in figure 2.1.[12]

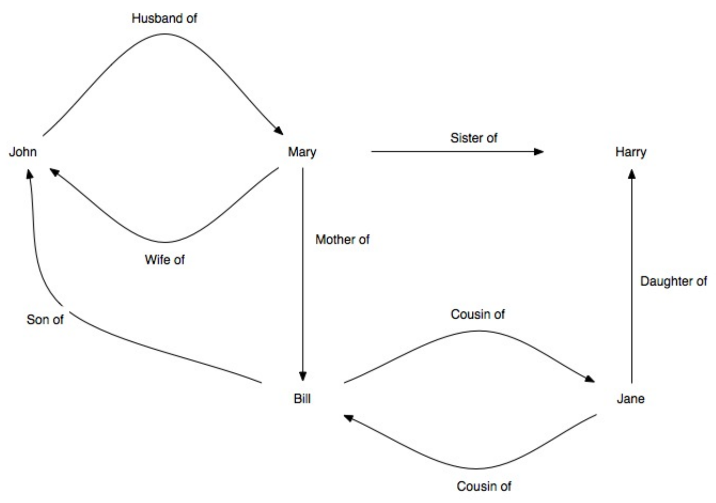


Figure 2.1: A finite relational graph

This model nicely shows that information can be presented precisely and without excess meaning.[12] The direction of the arrows is important as arrows in opposite directions do not necessarily have the same labels. Anderson considers this graph to represent ideas which are linked to each other.[12] This should, when starting out at one node elicit the next idea (node) via a link (arrow). Anderson developed this idea further and created propositional networks as a special form of the finite labeled graph. Anderson gives the following simple example in figure 2.2.[13]

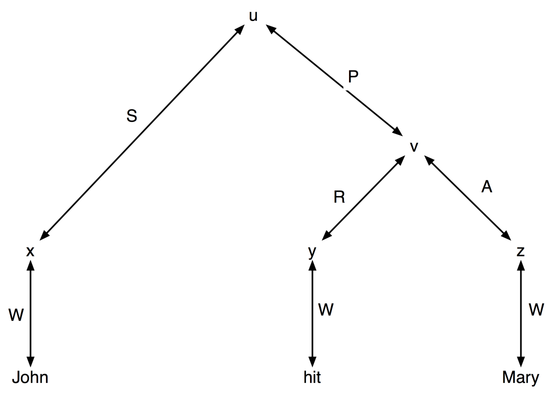


Figure 2.2: A propositional network

Obviously, one can observe a quite important difference from the finite labeled graph: there is a hierarchical structure. In a propositional network, each of the nodes denotes a proposition.[14] In this case there is a set of nodes {*u, v, x, y, z*}. These nodes have truth values, i. e. propositions are the smallest unit of knowledge that can be determined to be either true or false.[15] Furthermore, there is a set of functional variables {*S, P, R, A, W*}. They are defined as *S = subject, P = object, R = relation, A = agent* and the lowest level of syntactic function *W = item.* When looking at the network, it is now possible to identify the word *John* as the subject of a certain action. The subject node *x* containing *John* is then tied via the root node *u* to a certain object node *v.* The object of John's action is a an agent, in this case *Mary*, which is represented by the node *z.* The relation between the subject John and the object agent Mary is the given by node *y*, which contains the item *hit.* It therefore becomes clear that apparently *John hit Mary.* Anderson claims that propositional networks attempt to "set forth a a language of the mind, a 'mentalese' in which all knowledge is to be represented."[16] A useful didactic context for propositional networks is illustrated by Friedenberg and Silverman, who define propositional networks as representations of simple factual properties of certain objects in the world (as illustrated in image 2).[14] Yet Furthermore, they can also represent a category relationship, resulting in so called 'is-a' link or a property type relationship with a 'has-a' link.[14] Hence, in a very simple network with the two nodes (*car, jeep*), one can express the relationship between the two nodes by saying "a jeep *is a* car". Hubwieser gives the following example in figure 2.3.[17]

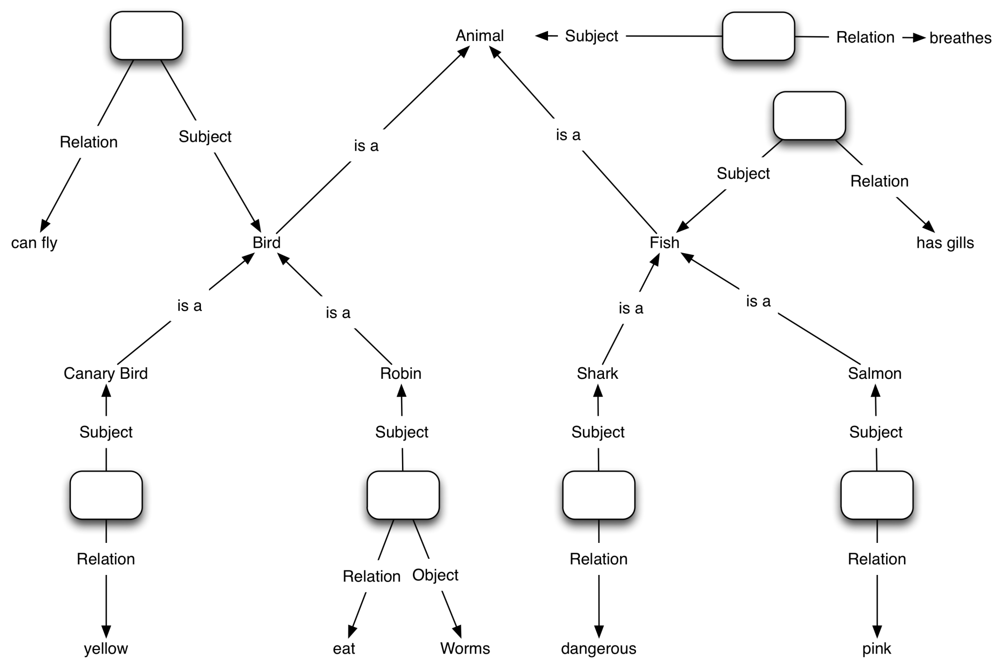


Figure 2.3: Example network for animals

This propositional network shows several points that are relevant for didactics and that are crucial in order to understand how to present input in class. Hubwieser claims that the following statements are important from a cognitivist point of view:[17]

- before starting the teaching sequence, learners should be aware of the target and the significance of the learning process
- the subject matter should be embedded in superior relationships
- the subject matter should be presented and structured in a way that facilitates the creation and acquisition of categories
- there should be as much links to already existing knowledge as possible

Finally, to give an example from computer science, figure 2.4 shows a propositional network about software that nicely illustrates how teaching sequences should be structured and presented to allow a fast connection of knowledge and efficient learning.

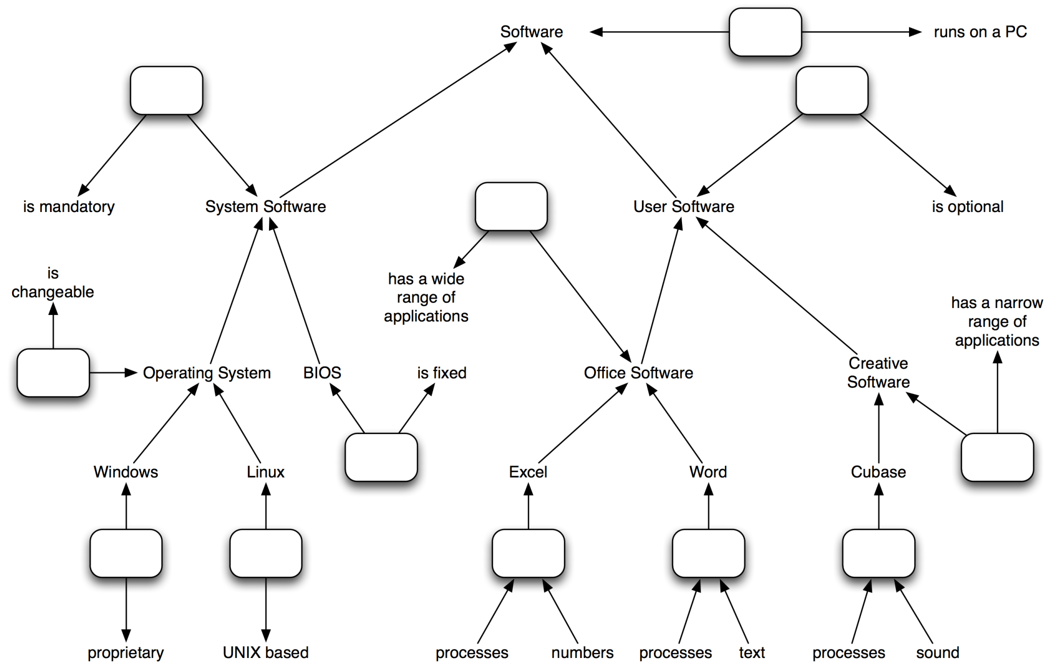


Figure 2.4: Example network for software

As can be seen in the graph above, learning content can be deconstructed into a propositional network and the subtopics can be put in a relationship to each other. This is a particularly important principle of didactics, which has its application in everyday teaching practice. Students are by far more comfortable when they are able to put new subject matter in context and relate it to already existing knowledge. It is therefore crucial to understand how the mind maps the information stored in the brain in order to be able to present new subject matter best to students. Furthermore, it is certainly significant which stage of cognitive development one can expect and to what extent the mind has developed in learners. This leads to Jean Piaget's *Theory of Cognitive Development* which not only includes aspects from cognitivism, but also relates to the third major theory discussed in this thesis, constructivism.

2.3 Piaget's Theory of Cognitive Development

Jean Piaget introduced several highly significant terms to cognitivism that greatly influence the way one can describe information mapping in the mind. First, Piaget defines the term *schema* and its plural *schemata*.[18] As Piaget used to work as a biologist in his early years, he believed that the structure of mind is similar to the structure of the body.[18] Wadsworth claims schemata to be "cognitive or mental structures by which individuals intellectually adapt to and organize the environment."[19] One can describe schemata as concepts or categories that exist in the mind and which are used to identify and process incoming stimuli. Wadsworth also describes schemata like an index file in which every index card represents a schema. Therefore, adults have many schemata simply based on their experience, whereas children, especially infants, have very few schemata.[19]

As an example, it is assumed that a child knows what a dog is, i. e there is a schema in the mind that describes the typical features of a dog. When this child sees an animal which it has never seen before, for example a wolf, it might mistake the wolf for a dog, as quite many characteristics of the two animals overlap (similar stature, fur, howling, ...). It is only when somebody explains the difference, the child creates a new schema for the wolf with the unique features that characterize it. This example illustrates another important term closely connected with schema: *assimilation*.[20] The dog-wolf example explains what happens when due to a missing schema a new object is assimilated to another category. The child does not recognize the wolf and simply classifies it as a dog. Yet another process that could happen is *accommodation* - not the object is classified in the wrong category, but the category (or schema) itself becomes modified.[20] In this case this would mean that the child alters the dog-schema to fit the characteristics of the wolf. Accommodation and assimilation are processes that correct each other and maintain a certain balance, which Piaget defines as *equilibrium*.[20] For example, it would be undesirable when a person always assimilates new stimuli. This person would over time generate very large schemata and would not be able to differentiate.[20]

It is now established how Piaget describes the structure of information in the mind. He classified the ways information can be handled by the stages of cognitive development:[21]

1. **The stage of sensorimotor intelligence (0-2 years)**: during this stage, behavior is primary sensory and motor. Very few schemata exist and the child cannot represent objects and events conceptually.
2. **The stage of pre-operational thought (2-7 years)**: in this stage language is developed and concepts develop rapidly. Reasoning during this stage is pre-logical (semi-logical), as it is dominated by perception.
3. **The stage of concrete operations (7-11 years)**: the child develops the ability to apply logical thought to concrete problems.
4. **The stage of formal operations (11-15 years and older)**: the cognitive structures reach their greatest level of development and therefore the child is able to apply logical reasoning to all classes of problems.

Cognitive development always follows the order of the stage above, yet it is possible that some children enter certain stages sooner or later as the age limits given here are based on the average development of a child. Moreover, it might be possible that some children do not enter the stage of concrete of formal operations at all.[22]

Summing up the conclusions that arise with regard to teaching, Hubwieser states the following:[23]

- in primary school, teaching should always utilize concrete objects, which should as far as possible be actually presented in class
- in secondary school, abstract theoretical concepts are not helpful
- 7th grade is the earliest possibility to teach formal operations
- it is questionable to divide children into several school types before they reach the last stage as later development can not take form at all or may be amplified reversely

This has some consequences for teaching computer science: some issues might be too complex for learners to understand. It is then the responsibility of the teacher to try presenting topics that require formal operations in another form that might be better suited to the learners' capabilities.

2.4 Constructivism/Constructionism

The behaviorist and cognitivist concepts that are relevant for teaching have been established. The third major learning theory is constructivism (or constructionism), which seems to dominate the view of learning in current educational research.[24] According to Matthews, constructivism consists of two core propositions:[25]

- knowledge is constructed actively by the cognising subject - it can therefore not be passively acquired from the environment
- the process of acquiring knowledge is an adaptive process that organizes the learner's experiential world - it does not discover an independent pre-existing world which is outside the mind of the knower

Olssen mentions a number of propositions that are to a certain extent contained within the core definitions above: Miller and Driver claim that "knowledge is personally and socially constructed";[26] "knowledge is rather 'made' than 'discovered' and that interpretative categories are prior to facts",[27] "truth is 'provisional' rather than 'certain', and 'limited' rather than 'foolproof'",[28] and that "rather than revealing an objective, independent world, knowledge gives us 'constructs' or 'frameworks' by which we make sense of experience".[29] Similarly, Fox defines constructionist claims as follows:[30]

- learning is an active process
- knowledge is constructed, rather than innate, or passively absorbed
- knowledge is invented not discovered
- all knowledge is personal and idiosyncratic
- all knowledge is socially constructed
- learning is essentially a process of making sense of the world
- effective learning requires meaningful, open-ended, challenging problems for the learner to solve

One might think that these propositions result in a unified definiton of constructivism. This is not the case, as Fox notes the following sub-categories: Piagetian constructivism,[31] Neo-Vygotskian constructivism,[32] Feuerstein's mediated learning,[33] radical constructivism[34] and social constructivism of various forms.[35]

All these constructionist movements may have their application in a certain field, yet Hubwieser notes that recently a more moderate constructivist approach is being used in contrast to radical constructivism.[36] Radical constructivism claims that everything one perceives and knows is a creation of the observer.[37] The reality as such cannot be accessed, which is not to say that it does not exist. Radical constructivism merely says that it is impossible for the individual to see reality, because everybody creates his individual interpretation of reality which might not be the same as someone else's interpretation.[38] Hubwieser mentions the following constructivist movements that are particularly relevant in CS didactics:[39]

Situated Cognition: In this branch of constructivism, in order to construct knowledge, the social setting and the topical context are particularly important.

Anchored Instruction: The learning content is tied to a so called *narrative anchor* in this approach, which means that one uses stories in which authentic and interesting problems are embedded.

Cognitive Flexibility: When discussing rather complex and relatively unstructured learning content, one should avoid inappropriate simplifications. Instead, it is better to show learners the actual complexity of the problem by offering different approaches at different times in changing contexts with different targets in various perspectives. This creates several independent approaches to the topic which in turn facilitates memorization and application.

Cognitive Apprenticeship: Along the lines of traditional apprenticeship, one can present the approaches and problem-solving methods of authentic examples to illustrate the work of experts to learners. The setting in which the learning process takes place should be as close to a real working environment as possible.

It is certainly clear that for some topics and subjects, some of these approaches may be not as good as others, yet when thinking of computer science and particularly programming, one immediately can say that cognitive apprenticeship is a highly useful method - for example, a certain job experience is a teaching requirement in technical schools in Austria.[40] That of course is an attempt to transfer as much knowledge from the experience of professionals to the school.

Finally, Hubwieser notes several points that can be derived from constructivism and applied in didactics:[41]

- active involvement with the learning content is absolutely necessary (as far as this is possible)
- learners should explore problem-solving methods independently
- the teacher should act as tutor and not as presenter
- during instruction, the teacher should give the learners enough time to properly go through the knowledge construction processes
- learning settings should be as close to reality as possible
- the same learning content should be explored from different angles

All these points illustrate the exceptionally good suitability of constructivism for a didactical framework for computer science. Constructivist approaches are also strongly reflected in concepts discussed in chapter 3 (especially *active learning*).

2.5 Summary

In order to briefly summarize this chapter, one can conclude that the three most important learning theories contribute certain important features to didactics of computer science. Hubwieser claims that the following points should be a basis for the teaching methodology discussed in the next chapter:[42]

- a comfortable learning environment is necessary, in which the learners have time to satisfy special interests and needs without pressure, thus ensuring that motivation and attention are encouraged and maintained.
- classification of learning content in larger contexts and clear structuring of the topics allows learners to develop propositional networks.
- an active examination with the topics is desirable, where the teacher should create sufficient problem awareness before the presentation of solutions. These claims stem from the common notion of all constructivist approaches, which say that knowledge is actively constructed by the learner.

- various approaches and perspectives to the same topic should be offered, as stated in *Cognitive Flexibility*.
- one should aim at creating problem situations that are as authentic as possible, as this enables learners to solve problems in a setting where these skills are actually needed.
- learning content should be age-appropriate, as certain problem-solving skills develop at a specific age.

It is now established that no learning theory can stand on its own. Quite contrary, only by applying parts of all theories one can arrive at a set of rules that facilitate teaching and benefit the learning of computer science. These rules are the basis for the approaches to teaching CS in the next chapters.

3 Theoretical foundations for CS teaching

This chapter focuses on the theoretical basis for the interdisciplinary teaching project discussed later on. For a thorough understanding of how teaching should take place, a few concepts and didactical tools shall be illustrated. Firstly, a clear definition of what the term *didactics* in computer science encompasses is given. Secondly, a few approaches to teaching computer science are presented which are utilized in the teaching project.

3.1 A definition of didactics for computer science

At the beginning, a definition of the term *didactics* is necessary. Therefore the question poses itself: *What is a didactic model or concept?* The following quote illustrates these terms nicely:[43]

> A didactic model is a theoretical construct that is as complete and universally valid as possible and that is used for planning and analyzing instructional action for teaching and learning situations.
>
> If the strict conditions with regard to completeness and universal validity are not fulfilled, one defines this as a didactic concept.

This definition sufficiently encompasses the requirements for the teaching project that is the presented in this thesis. The model must be complete and universally valid to the highest possible extent. This is certainly a major problem, as teaching is a highly complex process and subjects differ greatly in terms of methods used in class and requirements of the curriculum.[44] This is what brings the discussion to subject didactics, or more accurately *teaching methodology of a particular subject*. One can certainly formulate principles of general didactics, but a broad discussion of this matter would go beyond the scope of this thesis. Yet a discussion of didactics of computer science, or teaching methodology of computer science, is a crucial point for a thorough understanding of the narrower field of CS didactics of programming.

The first important question is how all the interrelated fields that influence didactics in general (pedagogy and psychology) create the special field of CS didactics that enables the teaching of computer science in school. Schubert and Schwill propose an embedding of didactics in various fields as shown in figure 3.1.[45]

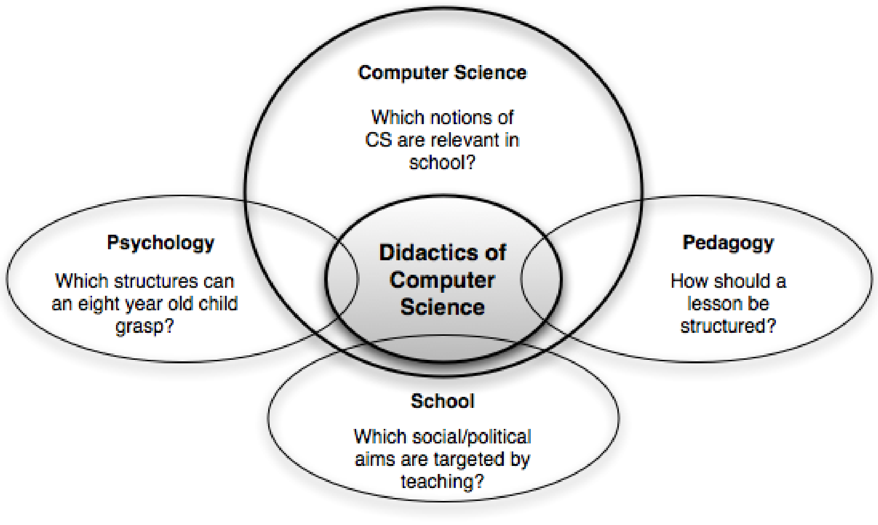

Computer Science
Which notions of CS are relevant in school?
Psychology
Which structures can an eight year old child grasp?
Didactics of Computer Science
Pedagogy
How should a lesson be structured?
School
Which social/political aims are targeted by teaching?

Figure 3.1: Embedding of didactics of computer science

It is quite clear from the figure that didactics of computer science is in principle a special discipline of computer science. Moreover, it is an interdisciplinary field that has to take into account various considerations in order to work: certainly there is a strong influence of pedagogy. It is of course fundamental for an accurate model of didactics that the teacher or instructor knows how to structure a lesson, but it is furthermore highly significant how to deliver the content to the learners. This is the field of psychology where the learning theories discussed in the previous chapter play an important role. Finally, also the school as an institution influences the teaching methodology.[45]

3.2 Principles of teaching methodology

As it has been clarified which areas influence didactics, the focus shall now be on didactics as such. When thinking about an approach to teaching computer science one has several options. As noted above, constructivism is the most powerful approach to use. Yet it still remains unclear which teaching concepts to use. Humbert notes that teaching methodology does not claim to be valid for every subject, it has to be applied and modified in order to satisfy the needs of the subject's didactic model (i. e. for a particular subject it is then universally valid and complete, as claimed above).[46] In the following sections, several definitions shall lay the basis for the construction of didactically ideal teaching.

3.2.1 Forms of teaching

Teaching obviously requires a certain form of interaction between teachers and students. Commonly, this interaction can be said to consist of two components: a *social form* and *activities* - social forms are classified as follows:[47]

S1: *teaching in front of the class:* here, all learners in class can participate equally, and can freely communicate with each other.

S2: *group teaching:* learners are divided into groups, communication only takes place within the group.

S3: *private teaching:* learners are isolated from each other. Therefore, communication is not possible or allowed. This form of teaching is commonly used in music teaching in specialized music schools (not high schools).

The second component, the activities, can also be divided into three groups:[48]

A1: *communicative form:* the learners concentrate on taking in the topics presented by the teacher. There is little or no room for the pupils to actively determine the course of instruction

A2: *guided exploration:* the learners actively co-determine the course of the teaching by their actions. The teacher encourages this form of participation explicitly.

A3: *free research:* the teacher merely gives incentives, the actions of the learners determine the course of the teaching.

These activities and social forms can be combined into nine forms of teaching. Several of these forms have explicit names. For example, S1 and A1 combined results in the *teacher-centered* form of teaching, which simply means that the teacher speaks and the learners listen and try to understand (i. e. a lecture).[48] This does not seem to be the best approach. Thus one can construct other forms that fit especially the didactic requirements of computer science: social forms S1 and S2 are to be favored, in particular group work is highly effective and widely used in teaching practice. From the activities, all three forms are acceptable - it strongly depends on the topics which of them is best.

3.2.2 Problem based learning

Humbert defines a problem as "a not routinely solvable task."[49] Hubwieser quotes Edelmann, who presents a slightly different description:[50]

> A problem is therefore identified by three characteristics:
> - undesired original state,
> - desired target state,
> - a barrier that obstructs the transition from the original to the target state at the moment.

Edelmann also gives a definition of the term *task*, which is that with a task "there are rules (knowledge, know-how) that help finding a solution."[51] Hubwieser claims that problem based teaching is one of the commonly acknowledged principles of *computer literacy*,[52] a term that refers to the use and understanding of computers as a basic skill like reading or writing.[53] Thus the process of problem solving seems to be a well suited means of teaching computer science.[54] A schematic diagram of said process can be described as shown in figure 3.2.[55]

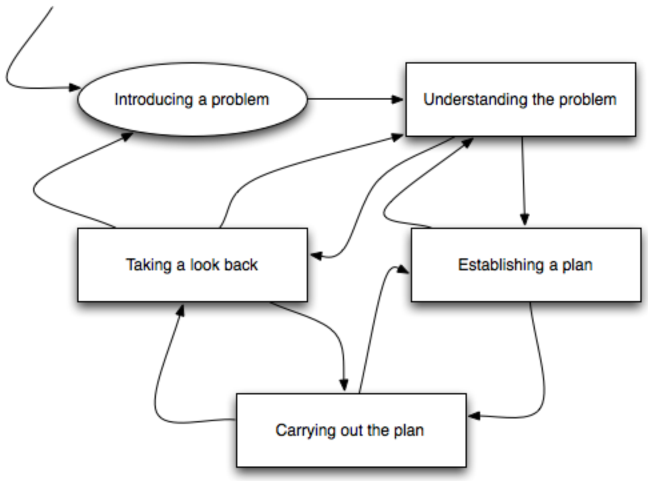


Figure 3.2: Problem solving process

It becomes clear why this method is a highly recommendable way for CS teaching: it allows a trial and error approach. Learners can at all times go back in the process and review the steps and, when necessary, modify their plan. Additionally, learners might understand the problem afterwards, when taking a look back, others might not need to do that. It is also clear that this method can handle different learning speeds as well. Furthermore, if a problem is solved, it can lead to a new problem that starts the whole process again which encourages self-responsible learning.

Humbert mentions three classes of problems. Firstly, problems that learners are confronted with in actual problem situations. The learner solves the problem without noticing that he has learned something at the same time. Secondly, problems that the learner tries to solve automatically and independently but with a deliberate learning intention. Thirdly, problems that the teacher poses to the learners for instructional purposes.[56] Problem classes 1 and 2 are a good entry point for teaching and due to the nature of problems - one can debate problems constructively - are a way of implementing the constructivist learning theory.[56]

The topic of problem classes and their application in teaching raises an important issue: the teacher is responsible for selecting suitable problems. This is a point where Hubwieser claims that the didactical abilities of the teacher are tested. The level of complexity should on the one hand be high enough that the learners cannot solve the problem without the concepts learned before (or if they can, only by putting in considerably more effort), on the other hand it should not be to hard either.[54] Problem based teaching avoids another undesired effect, which is the simple training of applications. A typical example is the teaching of programming in school, where learners are often unsatisfied with the outcome because they do not see the practical use of the concepts learned.[54]

3.2.3 Active learning

If one tries to define the term *active learning*, a simple interpretation seems obvious: learners should not be passive recipients of knowledge, but active constructionists (which draws a connection to the learning theory illustrated above). Humbert mentions that active learning is based upon Bruner's theory of *explorative learning*,[57] a process which is illustrated in figure 3.3.[58]

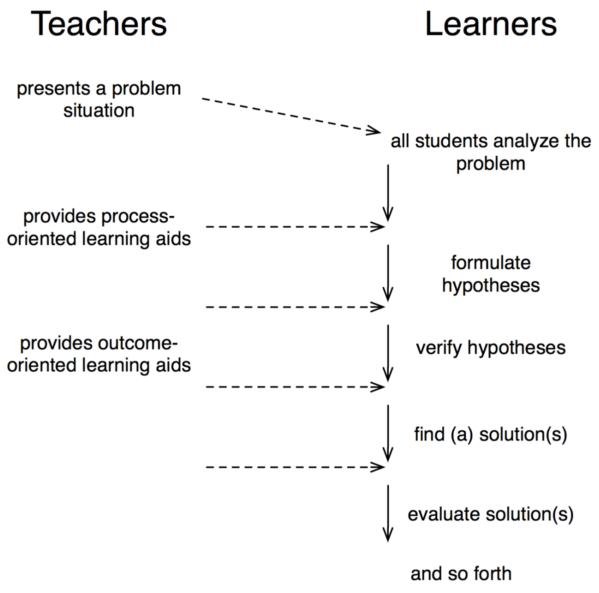

Teachers
Learners
presents a problem situation
all students analyze the problem
provides process-oriented learning aids
formulate hypotheses
provides outcome-oriented learning aids
verify hypotheses
find (a) solution(s)
evaluate solution(s)
and so forth

Figure 3.3: Explorative learning

It is quite clear that active learning is closely connected to problem based learning. The difference when applying active learning is that learners actively work on creating information and knowledge, there is less input necessary from the teacher. As can be seen from the graph, the teacher merely presents a problem and provides a certain amount of help during the process. The steps of creating and validating a hypothesis are also carried out by the learner, not only the part where the task is to find solutions and evaluating them. Particularly in computer science the implementation of active learning is a quite suitable possibility to give learners the chance of discovering concepts on their own. This point is very much reflected in the teaching project of this thesis; learners are supposed to do as much as possible on their own. Still, especially with beginners in CS is it obviously necessary to give a certain amount of assistance, but that can be reduced the more advanced the learners are.

What is the point in giving the learners the opportunity to actively do something instead of just lecturing them (which sometimes undoubtedly also has its justification)? For once, the most obvious reason (apart from avoiding boredom) is the significantly increased learning effect. It is commonly known that a very good way to learn something is by explaining it to somebody else, which means that an active involvement with the subject matter greatly helps to understand and actually learn it.

3.2.4 Project based learning

Project based learning can be seen an extension of problem based learning and active learning. It is of course possible to discuss and work on problems in class as illustrated in the previous section, yet it is preferable to put these problems into context. Such a context can be a project, taken from 'real life', which nicely points out the practical relevance of the subject matter to the learners. A project can be started for a wide range of problems: websites, programming applications, an art project, a music project and many more. Project based teaching is a quite important method in teaching programming, therefore the concept shall be discussed in some depth. When thinking about the placement of project based learning (or project based teaching, the terms are used synonymously here), Schubert and Schwill propose the schematic shown in figure 3.4 in accordance with the forms of teaching discussed above.[59]

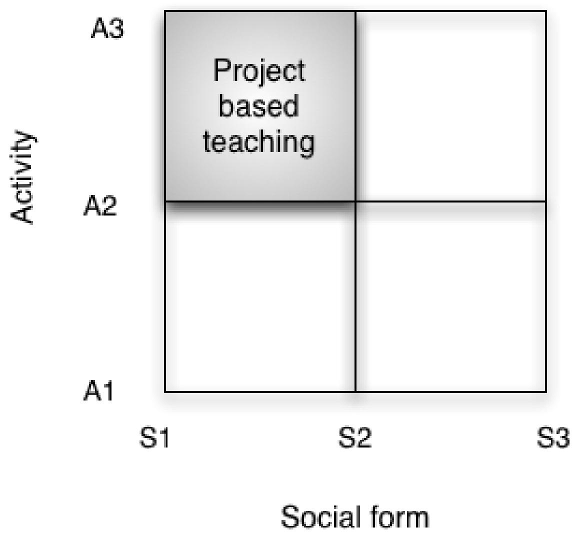


Figure 3.4: Categorization of project based teaching

Thus it becomes clear that a project allows a relative freedom of activity, i. e. the learners can (within the boundaries of a specified topic) freely decide which aspects of the project they should pursue first and how to do it. With regard to the social form it is obvious that a project cannot be an S3 form (private instruction). A project especially in computer science stresses cooperative learning and helps learners in developing an analytical approach to a problem.[60] When thinking about the pedagogical implications of projects, several points are highly significant.

Firstly, the project should be taken from the daily context with which learners are familiar. The project should also not be bound to a single scientific subject and thus not to a particular school subject on its own. This enables learners to think globally and outside the box of subject boundaries.[61]

Secondly, the project should be adjusted to fit the interests and wishes of all people involved (teachers and learners). It is particularly the teacher's responsibility to elicit the learner's interest in the topic in order to create a successful and motivated project team.[61]

Thirdly, one of the most important points that distinguishes project based teaching from regular teaching is the possibility for a certain amount of self-organized and self-responsible work.[61] At the beginning of each project teachers and learners should negotiate a way of how the work will be carried out. There have to be clear definitions as to how the project will be evaluated, what the target is, a timeline needs to be set (with strategies on how to manage delays) and so called *milestones* denote a stage the project should be in at a specific point in time.[61] Having clarified all these conditions, the project members can organize themselves and work independently to reach the project target (which should always exist - the planning of the project always should aim at a target that can be evaluated).

Fourthly, as already noted above, it is crucial for the project to have a certain amount of practical relevance.[61] Learners might find it interesting to work on a project, yet a purely artificial scenario is undesirable. Learners might be demotivated when they do not see which practical application the project actually has.

Fifthly, a project preferably should not only have an outcome in terms of learning but also some sort of final product (e. g. a film or a website) with proper documentation that can be presented to the public and subjected to criticism.[62] This also reinforces the previous argument, as learners finally see that their work resulted in an actual product.

Sixthly, it is recommendable to incorporate as many senses as possible in the process, which means that physical and technical abilities should be incorporated as well, as traditional instruction is largely focused on intellectual skills.[63]

Seventhly, a major aspect of project based learning is social learning.[63] Communication within the group of learners and also communication between learners and teachers on a peer level is quite important, as the learners have the feeling that the teacher does not necessarily know everything and that a transfer of knowledge can take place in both directions.[64]

Finally, interdisciplinary projects go beyond the scope of a subject and bring together several disciplines.[63] It is almost impossible to start a project in school without thinking about other subjects to collaborate with. This again shows the practical relevance to the learners and avoids that the project is carried out just for the sake of exercise. Bringing in other subjects very often also helps to understand certain applications of principles discussed in class. The teaching project of this thesis strongly reflects many of these principles, a discussion of which follows in chapter 5.

3.3 Summary

To summarize the approaches to teaching computer science, it can be said that clearly CS is a subject that obviously seems to fit perfectly in the concept of project based learning. In computer science, one always works on some kind of project, be it designing a website or programming an application. Moreover, very seldom does one work alone on a project, therefore the collaborative character of the subject is particularly important. When it comes to the problem of how the subject matter is to be delivered in class, the solution seems fairly straightforward; having clarified the forms of teaching and the activities with which they can be combined, the result shows that a simple teacher-led lecture or strongly teacher-guided lesson is not the best way to teach, especially when working with the computer. Problem based learning and in connection with it also active learning promise significantly better results, as learners are not only forced to actually do something, they need to develop the knowledge themselves and extract the important information out of hypotheses and theories they create when solving problems. This greatly reinforces the argument in favor of the constructionist approach in computer science, yet still the cognitivist aspects are implemented as well; teachers need to be aware of the learning curves, have to plan the lessons and the help they give to the learners accordingly. Furthermore, as pointed out above, the selection of problems and project topics is a major responsibility of the teacher and is a decisive factor whether the whole teaching concept succeeds (a paradigm where Piaget's theory needs to be kept in mind). On the whole, combining all the theories and approaches illustrated up until now it becomes clear what ideal teaching in computer science should look like. As a next step, this knowledge is applied to programming.

4 Teaching programming

4.1 General ideas

The topics discussed are now brought together in an attempt to create a teaching methodology for programming. The question at hand is: why does programming need a special didactical approach and what could this approach look like? Generally, programming is a discipline that requires a certain way of analytical thinking. Developing algorithms and a knowledge of data structures are preconditions in order to be able to program efficiently. Also, programming requires an extensive amount of exercise - it is impossible to learn programming by reading a book, it is necessary to actually program applications in order to acquire the knowledge.

In school or at university, the teacher has to solve several problems when designing his approach to teaching programming. First and foremost, a crucial point is to be able to to think algorithmically.[65] This might be the first barrier for many learners. The syntax of a particular programming language is not so much a problem, as there are many languages that are very learner-friendly (e. g. Logo or Scheme). The problematic aspect of teaching programming is how to teach learners to develop a solution for a given problem and then developing an algorithm that fits best in order to implement the solution in a specific programming language. Therefore one can say that the instruction should not so much (at least at first) focus on how the programming language works. New languages are constantly developed and existing languages keep changing or become outdated. Thus, an abstract knowledge of how to develop an efficient algorithm and how data is structured is far more useful, because if these concepts are clear it is by far easier to apply them in a programming language. The actual syntactical implementation is then not that much of a problem, as the rules of how to program in a particular language become quite clear if the fundamental knowledge that applies to all languages is solid.

4.2 General approaches to teaching programming

4.2.1 Semiotic ladder

A number of thoughts on the difficulties of teaching programming have been presented. As a next step, one may think about didactical approaches to teaching programming. Kaasbøll presents two important theories that are highly relevant for the teaching project in chapter 5. Firstly, Kaasbøll mentions the *semiotic ladder*, illustrated in figure 4.1.[66]

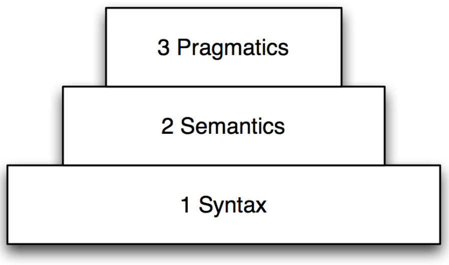


Figure 4.1: Semiotic ladder

This approach is characterized by the syntax of a programming language as the basis for teaching programming. If the learners know how the language works they can make their way to the semantics stage and learn language constructs. If the first two steps are fulfilled the learners can move on the use of the programming language for specific purposes, i. e. pragmatics.[66]

In principle, this approach seems perfectly fine, and indeed one can argue that for an independent use the semantics and at the very beginning the syntax of a programming language has to be learned. Still, this approach focuses solely on the learning process of a particular programming language and does not include the issues some learners of programming have when they first attempt to program - algorithms and data structures. A working knowledge of these concepts is assumed in this approach. Therefore can this approach be considered inappropriate for absolute beginners of programming. Still, this approach finds its application in the teaching project in chapter 5, yet only at a later point in time, around the third week of the project. At this point the instruction shifts from presenting abstract concepts (like data structures by means of music) to concrete implementations and syntactical rules of the MidiCSD language.

4.2.2 Cognitive objectives taxonomy

A second approach presented by Kaasbøll is the *cognitive objectives taxonomy* as illustrated in figure 4.2.[66]

Here, the approach is much more intuitive: learners first of all run the program. They see the output and get curious - furthermore, this is an ideal starting point for introducing a propositional network as defined in chapter 2.2. The learners then start reading the code. Certainly, this step needs some preparation that enables the learners to understand the syntax, but with syntactically simple languages like Logo, also this step is very intuitive and easily accessible to learners. Moreover, it also touches upon a point discussed in section 2.4 - reading the code is an active way of learning and constructing knowledge. The third step then involves changing the code and observing the results. Finally the learners should have gained enough

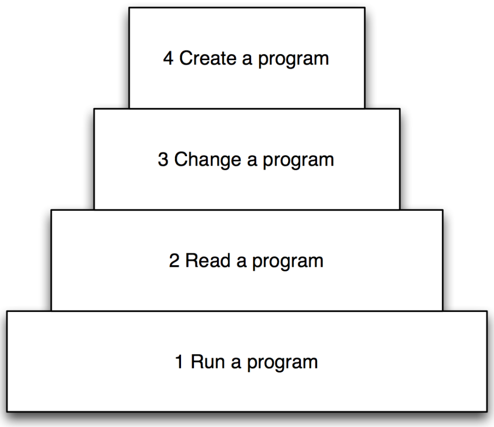


Figure 4.2: Cognitive objectives taxonomy

experience to create a program by themselves. These exact steps are followed in the teaching project. The advantage of this method is that all learning theories are perfectly incorporated. From behaviorism, the the 'drill'-aspect is included when it comes to reading programs - learners do it until they understand it. In connection, the constructionist part is the strong self-reliance and active involvement in the deconstruction of programs and this approach provides a more natural way to get in touch with programming. The learners slowly 'grow' into the world of programming and can easily construct a propositional network. Even more, this method can present complex and abstract topics in a simple and straightforward manner, which allows, according to Piaget's theory, the application of the method also in a younger learner group.

4.3 Music and programming - a contradiction?

The question now arises, as it has been clarified how to proceed in teaching programming, how to deliver the concepts and topics related to programming. Obviously, this is a very complex topic and requires a well prepared approach to avoid demotivation already at the beginning. Traditional programming courses usually focus on teaching the basic syntactical rules of a language and then some exercises, the meaningfulness of which can be argued about. Very often then, learners complain that they do not see the practical use of the exercises they do and lose interest (a fact that the author of this paper has also experienced in his teaching). Once a start is made, interest builds up automatically, and a feeling of accomplishment arises (this again relates to behaviorist theories, cf. chapter 2.1).

The hypothesis of this thesis is that music can serve as a means to teach abstract programming principles to learners who have no programming background whatsoever. As already mentioned, a teaching project is discussed in chapters 5 and 6 and the workbook along with supplementary material for the teacher is to be found in the appendix. This chapter focuses on the theoretical didactical implications.

4.3.1 An argument in favor of music

It can now be argued why music is a suitable means to teaching programming. Guzdial and Soloway make a very good case for music: it appeals much more to learners than traditional text based exercises. For an age group they label "Nintendo generation",[67] simple text based introductory examples like the famous 'Hello World!' are not suitable for eliciting interest. In fact Guzdial and Soloway claim:[68]

> Let's consider a popular textbook for CS1 today, Deitel and Deitel's *Java: How to Program* [. . .] The first program discussed in Deitel and Deitel is producing a line of text, akin to "Hello World." The second places the text in a window. The next few produce numeric outputs in windows and then input numbers and generate calculator types of responses. Would one expect these kinds of exercises to be the ones to engage the MTV generation? Such exercises are exactly what the AAUW report describes as "tedious and dull."

This certainly poses a significant problem for a didactical design of programming instruction. Yet Guzdial and Soloway also have an answer for this problem. Years ago, when computers could not offer today's extent of multimedia capabilities, a textual approach seemed enough to learn about programming and data structures. However, the current generation of learners is used to the computer as a multimedia machine that can easily process music, video and graphics at a very high level. Therefore programming, even introductory programming has to live up to this standard to a certain extent.[68]

An example from teaching practice: a student of a colleague was quite excited about learning how to program when he found out that the computer game *World of Warcraft* is written in C++. He strongly asked to be instructed in this programming language so that he could also program a similar game. Of course, one cannot start teaching programming with C++, an extraordinary complex language, and even then game programming is far out of reach for beginners. Yet still, it nicely shows the incentives that are relevant for learners today. Computer science teaching should take notice of that.

4.3.2 Data structures in music

The previous argument has shown that music can indeed serve as a viable means to teaching programming in terms of interest and student motivation. Now one can show that music can not only elicit interest but also convey significant principles of programming. Data structures are a very good example as sheet music basically is a way to organize data - musical data, i. e. tones. Of course, the notation is quite different to programming notation, but it can be shown that one can easily switch between the two models. A look at the following simple tune in figure 4.3 already shows several things one can notice form a computer scientist's point of view.

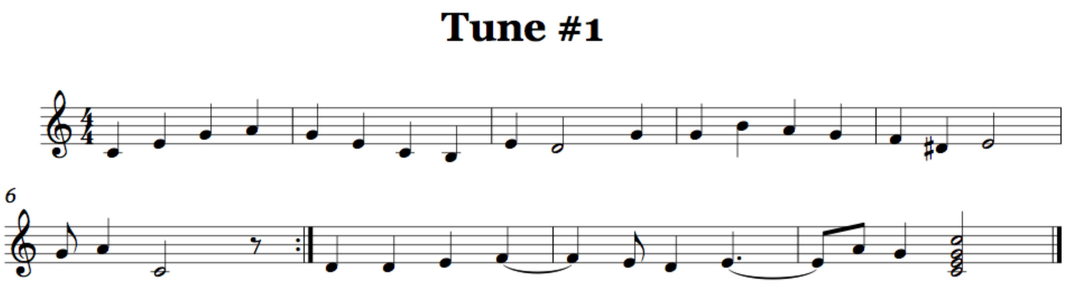


Figure 4.3: A simple tune

This song features a series of notes with different pitches and durations. Sheet music notation can be quite complex, yet here one can easily observe that the sequence of notes is managed in measures. Furthermore, it seems that this tune is in c-major and has a 4/4 time signature. Finally, one part of the song is repeated. In terms of data structures, this song does not seem to be overly complex. Still, one can extract structural information shown in figure 4.4 from the piece.

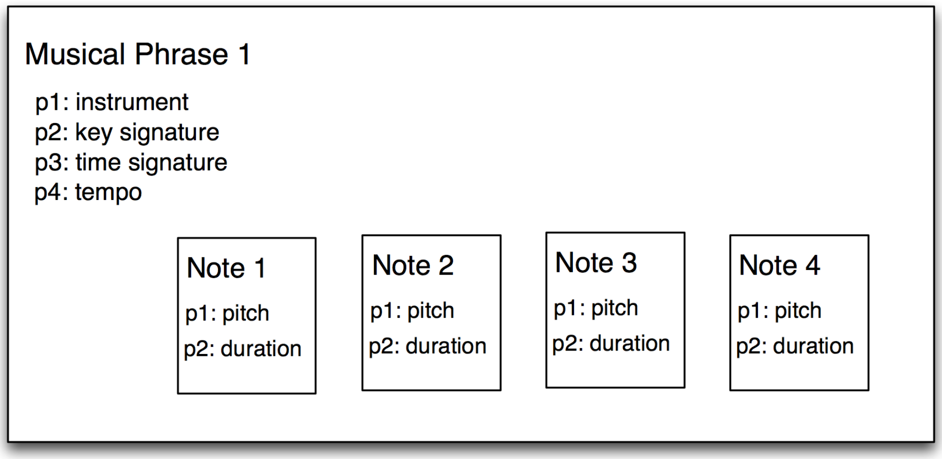


Figure 4.4: Objects in music

This image shows the formal structure of the tune, as one would define it from a programming perspective: first, there is a musical phrase that has certain properties. The time signature, key signature or the instrument that is used to perform is can be such properties. A phrase can be a whole piece, but not necessarily. Considering that for example the key signature can change in a song, two phrases would be necessary then. A phrase can therefore also be just one measure. The phrases contain notes (at least one note) which in turn have properties again, pitch and duration being the two properties that exactly define a note. Hence, the tune above can be described using two phrases (figure 4.5).

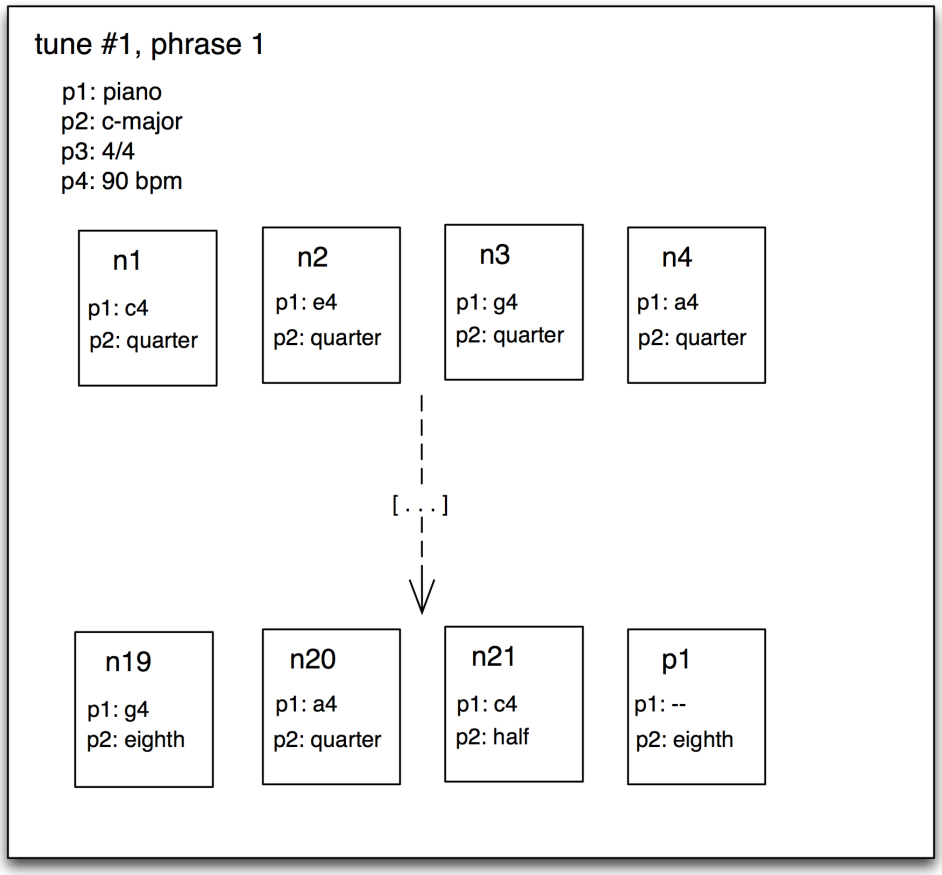


Figure 4.5: Phrase 1, tune #1

This phrase illustrates the first part of the tune, up to the bar indicating the repetition. This phrase, or object, is then used a second time, in programming one could classify this as a loop. After the the repetition the song continues with the second phrase (figure 4.6).

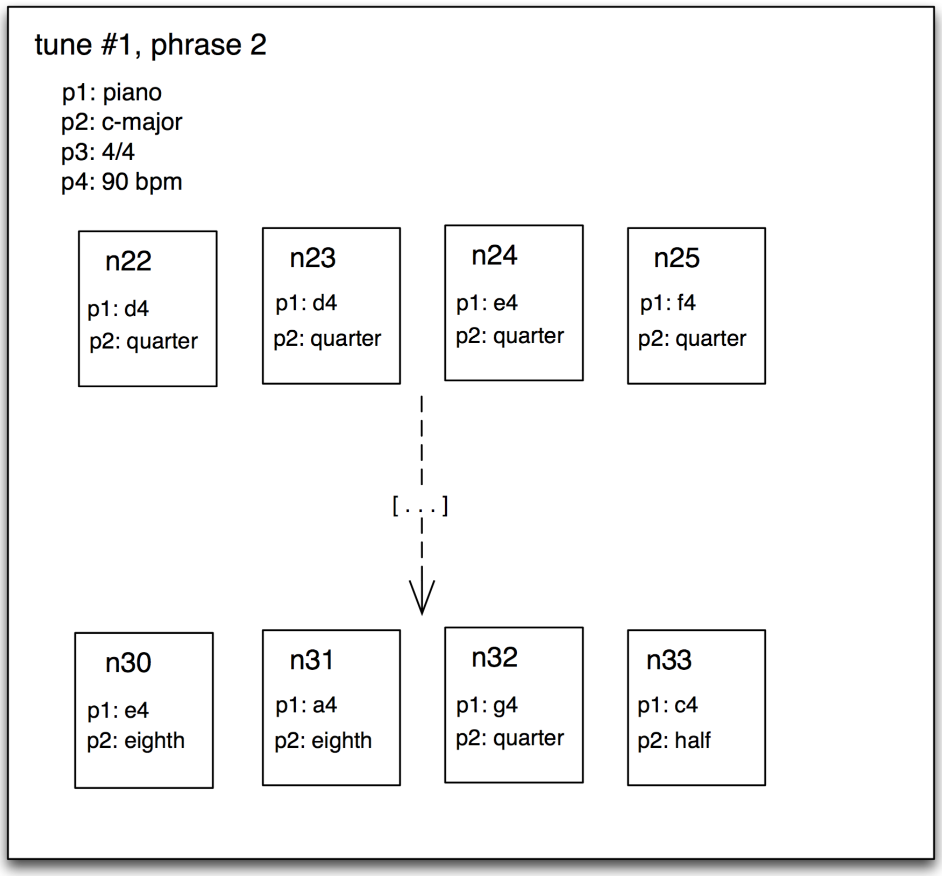


Figure 4.6: Phrase 2, tune #1

The song then continues until the end. The important point to notice here is that music can essentially be seen as an object-oriented data structure that can be used flexibly to create and describe pieces of music. The constructors for the objects/phrases (implementing an example method) above can be described in the following pseudo-code.

```
define: phrase(key, time, tempo, instrument){
        key = cmajor
        time = 4/4
        tempo = 90 bpm
        instrument = piano
        notes = []

        function addnote(pitch, duration){
         notes += new note(pitch, duration)
        }

        play{
                transfertomidioutput
        }

        function transpose_song(offset){
                if offset > 0
                        key_signature += offset;
                else if offset < 0
                        key_signature -= offset;
        }
}
```

A similar pseudo-code defines a note:

```
define: note(pitch, duration){
        pitch = c
        duration = 1/4

        function transpose(offset){
                if offset > 0
                        pitch += offset
                else if offset < 0
                        pitch -= offset
        }
}
```

Finally, the following simple code shows the creation of the computational representation of the example song.

```
class tune#1{
        main(){
                phrase1 = new phrase(c-major, 4/4, 90, piano)
                phrase2 = new phrase(c-major, 4/4, 90, piano)

                phrase1.addnote(c4, quarter)
                phrase1.addnote(e4, quarter)
                phrase1.addnote(g4, quarter)
                phrase1.addnote(a4, quarter)
                phrase1.addnote(g4, quarter)
                phrase1.addnote(e4, quarter)
                phrase1.addnote(c4, quarter)
                phrase1.addnote(b4, quarter)
                phrase1.addnote(e4, quarter)
                phrase1.addnote(d4, half)
                phrase1.addnote(g4, quarter)
                phrase1.addnote(g4, quarter)
                phrase1.addnote(b4, quarter)
                phrase1.addnote(a4, quarter)
                phrase1.addnote(g4, quarter)
                phrase1.addnote(f4, quarter)
                phrase1.addnote(dis4, quarter)
                phrase1.addnote(e4, half)
                phrase1.addnote(g4, eighth)
                phrase1.addnote(a4, quarter)
                phrase1.addnote(c4, half)
                phrase1.addnote(p, eighth)

                phrase2.addnote(d4, quarter)
                phrase2.addnote(d4, quarter)
                phrase2.addnote(e4, quarter)
                phrase2.addnote(f4, quarter)
                phrase2.addnote(f4, quarter)
                phrase2.addnote(e4, eighth)
                phrase2.addnote(d4, quarter)
                phrase2.addnote(e4, quarterextended)
                phrase2.addnote(e4, eighth)
                phrase2.addnote(a4, eighth)
                phrase2.addnote(g4, quarter)
                phrase2.addnote(c4, half)

                repeat(phrase1.play)
                phrase2.play
        }
}
```

Of course, this is only non-working pseudo-code, far away from an actual implementation, therefore no particular programming language is represented here. Still, it shows how musical structures translate into objects, on which operations can be performed (like the repeat - command). Hence one can say that the data structure of music can indeed represent data structures needed for programming and can constitute a promising teaching approach.

MidiCSD

The question now arises how to implement musical operations on the computer. Clearly, beginners cannot directly program in languages like Logo, Java or Scheme. A simple note in the impromptu environment for Scheme looks like this:[69]

```
(au:clear-graph)

(define piano (au:make-node "aumu" "dls " "appl"))
(au:connect-node piano 0 *au:output-node* 0)
(au:update-graph)

(play-note (now) piano 60 80 (* 1.0 *second*))
```

Whereas this might still be a construct that may be possible to teach, a simple extension of the code to play a full scale already renders the code very complex:[69]

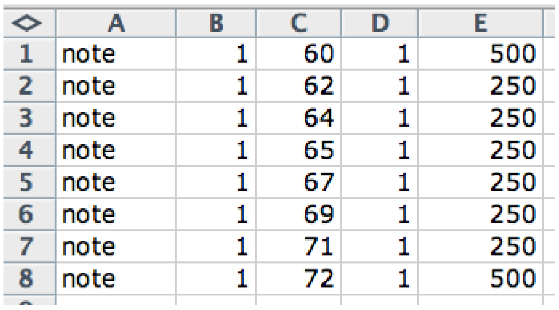
```
(au:clear-graph)
```

```
(define piano (au:make-node "aumu" "dls " "appl"))
(au:connect-node piano 0 *au:output-node* 0)
(au:update-graph)

(au:print-graph)

(define pitches '(60 62 64 65 67 69 71 72))
(define dynamics '(80 80 80 80 80 80 80 80))
(define rhythms '(0.5 0.25 0.25 0.25 0.25 0.25 0.25 0.5))

(print pitches)

(define play-sequence
   (lambda (time inst plst dlst rlst)
      (map (lambda (p d r)
             (play-note time inst p d (* r *au:samplerate*))
             (set! time (+ time (* r *au:samplerate*))))
        plst
        dlst
        rlst)))

(play-sequence (now) piano pitches dynamics rhythms)
```

This obviously is not code that one can expect beginners to understand. Therefore, for simplicity reasons and to provide an easy introduction to programming operations, the actual complex source code implementing the musical structures and operations has to be presented in an easily accessible way. This is where MidiCSD comes in.[70]

Firstly, it provides a familiar working IDE for most learners, MS Excel is an application that is quite common and most learners are familiar with it (to the extent that is necessary for using MidiCSD).

Secondly, its graphical user interface is relatively easy to use. As the name says, MIDI phrases are defined in tables, which are constructed using the *phrase language*, and operations on phrases can be taken from a set of commands called *macro language*. These constructs are fairly easy to learn but still provide the learner with a powerful tool to create and modify MIDI sound files. The above defined scale in Scheme then would look as follows in MidiCSD:



	A	B	C	D	E
1	note	1	60	1	500
2	note	1	62	1	250
3	note	1	64	1	250
4	note	1	65	1	250
5	note	1	67	1	250
6	note	1	69	1	250
7	note	1	71	1	250
8	note	1	72	1	500

Figure 4.7: Scale in MidiCSD

The didactical approach to this is illustrated in the workbook (cf. appendix), as the table at first sight might look somewhat confusing to learners who are unfamiliar with the MIDI format (e. g. the purpose of the channel). Thus the teacher has to do some preliminary work; illustrating the musical structures and drawing parallels to MIDI (in terms of pitch numbering and duration). Still, MidiCSD promises a careful introduction to musical programming and programming as a whole.

4.3.3 Further possibilities of music in CS teaching

If foundations for algorithms and data structures are laid, one can continue learning a particular programming language. For this purpose, several approaches are possible in various programming environments.

Programming in Squeak

Guzdial and Soloway propose a interesting approach to teaching computing: multimedia programming in Squeak.[71] Squeak is a programming language and environment based on Smalltalk that has special features to work with multimedia content.[72] Guzdial and Soloway use Squeak to replace the 'Hello World!' exercise with a voice recording of the learner himself, as Squeak offers sound recording.[71] The equivalent to the 'Hello World!' procedure would then be the learners attempting to play their recorded message. This is, thanks to the rather simple syntax of Smalltalk rather easy:[71]

```
(SampledSound soundNamed: mySound) play
```

Here, the the object-orientation is far less a problem than with Java or C++, as the syntax is simple enough to allow a good understanding of the concept (with the foundation with MidiCSD). A highly convenient feature is the ability of Squeak to convert the recorded sound into an instrument:[71]

```
((SampledSound soundNamed: mySound) pitch: c) play
```

This merely requires a simple extension of the code and it is still fairly intuitive and logical with regard to is structure. Consequently, when the basics of the language are learned, the users can create more complex programs that still do not require a lot of code (which is one of the major advantages of Smalltalk/Squeak over C++):[71]

```
forfreq: freq amplitude: amp duration: seconds

| sr anArray pi interval samplesPerCycle maxCycle rawSample |

sr := SoundPlayer samplingRate.
anArray:= SoundBuffer newMonoSampleCount: (sr * seconds).
pi := Float pi.
interval := 1 / freq.
samplesPerCycle := interval * sr.
maxCycle := 2 * pi.

1 to: (sr * seconds ) do: [:sampleIndex |
        rawSample := ((sampleIndex/samplesPerCycle) * maxCycle) sin.
        anArray at: sampleIndex put: (rawSample * amp ) rounded.
].
^ anArray
```

This method describes the generation of a monophonic sound buffer array which is filled with a sine wave of a given frequency, maximum amplitude and its duration in seconds.[71] Due to the pure object-orientation, the variables can be defined easily (this again continues the object-oriented path that starts in MidiCSD) and a for-loop completes the array containing the sound. Another method enables the combination of two sound buffer (arrays) with the same index values:[73]

```
combine: soundbuffer1 and: soundbuffer2

| newsound |

(soundbuffer1 size) = (soundbuffer2 size)
ifFalse: [^ self error: Sound buffers must be of the same length].
newsound := SoundBuffer newMonoSampleCount: (soundbuffer1 size).

1 to: (soundbuffer1 size) do:
        [:index | newsound at: index put:
        (soundbuffer1 at: index) + (soundbuffer2 at: index)].
^ newsound.
```

The former example corresponds to the definition of a musical phrase (as in MidiCSD's phrase language) and the latter illustrates an operation on phrases that corresponds to the macro language. Thus one can see programming in Squeak/Smalltalk as the next step to bring learners carefully from an simplified special-purpose language like MidiCSD to a universally useable fully featured programming language that incorporates many tools and concepts necessary for other mainstream languages (C++, Java, C#, Objective-C, Scheme, Lisp). Guzdial and Soloway believe that this approach to elementary concepts of programming like array manipulation is strongly preferred by students.[73] Furthermore, Squeak offers much more features with regard to multimedia - one can not only look at the waveforms and play the sounds, even Fourier analyses are possible.[73] Clearly, this it not introductory programming anymore, but it shows the enormous potential the language offers in teaching.

Design patterns

It has been shown that basic functionalities can also be taught by using music and multimedia. One might now argue that music can also serve as a means to teaching advanced topics in programming, i. e. design patterns. Hamer is in favor of this approach as he claims:[74]

> Attempts at teaching design patterns in the same style as data structures and algorithms are doomed to failure. For teaching data structures, it works well to present a sample of the major forms (e.g., array-based lists, linked lists, hashing, and binary search trees). Data structures are "solutions in search of a problem." Design patterns, on the other hand, are tools for doing design (i.e., generating solutions). To learn a design pattern, students must experience the use of the pattern in practice and relate the pattern to other techniques.

It is therefore worth considering to approach such a teaching situation with music. Hamer proposes a teaching project that requires learners to write the complete code by themselves (in Java, although implementations in C++ or C# are of course also possible).[74] It is therefore necessary that the learners already have a certain level of proficiency in not only basic programming concepts, but also object-oriented programming languages. Java is a complex language that imposes a strict set of syntactic rules upon the programmer, thus at least some experience in programming in Java is absolutely necessary in order to be able to fully understand and benefit from the teaching of such advanced topics like design patterns.

Hamer notes that the project focuses on musical composition and enables learners to express their musical talent and nicely shows the connection between programming and art.[74] The project shall address the following issues:[74]

- exploration of the design patterns *Composite*, *Decorator*, *Factory* and *Visitor*
- further deepen the understanding by emphasizing the analogy between program structure and musical structure, which includes:
  - sequential composition (cf. statement sequencing)
  - parallel composition (cf. threaded code)
  - musical repeats (cf. loops)
  - variant endings (cf. conditional statements)
- develop learners' skills of abstracting patterns and identifying underlying structures
- present a connection between structural design patterns and context-free grammars
- the application of formal methods (e. g. giving proofs of equivalence of various musical forms)

Points two and three are to a certain extent already touched upon in MidiCSD, as it too illustrates loops, sequential and treaded code, but the other points are new and quite demanding to learn when studied solely on a theoretical level. Especially the proofs of equivalence are a completely new topic that is an interesting new concept, yet in as this would expand the argument of this thesis to logic and theoretical computer science this section concentrates solely on the musical interpretations of the design patterns.

Hamer begins by outlining a musical structure. It is similar to the structure illustrated in section 4.3.2, extending the music model slightly (figure 4.8).[75] The structure defined here is described using the design patterns *Composite* and *Decorator* and consists of **Notes** (again with the parameters pitch and duration) and

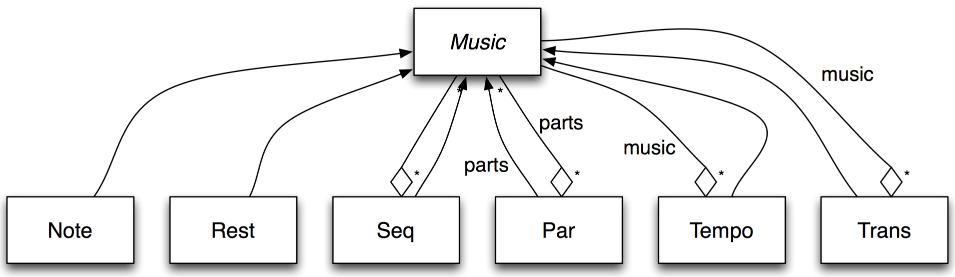


Figure 4.8: UML diagram for music

Rests (which only have a duration).[75] Notes and rests can be assembled to larger units using either the **Seq** (for a sequence of notes) or the **Par** (for a parallel arrangement of notes, i. e. a chord) constructor.[75] The abstract class **Music** then allows the arbitrary nesting of musical constructs.[75]

The musical term can then be modified by four *Decorator*-classes (**Tempo**, **Transpose**, **Instrument** and **Phrase**). The purpose of the first three classes is fairly obvious. The **Phrase**-class is an attempt to describe musical phrasing algorithmically.[75]

The *Factory* design pattern is then used for constructing notes and rests. The middle C could be constructed like this:[75]

```
Note middle_C_quaver = new Note(32, 0.25);
```

To facilitate writing notes, a number of auxiliary functions can be constructed:[75]

```
Note c(int octave, double duration) {
        return new Note(octave 12, duration);
}
Note d(int octave, double duration) {
        return new Note(octave 12 + 2, duration);
}
//- etc.
```

```
Note sharp(Note orig) {
        return new Note(orig.pitch( ) + 1, orig.duration( ));
}
Note flat(Note orig) {
        return new Note(orig.pitch( ) - 1, orig.duration( ));
}
```

Which then allows the middle C (quarter note) to be constructed like this:[75]

```
Note middle_C_quaver = c(4, 0.25);
```

It is clear that these operations and functions demand a certain level of programming skill, yet it nicely illustrates the use of design patterns. Hamer also briefly addresses the use of a context free grammar, as he claims that it offers a more concise description of the musical structure than the UML diagram.[75] In addition, Hamer notes that it deepens the understanding of students when they are exposed to alternative forms of notation.[75] According to Hamer, an abstract grammar for music therefore would look as illustrated in figure 4.9.[76]

Music ::= Note *Pitch Duration*
| Rest *Duration*
| Seq Music Music ...
| Par Music Music ...
| Tempo *Ratio* Music
| Transpose *Offset* Music
| Instrument *Name* Music
| Phrase *Attribute* Music

Figure 4.9: A context-free grammar for music

Finally the piece, which is described and its objects implemented, has to be performed. As with MidiCSD, Hamer suggests the MIDI standard as music output.[76] In order to convert a **Music** term (a tree) into MIDI (a linear sequence), Hamer notes that the structure needs to be flattened by a certain operation, which can be written with a recursive traversal (although various complications arise when the required MIDI headers and event formats are generated).[76] To avoid these complexities, a (simpler) intermediate form is generated first, which results in two translations, but the second (encoding the details in MIDI) can usually be provided to the students as a callable library.[76]

The students then have to write a function that generates a list of musical 'events' for each **Note**, which calculates the absolute time, instrument, pitch and duration.[76] A 'performance context' maintaining the current time, instrument, pitch, offset and tempo factor is required for an efficient solution.[76] While the *Decorator* objects can modify and restore the relevant part of the context, **Seq** and **Par** update the current time.[76] An outline of the code looks like this:[76]

```
class Context {
        double time;
        int volume;
        String instrument;
        double dt;
        int dp;
        SortedSet events;
}
...
class Tempo {
        Music part;
        double tempo;

        void perform(Context ctx) {
                ctx.dt /= tempo;
                part.perform(ctx);
                ctx.dt *= tempo;
        }
...
}
```

Hamer notes that at this stage, the *Visitor* pattern can be introduced, which can be presented as an example of a failed pattern, as the assumed advantages of adopting the pattern do not arise.[76] Hamer claims that exposing students to situations where patterns do not fit is quite important - *Visitor* involves the collection of all virtual **perform** functions inside the **Context** class.[76] A generic **visit** method is left in place of the virtual functions.[76] This method selects the appropriate (overloaded) **perform** method from the **Context** class.[76] The full pattern then requires that the new **perform** methods have to be placed in an interface (with a generic name such as **accept**).[76] At the end, the code does the same but becomes more difficult do understand.[76]

To complete the discussion of the elements given in the grammar above, Hamer notes that a **Phrase** modifies the **Events** that are generated from the enclosing **Music** object.[76] He gives the example of the attribute 'MF' (denoting *mezzo-forte*, medium loudness), which could set the volume of each event to 110% of normal.[76] Analogously one can implement other musical performance elements like a *crescendo* or *diminuendo*.[76]

4.4 Summary

In this chapter a theoretical concept for teaching programming has been created. In order to have a general background it is argued that of the two approaches to teaching programming, the semiotic ladder and cognitive objectives taxonomy, the latter is preferable as it provides a more natural access point not only for teaching a particular programming language but also general concepts like algorithms and data structures. For this purpose one can choose music as a means to present the said concepts, as it can be shown that musical structures directly reflect back on data structures, particularly for object-oriented programming. A concrete implementation of the cognitive objectives taxonomy and music is found in MidiCSD, a user-friendly simplified programming environment that provides a good starting point for basic programming concepts, but also abstract constructs like nested objects. Using MidiCSD as a starting point, one then can use music in further projects to teach fully featured languages like Smalltalk. It is even possible to teach advanced topics of programming such as design patterns.

5 Teaching project

The aim of this chapter is to put the theories and hypotheses developed into practice. Given the discussion about teaching programming (cf. chapter 4) and learning theories, especially constructivism and the theory of cognitive development (cf. chapter 2.3 and 2.4), the next step is to test the developed hypothesis in practice. MidiCSD serves as a means to do this and the following description of the teaching project investigates whether MidiCSD is a viable example in CS teaching.

5.1 Prerequisites

The project intends to introduce learners to elementary concepts of algorithms, data structures and programming. Therefore, no knowledge of programming and its related fields is assumed. However, a certain level of knowledge of music theory is necessary, i. e. the learner needs to be able to read music. Moreover, a certain extent of ability to use the computer is needed. MidiCSD is an MS Excel add-on, but it is not necessary to be familiar with spreadsheets to use MidiCSD.

5.2 Target group

The project is suitable for learners above the 5th form (in the Austrian high school system). Learners at this age (15+) already have some knowledge of how to work productively with the computer and its applications. Still, it might be possible to carry out the project with younger learners. This cannot be defined generally and depends on the learning progress of the individual class. It can therefore be perfectly suitable to use the project with 10 to 14 year olds (some simplification of the project paper might be necessary) if the school has a strong focus on computer science.

Additionally, the project can also be implemented at university, which requires a few changes. The timeline is different for students and the final project is a bit different - it is not a composition competition, but rather a group project that is assessed.

5.3 Learning targets

The learners should

- understand the concept of data structures by means of the structure of music
- understand the connections between sheet music and music in the MIDI file format
- understand the basics of object-oriented data structures
- understand how interpreter based programming languages work
- understand how a compiler works
- understand how the macro language of MidiCSD operates on phrases/objects and how this is related to assembler-based programming
- be able to compose and implement a piece of music in MidiCSD and compile it

5.4 Timeline

5.4.1 At school

The timeline depends strongly on the amount of computer science lessons the learners have per week. If instruction takes place only once a week, the duration of the project can be estimated to be about five weeks. Especially toward the end of the project, where actual development of songs takes place, the time consumption can vary strongly, depending on how fast the learners work and make progress.

If instruction takes place several times a week the duration can be shorter (e. g. CS classes at university). Still, three weeks can be considered to be the absolute minimum timeline for this project.

5.4.2 At university

University students can be expected to work and understand the concept much faster than school students. Therefore the duration of the required presentation by the teacher is significantly reduced to a mere introduction to the musical structures and MidiCSD. A fixed timeline can therefore not be given, it depends on how much time the teacher allows the students to explore the possibilities of MidiCSD and to create their individual songs.

5.5 General expectations

The project attempts to illustrate the concepts mentioned in the learning targets above in an indirect way. Instead of directly telling and lecturing learners how a compiler or interpreter works, the objective is to let the learners experience the effects of interpretation and compilation and afterwards give an explanation that is then completely clear to everybody. The method to teaching data structures is similar. By analyzing and working with musical pieces the concept becomes innate in a way that no explicit explanation is necessary. Therefore the project promises to be quite successful in outlining the key concepts of programming to the learners.

5.6 Course structure

The teaching project is a mix of teaching form S1/A2 and S2/A2 as defined in section 3.2.1. Learners basically work in groups most of the time, but occasional lectures of the teacher are necessary, especially for introductory concepts and repetitions.

5.7 Lesson plans (for school)

The lesson plan is a rather flexible suggestion. The accompanying workbook is deliberately written in a school-book style, which allows lessons to be constructed freely around it. An example lesson plan for a duration of five weeks (two double-lessons per week) could look as follows.

5.7.1 First week

In the first week, the existing knowledge of music is put into the context of computer science (table 5.1). The teacher points out how a piece of music is constructed and how this is relates to the concept of data structures. As this is the entry point for the whole project, it is particularly important that all learners understand the issue at hand. If needed, additional time should be spent in order to ensure that all students can analyze a piece of music in terms of its musical structure.

The next step is to introduce students to the MIDI file format and how to 'translate' simple songs from sheet music format to the MIDI format. This should be shown with the help of MidiCSD. A solid understanding of the connections between the two formats is crucial, as the rest of the project is constructed on the assumption that this concept is clear to everyone.

Time	**Activity**	**Interaction**	**Media**
20	description of music structure	lecture	
10	analysis of an example song	group work	workbook
10	comparison of results	open class	
20	introduction to the GUI of MidiCSD	lecture	
10	analysis of example song in MidiCSD	group work	
20	discussion of the MIDI file format	group work	workbook
10	revision	lecture	

Table 5.1: Lesson plan week 1

5.7.2 Second week

The second week of the project deepens the knowledge of MIDI operations (table 5.2). Significant importance is to be attributed to the topic of the use of relative time. This might not be easily accessible for all learners, therefore a thorough explanation, visualization and exercise is necessary. This may take up more time, yet this time is well spent as learners can then create complex polyphonic songs, which widens the possibilities in the final project activity greatly. For this purpose it may be quite helpful to distribute handouts with exercises on this matter and refer to the workbook.

The focus then shifts to the musical phrases and the operations that can be performed with them. Particularly the illustration of phrases as objects that have certain properties is important, as it introduces learners without too many abstract explanations to object-orientation, an important concept of programming.

MidiCSD is then used to demonstrate the principle of assembler-programming. The macro language of MidiCSD is used to perform operations on musical phrases. An emphasis on the thorough explanation of how the macro language works is strongly recommended, as it represents the first contact of learners with actual commands of a programming language and its results. The learners therefore first try to figure out which effects the macro language might have on the phrase (as described in the workbook) and then the concept is discussed in an open class.

Time	**Activity**	**Interaction**	**Media**
25	discussion on relative time in MidiCSD	lecture and group work	workbook
20	phrase operations	group work	workbook
15	macro language of MidiCSD	group work	workbook
15	discussion of macro language	open class	
15	activity on compiling and interpreting code	group work	workbook
10	revision and Q & A	lecture	

Table 5.2: Lesson plan week 2

5.7.3 Third week

Time	**Activity**	**Interaction**	**Media**
10	the song-development process	lecture	
30	activity: song creation	group work	
10	implementation in MidiCSD	group work	workbook
25	canon-exercise	group work	workbook
15	discussion and presentation of songs	group work	
10	revision	lecture	

Table 5.3: Lesson plan week 3

The third week turns to more and more group work (table 5.3). The teacher introduces and summarizes the lesson, but the main work is done in groups. The learners already have the necessary skills to develop a MIDI song by themselves and in the spirit of constructionist learning the process should be as independent from the teacher as possible. The learners should bring a simple piece of music with them to work on and the process from song translation to MIDI and its implementation as a simple phrase in MidiCSD should be no problem. When first attempting to create the canon with help of the macro language, several problems can appear that require the assistance of the teacher. Still, due to its similarity to an example in MidiCSD, the problem should be solvable for all learners. Finally, the songs of the project groups should be presented in class. This serves as a test run for the compositions the learners will create in the project later on.

5.7.4 Fourth week

The fourth week (table 5.4) introduces the last feature the learners can use in the project: polyphonic songs with help of relative time. The concept is briefly revised and then applied in the example songs of the previous lecture. As all relevant concepts have been presented and worked through, the project work in the groups can start. The rest of the week's lesson is dedicated to finding or composing a song and completing the first steps of the development process.

Time	Activity	Interaction	Media
10	revision	lecture	
30	applying polyphony with relative time	group work	
10	discussion of results	open class	
50	project work in the groups	group work	

Table 5.4: Lesson plan week 4

5.7.5 Fifth and final week

The final week of the project is entirely focused on the project work in the groups (table 5.5). The learners are advised to continue working on the song between the fourth and fifth week at home and the final project work is done in class. The last part of the lesson is used for presenting the results of the project work.

Time	**Activity**	**Interaction**	**Media**
80	project work in the groups	groups work	
20	presentation of the results	open class	

Table 5.5: Lesson plan week 5

6 Project evaluation

6.1 Sample group

The teaching project was evaluated with a group of 12 university students, all of which have a musical background. As can be seen in figure 6.1, female students were in the majority (7 females to 5 males).

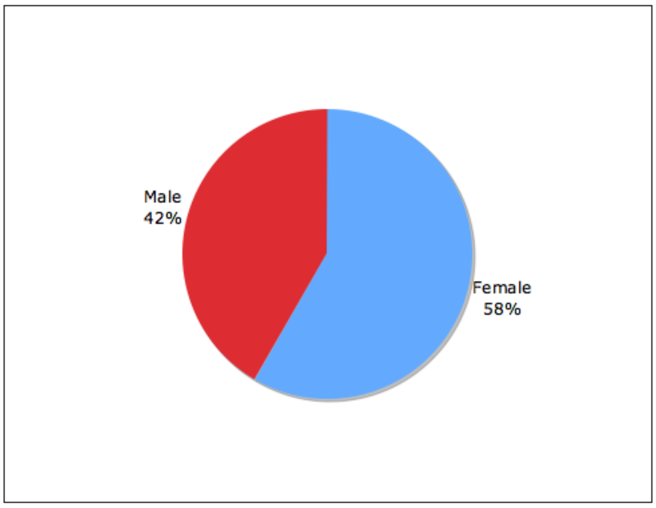


Figure 6.1: Gender ratio

The workbook was not used for the evaluation, as it is principally aimed at high school students. Instead, an introductory presentation and a handout with tasks (ranging from quite simple to more complex implementations) were used. These materials are to be found in the appendix. The students worked together in groups of two (one all-male, two all-female and two mixed groups) and, in order to contrast group results to individual results, one female and one male individual working on their own. This resulted in 7 MIDI spreadsheets that were handed in for evaluation. The students' academic background is almost in all cases not tied to music - only two of the students study musicology. The areas of study are divided as illustrated in figure 6.2.

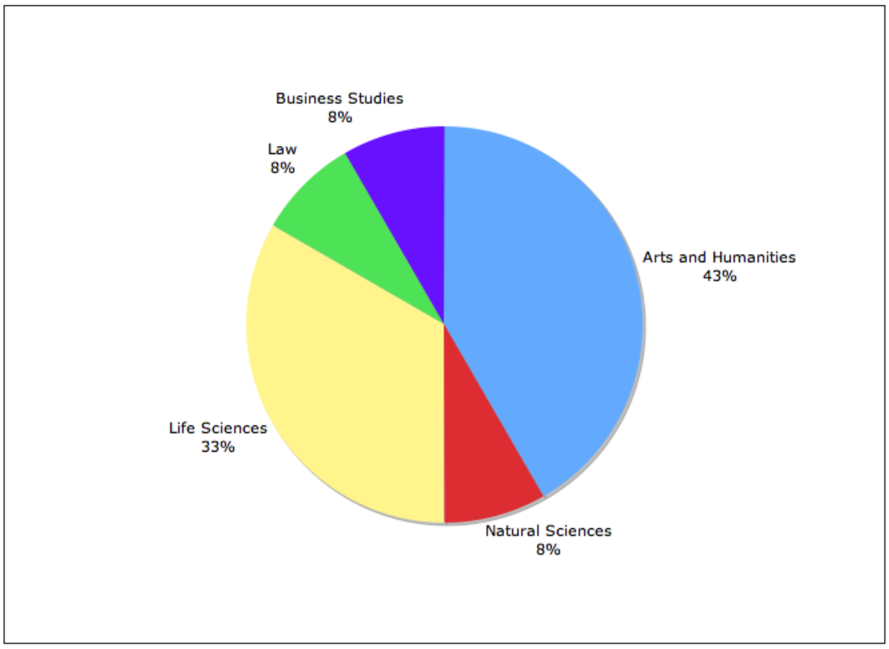


Figure 6.2: Study areas of sample group members

Still, it needs to be mentioned that almost all of the students have strong ties to music in their spare time, half of them are choir singers and the others also play an instrument or have musical talent or knowledge of music theory to some extent.

6.2 Teaching sequence

Due to time constraints, not the entire possibilities that MidiCSD offers have been tested. The teaching sequence focused on an understanding of musical pieces and the notes they contain as objects and the actions that can be performed on them using the macro language. Therefore an introductory presentation was given to illustrate the key concepts mentioned as well as the syntax of the macro language. With regard to active learning, concepts like the relative time and macro language syntax were discovered by giving students time to explore the examples in MidiCSD themselves.

After the presentation, the students divided into groups and worked on the handout to solve the exercises. Finally the results in form of Excel spreadsheets were handed in for evaluation along with some statistical data.

6.3 Selected tasks

The outcome can generally be interpreted as being very successful. Very few problems occurred during the evaluation and the outcome presents very interesting findings. On figure 6.3 the solved exercises are shown, with no. 11 being macro codes not requested on the task sheet.

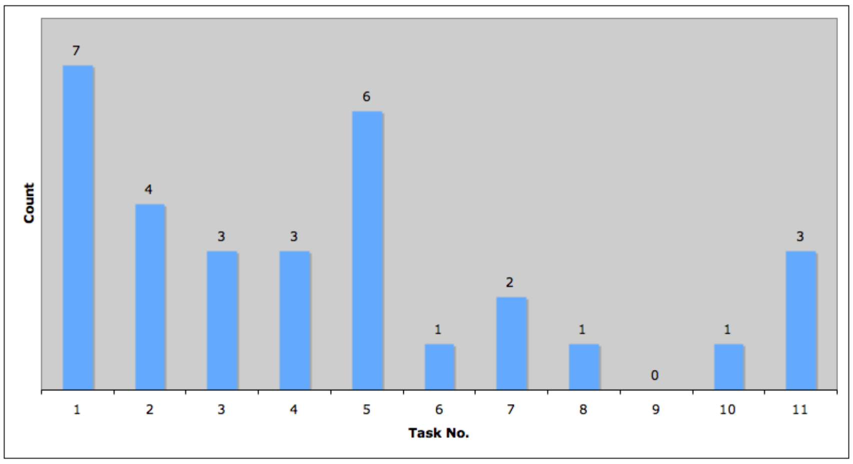


Figure 6.3: Choice of tasks

Obviously in all 7 instances, the groups/individuals managed to translate the sheet music to its MIDI representation in task 1. Most groups solved the more easy tasks 2-5, yet a few groups tried and succeeded in solving the more advanced tasks 6, 7 or 8. Task 10, the creation of the MIDI file was accomplished once with the teacher's help. Sound effects summarized under no. 11 are for example speed changes and time shifts.

6.4 Significant results

6.4.1 Naming conventions

In order to distinguish between the groups the following convention shall be utilized: I-M (individual, male), I-F (individual, female), A-F 1/2 (all-female groups), A-M (all-male group) and Mixed 1/2 (mixed groups).

6.4.2 Task 1 and 2

Every group managed to produce a solution for the midi table of the song, which is shown in table 6.1. The table is taken from the spreadsheet that was handed in by I-M. No group found it difficult to assign the correct pitch numbers and duration values to the table. Because of the simple syntax of the MIDI phrase, task 1 was solved very quickly. Several groups included chords in the phrase, as requested in task 2. An example is shown in table 6.2 (taken from A-F 2), which already required a deeper knowledge of music theory, especially when constructing subdominants and dominants. Still, also this task posed no great problem to the groups who chose to work on it.

reltime	**saints**				
0	note	1	60	1	250
250	note	1	64	1	250
250	note	1	65	1	250
250	note	1	67	1	1250
1250	note	1	60	1	250
250	note	1	64	1	250
250	note	1	65	1	250
250	note	1	67	1	1250
1250	note	1	60	1	250
250	note	1	64	1	250
250	note	1	65	1	250
250	note	1	67	1	500
500	note	1	64	1	500
500	note	1	60	1	500
500	note	1	64	1	500
500	note	1	62	1	1250
1250	note	1	64	1	500
500	note	1	62	1	250
250	note	1	60	1	750
750	note	1	60	1	250
250	note	1	64	1	500
500	note	1	67	1	500
500	note	1	67	1	250
250	note	1	65	1	1000
1000	note	1	65	1	250
250	note	1	64	1	250
250	note	1	65	1	250
250	note	1	67	1	500
500	note	1	64	1	500
500	note	1	60	1	500
500	note	1	62	1	500
500	note	1	60	1	1250

Table 6.1: MIDI table of 'Oh when the saints' (I-M)

reltime	**saints**				
0	note	1	60	1	250
250	note	1	64	1	250
250	note	1	65	1	250
250	note	1	67	1	1250
0	note	1	60	1	1250
0	note	1	64	1	1250
1250	note	1	60	1	250
250	note	1	64	1	250
250	note	1	65	1	250
250	note	1	67	1	1250
0	note	1	72	1	1250
0	note	1	64	1	1250
1250	note	1	60	1	250
250	note	1	64	1	250
250	note	1	65	1	250
250	note	1	67	1	500
500	note	1	64	1	500
500	note	1	60	1	500
500	note	1	64	1	500
500	note	1	62	1	1250
0	note	1	59	1	1250
0	note	1	55	1	1250
1250	note	1	64	1	500
500	note	1	62	1	250
250	note	1	60	1	750
750	note	1	60	1	250
250	note	1	64	1	500
500	note	1	67	1	500
500	note	1	67	1	250
250	note	1	65	1	1000
1000	note	1	65	1	250
250	note	1	64	1	250
250	note	1	65	1	250
250	note	1	67	1	500
500	note	1	64	1	500
500	note	1	60	1	500
500	note	1	62	1	500
500	note	1	60	1	1250
0	note	1	64	1	1250
0	note	1	67	1	1250
0	note	1	72	1	1250

Table 6.2: MIDI table including chords (A-F 2)

6.4.3 Tasks 3 and 4

These tasks required the use of macro language for the first time. Interestingly, the first impulse of many of the groups working on any macro language exercise was to copy and paste the example code already given in MidiCSDDemo and to go on from there. Certainly, as at the beginning everybody was still somewhat insecure about the syntax, most groups decided to work their way from an already known code. Group Mixed 1 presents the following solution for task 3:

```
            clearall
Saints      define      B3:G35
init        define      B37:E38
saints      transpose   4
saints      play
```

It is interesting that although the students copied the *init* phrase from the example songs, as this would not have been necessary, they did not change the instrument. Task 4 is similarly simple and was solved by I-F perfectly:

```
            clearall
saints      define      B2:G34
saints      reverse
saints      play
```

These quite easy tasks were meant as an introduction to the syntax of the macro language and posed no significant problem to any group that worked on this task.

6.4.4 Task 5

Task 5 already required a more complex code, as multiple phrases and instruments were necessary. Although the exercise shows strong similarities to the canon in the example song, it produced several interesting results. Group A-F 1 presents the following solution:

```
            clearall
saints      define        C5:H47
init        define        C50:F53
saints1     copy          saints
saints2     copy          saints
saints3     copy          saints
saints2     rechannel     1        2
saints3     rechannel     1        3
saints2     timeshift     16000
saints3     timeshift     32000
all         copy          init
all         merge         saints1
all         merge         saints2
all         merge         saints3
all         changespeed   2
all         play
```

There are some interesting observations one can make here. The *changespeed* command was used to shorten playback time in order to save time overall. Switching the channel of phrases 2 and 3 is of course necessary in order to match the instruments provided in *init*. The use of the time shift to accomplish the sequential playback of the phrases seems to stem from the example song provided in MidiCSD. Otherwise it would also be possible to write the following code:

```
            clearall
saints      define        C5:H47
init        define        C50:F53
saints1     copy          saints
saints2     copy          saints
saints3     copy          saints
saints2     rechannel     1        2
saints3     rechannel     1        3
```

```
saints1a     copy         init
saints1a     merge        saints1
saints2a     copy         init
saints2a     merge        saints2
saints3a     copy         init
saints3a     merge        saints3
saints1a     play
saints2a     play
saints3a     play
```

This avoids the need to sum up the overall duration of the MIDI phrase, which can get quite complicated when there are several chords involved, as there are some notes which must not be counted.

6.4.5 Task 6

The task was only partly completed by group Mixed 2. The code they provided was:

```
            clearall
when        define       B3:G36
when        invert       60
when        play
```

This is only a part of the actual solution. Firstly, the inverted phrase should start at the same note as the correct phrase, therefore the argument for the function *invert* should be 67. Moreover, it should be played together with the real phrase, which would have resulted in the following code:

```
            clearall
when        define       B3:G36
when1       copy         when
when2       copy         when
```

```
when1       invert       67
all         copy         when1
all         merge        when2
all         play
```

Group A-M claimed that they had also solved this task, yet unfortunately there is no evidence of this in their spreadsheet.

6.4.6 Tasks 7 and 8

These tasks require a little more work. Task 7 was solved by I-F through addition of another phrase to the original phrase illustrated in table 6.1. The bass line added is shown in table 6.3. The final macro code she developed takes the following form:

```
                clearall
MyLittleSong    define      B2:B34
saintsu         define      I2:N30
all             merge       saints
all             merge       saintsu
all             play
```

There are two errors in this code - first, the defined area of the phrase *saints* contains a typing error. The defined area cannot possibly be B2:B34, rather B2:G34. This is why the phrase name changes to `MyLittleSong`. Second, there is a syntax error in line 4: `merge` needs to be replaced with `copy`. This was one of the most frequent mistakes during the evaluation. The students found it difficult to understand why for the purpose of merging two phrases the first command needs to contain the keyword `copy`, the second `merge`. Thus the correct code would have been:

```
                clearall
```

```
saints          define         B2:G34
saintsu         define         I2:N30
all             copy           saints
all             merge          saintsu
all             play
```

reltime	**saintsu**				
850	note	2	55	1	500
500	note	2	60	1	500
500	note	2	55	1	500
500	note	2	60	1	500
500	note	2	55	1	500
500	note	2	60	1	500
500	note	2	55	1	500
500	note	2	60	1	500
500	note	2	55	1	500
500	note	2	60	1	500
500	note	2	55	1	500
500	note	2	59	1	500
500	note	2	55	1	500
500	note	2	59	1	500
500	note	2	55	1	500
500	note	2	60	1	500
500	note	2	55	1	500
500	note	2	60	1	500
500	note	2	55	1	500
500	note	2	60	1	500
500	note	2	55	1	500
500	note	2	60	1	500
500	note	2	55	1	500
500	note	2	60	1	500
500	note	2	55	1	500
500	note	2	60	1	500
500	note	2	55	1	500
500	note	2	60	1	1000

Table 6.3: Bass line for the original phrase (I-F)

Tasks 2, 7 and 8 at the same time were solved by A-M. They created a very impressive piece of music. The macro code for their MIDI file is quite simple, it only puts four phrases together, but from a musical point of view, the four phrases that create an orchestra version of the song are very well composed. The original song is basically the same as in table 6.1, the only difference is that it is here named *saints1*. The other three phrases are shown in tables 6.4, 6.5, and 6.6 (the last phrase was unfortunately not finished in time). The instruments used are shown in table 6.7.

reltime	**saints2**				
750	note	2	48	1	500
500	note	2	43	1	500
500	note	2	48	1	500
500	note	2	43	1	500
500	note	2	48	1	500
500	note	2	43	1	500
500	note	2	48	1	500
500	note	2	43	1	500
500	note	2	48	1	500
500	note	2	43	1	500
500	note	2	48	1	500
500	note	2	43	1	500
500	note	2	50	1	500
500	note	2	43	1	500
500	note	2	50	1	500
500	note	2	43	1	500
500	note	2	48	1	1000
1000	note	2	46	1	1000
1000	note	2	45	1	1000
1000	note	2	44	1	1000
1000	note	2	48	1	500
500	note	2	43	1	500
500	note	2	50	1	500
500	note	2	43	1	500
500	note	2	48	1	1250

Table 6.4: Bass line for *saints1* (A-M)

reltime	**saints3**				
750	note	3	60	1	1000
0	note	3	64	1	1000
0	note	3	67	1	1000
2000	note	3	60	1	1000
0	note	3	64	1	1000
0	note	3	67	1	1000
2000	note	3	60	1	1000
0	note	3	64	1	1000
0	note	3	67	1	1000
1000	note	3	60	1	1000
0	note	3	64	1	1000
0	note	3	67	1	1000
1000	note	3	59	1	1000
0	note	3	62	1	1000
0	note	3	65	1	1000
0	note	3	67	1	1000
2000	note	3	60	1	1000
0	note	3	64	1	1000
0	note	3	67	1	1000
1000	note	3	60	1	1000
0	note	3	64	1	1000
0	note	3	67	1	1000
1000	note	3	60	1	1000
0	note	3	65	1	1000
0	note	3	69	1	1000
1000	note	3	60	1	1000
0	note	3	65	1	1000
0	note	3	68	1	1000
1000	note	3	60	1	1000
0	note	3	64	1	1000
0	note	3	67	1	1000
1000	note	3	59	1	1000
0	note	3	62	1	1000
0	note	3	65	1	1000
1000	note	3	60	1	1000
0	note	3	64	1	1000
0	note	3	67	1	1000

Table 6.5: Additional chords (A-M)

reltime	**saints4**				
1000	note	4	72	1	500
500	note	4	69	1	250
250	note	4	72	1	250
1250	note	4	72	1	500
500	note	4	69	1	250
250	note	4	72	1	250
1000	note	4	72	1	1000
1000	note	4	72	1	1000
1000	note	4	71	1	1000

Table 6.6: Additional clarinet voice (A-M)

abstime	**init**		
0	instrument	1	57
0	instrument	2	59
0	instrument	3	4
0	instrument	4	72

Table 6.7: *init* phrase containing instrument assignments (A-M)

The following simple macro code was used to assemble the phrases:

```
                clearall
saints1         define        A1:F33
saints2         define        H1:M26
saints3         define        A35:F72
saints4         define        H35:M44
init            define        O30:R34
all             copy          init
all             merge         saints1
all             merge         saints2
all             merge         saints3
all             merge         saints4
all             play
```

This resulted in a quite impressive MIDI file. This group also compiled a MIDI file (with the teacher's help), as they requested it.

6.4.7 Task 9

Unfortunately, no group chose to work on task 9, which one can attribute to the limited time available. Moreover, the composition of a song is something that is not too easy if one is not an expert in music theory. Therefore it is understandable that nobody chose task 9 in the 90 minutes that were available at the evaluation after the initial presentation.

6.4.8 Task 10

Task 10 was only completed once and only with the help of the teacher. This exercise was not so much meant as a task to solve (as it requires no macro code, just the pressing of a button), but rather was there to elicit interest among the students. One group did express interest (A-M), therefore the task completed with the teacher's help.

6.4.9 Further macro codes (task 11 on the diagram)

Some other macro codes were created that either included commands not requested on the task sheet or a mixture of several tasks. For example, I-M produced the following code:

```
                clearall
saints          define        B3:G36
init            define        B38:E41
saints1         copy          saints
saints2         copy          saints
saints2         reverse
saints3         copy          saints
saints2         rechannel     1        2
saints3         rechannel     1        3
saints2         timeshift     16000
```

```
saints3         timeshift     32000
all             copy          init
all             merge         saints1
all             merge         saints2
all             merge         saints3
all             transpose     2
all             play
```

This not only plays the song three times with alternating instruments, it also reverses one phrase and transposes all three songs up by one tone. A similar implementation is found with Mixed 2, who created a canon with helicopter sound effects (instruments in table 6.8):

```
                clearall
when            define        B3:G36
init            define        B38:E41
when1           copy          when
when2           copy          when
when3           copy          when
when2           rechannel     1         2
when3           rechannel     1         3
when2           timeshift     2000
when3           timeshift     4000
all             copy          init
all             merge         when1
all             merge         when2
all             merge         when3
all             midifile      myfile
                playfile      myfile
```

One may note that there is a slight error in this code: the last two lines should be replaced by `all play` in order to work out. This particular syntax error is interesting as to how Mixed 2 arrived at this syntax, a similar structure is nowhere to be found in the examples or the documentation.

abstime	init		
0	instrument	1	33
0	instrument	2	116
0	instrument	3	126

Table 6.8: *init* phrase containing sound effects (Mixed 2)

6.5 Reflection and Summary

It has been shown that a wide variety of solutions were presented at the evaluation of the project. In order to summarize the assignments handed in, it is illustrated in figure 6.4 which group chose which tasks.

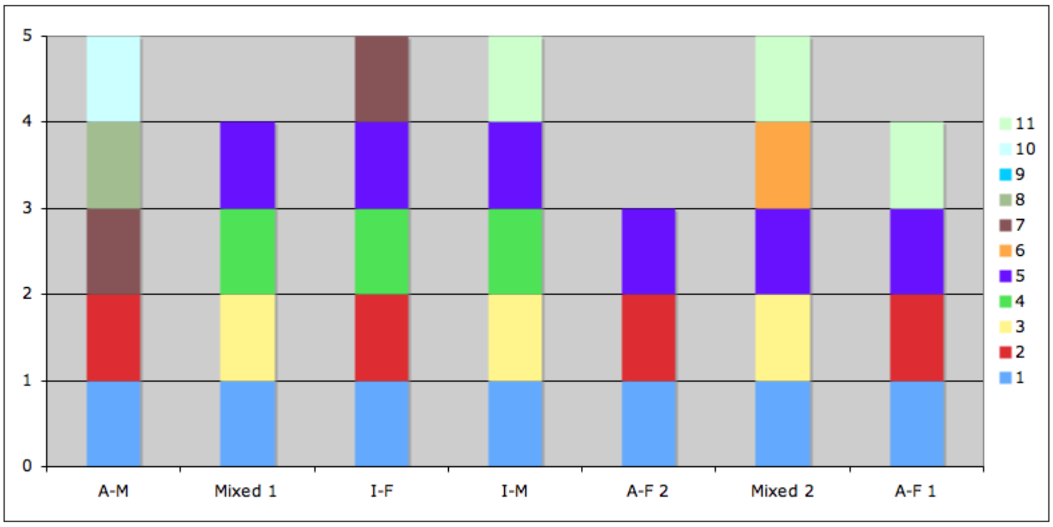


Figure 6.4: Distribution of tasks

It is interesting to note that there is no significant gender gap when looking at the completed exercises. Apart from group A-M, who performed very well with the more complex tasks, more or less all the groups managed to solve similar exercises. It is also interesting to note that working together not necessarily increases the productivity; I-M completed more exercises than A-F 2 or Mixed 1. Still, it was observable that students were more comfortable working in a group.

After the evaluation, several students pointed out that they found this way of doing some preliminary programming quite intuitive. Indeed, most macro code programs were syntactically correct and with regard to the relatively short time available quite well done. It is certainly not possible to go into the depths of macro language use (as already noted, phrase language was not touched upon due to time constraints), therefore many options are available to deepen music programming in MIDI. Still, this evaluation strongly indicates that the approach of using music in introductory programming promises good results. Moreover, as some students afterwards noted, it was also interesting and fun to program music, a fact that complies with Guzdial and Soloway's claims in 4.3.1.

On the whole, one can say that the hypothesis has been confirmed in the evaluation to a certain extent. Not all features have been tested, but it still seems clear that music indeed helped students very much to write simple programs. It can be expected that after a certain time working on this program, switching to a fully featured programming language can be anticipated to be a lot easier for beginners of programming.

7 Conclusion

This thesis has proposed the hypothesis that music and particularly musical structures may serve as a means to teaching programming, especially object-oriented programming. Several learning theories have been discussed and evaluated in terms of their possible contribution to teaching computer science. It has been clarified that computer science is a subject that requires a lot of active work from the students, thus reinforcing the strong constructionist aspect of a possible didactic model. Further is has been argued that in order to construct a viable and lasting propositional network that enables students to retain and independently increase knowledge, cognitivism offers several important contributions. Furthermore also age appropriateness plays a role, as defined in Piaget's theory. Behaviorism only contributes minor points to the model, still it has been noted that especially a comfortable learning atmosphere and positive reinforcement can significantly increase learning output.

Afterwards a few points have been made in order to clarify the term *didactics of computer science*, thus establishing a framework that allows to construct lessons and teaching sequences using problem based, project based or active learning, each of which incorporate to a significant amount many points mentioned in the discussion on learning theories.

Then it has been noted that teaching programming is a particularly difficult field of computer science to teach; algorithmic thinking and knowledge of data structures are a prerequisite in order to develop applications in any programming language. Therefore, it has been argued that there is a strong need to visualize data structures and the development of problem-solving algorithms to students in an easily accessible way, in the case of this thesis this is music. Several points have been made that musical structures in fact can serve as a model for abstract data structures and that MidiCSD, which significantly facilitates writing simple MIDI songs in MS Excel, can be used to teach these structures. For this purpose a workbook has been created that shall serve as a teaching aid to high school students or university students.

The assumptions developed in this thesis have been tested and confirmed in practice. Not the full extent of the possibilities available could be investigated, but the main argument of this thesis has been confirmed by a teaching sequence involving university students. The outcome showed that indeed virtually everybody had no problems constructing simple programs with MidiCSD and that the students were quite aware that they had managed to program a series of commands, much like in a common introductory programming class, except that here music was used instead of textual outputs in simple command line applications. Moreover, students were well aware of the object-oriented component that music features, an intuitive advantage that certainly has no counterpart in traditional programming classes.

Finally, one can say that music as a structure representing (musical) data is quite well suitable for teaching programming and all the neighboring fields it implies.

Notes

[1]See Hubwieser 2007: 3
[2]See http://paedpsych.jk.uni-linz.ac.at:4711/LEHRTEXTE/LERNEN/klassi.htm, accessed 11th Jan. 2010
[3]See Hubwieser 2007: 4
[4]See Hubwieser 2007: 5
[5]See Hubwieser 2007: 5
[6]See Hebb 1949 quoted in Hubwieser 2007: 5
[7]See http://www.nlm.nih.gov/medlineplus/parkinsonsdisease.html, accessed 11th Jan. 2009
[8]See Tolman and Honzik 1930, Köhler 1921, Koffka 1922, Wertheimer 1945, quoted in Hubwieser 2007: 5
[9]See Bruner 1957, quoted in Hubwieser 2007: 5
[10]See http://act-r.psy.cmu.edu/, accessed 11th Jan 2010
[11]See Anderson 1976: 146
[12]See Anderson 1976: 147
[13]See Anderson 1976: 148
[14]See Friedenberg and Silverman 2006: 230
[15]See Anderson 1976: 148, Friedenberg and Silverman 2006: 230
[16]See Anderson 1976: 149
[17]See Hubwieser 2007: 6
[18]See Wadsworth 1996: 13-14
[19]See Wadsworth 1996: 14
[20]See Wadsworth 1996: 17-19
[21]See Wadsworth 1996: 26
[22]See Wadsworth 1996: 27
[23]See Hubwieser 2007: 9
[24]See Fox 2001: 23
[25]See Matthews 1992, quoted in Olssen 1996: 276
[26]See Miller and Driver 1987, quoted in Olssen 1996: 276
[27]See Hacking 1990, quoted in Olssen 1996: 276
[28]See Confrey 1990, quoted in Olssen 1996: 276
[29]See von Glasersfeld 1984, Kelly 1955, quoted in Olsson 1996: 276
[30]See Fox 2001: 24
[31]See Piaget 1969, Lieben 1987, Adey and Shayer 1994, quoted in Fox 2001: 24
[32]See Wertsch 1985, Brown and Reeve 1987, Tharp and Gallimore 1988, quoted in Fox 2001: 24
[33]See Sharon 1994, quoted in Fox 2001: 24
[34]See von Glasersfeld 1996, quoted in Fox 2001: 24
[35]See Rogoff 1990, Mercer 1995, Fosnot 1996, quoted in Fox 2001: 24
[36]See Hubwieser 2007: 10
[37]See Terhart 1999: 631
[38]See Terhart 1999: 631-632
[39]See Hubwieser 2007: 10-11
[40]See http://www.bildungsberater-stmk.at/website/matura/lehrer.html, accessed 13th Feb 2010
[41]See Hubwieser 2007: 11
[42]See Hubwieser 2007: 67

[43]See Jank and Meyer 2007: 12 quoted in Humbert 2006: 32, [my translation]
[44]See Humbert 2006: 33
[45]See Schubert and Schwill 2004: 18
[46]See Humbert 2006: 39
[47]See Schubert and Schwill 2004: 293-294
[48]See Schubert and Schwill 2004: 294
[49]See Humbert 2006: 40, [my translation]
[50]See Edelmann 1986 quoted in Hubwieser 2007: 68, [my translation]
[51]See Edelmann 1986 quoted in Hubwieser 2007: 68, [my translation]
[52]See Friedrich 1995 quoted in Hubwieser 2007: 69
[53]See Schubert and Schwill 2004: 15
[54]See Hubwieser 2007: 69
[55]See Pólya 1967 quoted in Humbert 2006: 40
[56]See Roth quoted in Humbert 2006: 40
[57]See Bruner quoted in Humbert 2006: 43
[58]See Bruner quoted in Humbert 2006: 44
[59]See Schubert and Schwill 2004: 297
[60]See Xue and Zhu 2009: 654
[61]See Gudjons 1986 quoted in Schubert and Schwill 2004: 298
[62]See Gudjons 1986 quoted in Schubert and Schwill 2004: 298-299
[63]See Gudjons 1986 quoted in Schubert and Schwill 2004: 299
[64]See Baumann 1990: 221
[65]See Beza-Yates 1995: 1
[66]See Kaasbøll 1998: 196
[67]See Guzdial and Soloway 2002: 17
[68]See Guzdial and Soloway 2002: 18
[69]See `http://impromptu.moso.com.au/examples_2.0/01_bing.html`, accessed on 11th Jan. 2010
[70]For the MidiCSD package go to `http://www.sunsite.univie.ac.at/musicfun/midicsd`
[71]See Guzdial and Soloway 2002: 19
[72]See `http://www.squeak.org`, accessed 11th Jan. 2010
[73]See Guzdial and Soloway 2002: 20
[74]See Hamer 2004: 156
[75]See Hamer 2004: 157
[76]See Hamer 2004: 158

8 Bibliography

[1] *ACT-R: Theory and Architecture of Cognition.* `http://act-r.psy.cmu.edu/`, accessed 11th Jan 2010.

[2] Adey, P.; Shayer, M. *Really Raising Standards.* London 1994: Routledge.

[3] Anderson, John R. *Language, Memory and Thought.* Hillsdale NJ 1976: L. Erlbaum Associates.

[4] Baeza-Yates, Ricardo. Teaching Algorithms. In: *ACM SIGACT News*, (1995: 26/4), pp. 51-59.

[5] Baumann, Rüdeger. *Didaktik der Informatik.* Stuttgart 1990: Klett-Schulbuchverlag.

[6] Brown, A. L.; Reeve, R. A. Bandwidths of competence. In: Liben, L. *Development and Learning.* Hillsdale 1987: L. Erlbaum Associates.

[7] Bruner, Jerome S. *Contemporary Approaches to Cognition.* Cambridge MA 1957: Harvard University Press.

[8] Bruner, Jerome S. *Entwurf einer Unterrichtstheorie.* Düsseldorf 1974: Pädagogischer Verlag Schwann.

[9] Confrey, J. What constructivism implies for teaching. In: Davis, R. B.; Maher, C. A.; Noddings, N. (eds.). *Constructivist Views on the Teaching and Learning of Mathematics.* Reston 1990: National Council of Teachers of Mathematics.

[10] Edelmann, W. *Lernpsychologie - Eine Einführung* (2nd ed.). München/Weinheim 1986: Urban & Schwarzenberg.

[11] Fosnot, C. T. *Constructivism: theory, perspectives and practice.* New York 1996: Teachers College Press.

[12] Fox, Richard. Constructivism Examined. In: *Oxford Review of Education*, (2001: 21/1), pp. 23-35.

[13] Friedenberg, Jay; Silverman, Gordon. *Cognitive Science: an introduction to the study of mind.* Thousand Oaks 2006: Sage Publications.

[14] Friedrich S; Grundpositionen eines Schulfaches. In: *LOG IN*, (1995. 15/5, 6), pp. 30-34.

[15] Gudjons, Herbert. "Was ist Projektunterricht?" In: Bastian J.; Gudjons H. (eds.): *Das Projektbuch.* Berlin 1986: Bermann Helbig Verlag, pp. 14-27.

[16] Glasersfeld, Ernst von. *The Invented Reality.* New York 1984: Morton.

[17] Glasersfeld, Ernst von. Introduction: aspects of constructivism, chapter 1. In: Fosnot, C. T. *Constructivism: theory, perspectives and practice.* New York 1996: Teachers College Press.

[18] Guzdial, Mark; Soloway, Elliot. Teaching the Nintendo Generation to Program. In: *Communications of the ACM*, (2002: 45/4), pp. 17-21.

[19] Hacking, I. Natural kinds. In: Barret, R. B.; Gibson, R. F. (eds.). *Perspectives on Quine.* Cambridge 1990: Basil Blackwell.

[20] Hamer, John. An approach to teaching design patterns using musical composition. In: *Annual Joint Conference Integrating Technology into Computer Science Education: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, (2004), pp. 156-160.

[21] Hebb, D. O. *The Organization of Behaviour.* New York 1949: Wiley.

[22] Hubwieser, Peter. *Didaktik der Informatik.* Heidelberg 2007: Springer Verlag.

[23] Humbert, Ludger. *Didaktik der Informatik.* Wiesbaden 2006: Teubner Verlag.

[24] *Impromptu.* `http://impromptu.moso.com.au/`, accessed 11th Jan 2010.

[25] Jank, Werner; Meyer, Hilbert. *Didaktische Modelle* (5th ed.). Berlin 2002: Cornelsen Scriptor.

[26] Kaasbøll, Jens J. Exploring didactic models for programming. In: *Norwegian Informatics Conference.* Trondheim 1998: Tapir, pp. 195-203.

[27] Kelly, G. A. *The Psychology of Personal Constructs.* New York 1955: Norton.

[28] *Klassische Konditionierung nach Pawlow.* `http://paedpsych.jk.uni-linz.ac.at:4711/LEHRTEXTE/LERNEN/klassi.htm`, accessed 11th Jan 2010.

[29] Köhler, W. *Intelligenzprüfungen an Menschenaffen.* Berlin 1921: Springer.

[30] Koffka, K. "An Introduction to Gestalt Theory." In: *Psychological Bulletin*, (1922: 19), pp. 531-585.

[31] *Lehrerausbildung.* `http://www.bildungsberater-stmk.at/website/matura/lehrer.html`, accessed 14th Feb 2010.

[32] Liben, L. *Development and Learning: conflict of congruence?.* Hillsdale NJ 1987: L. Erlbaum Associates.

[33] Matthews, M. R. Constructivism and empiricism: an incomplete divorce. In: *Research in Science Education*, (1992: 22), pp. 299-307.

[34] Mercer, N. *The Guided Instruction of Knowledge.* Clevedon 1995: Multilingual Matters.

[35] *MidiCSD, support for Midi sound and Music from within Excel*, `http://sunsite.univie.ac.at/musicfun/MidiCSD/`, accessed 11th Jan 2010.

[36] Miller, R.; Driver, R. Beyond processes. In: *Studies in Science Education*, (1987: 14), pp. 33-62.

[37] Ming Xue, Changjun Zhu, The Application of 'Project-oriented' Teaching Mode in Computer Course of Advanced Vocational Education, *Computer-Aided Software Engineering, International Workshop on*, pp. 652-654, 2009 IITA International Conference on Control, Automation and Systems Engineering (case 2009), 2009.

[38] Olssen, Mark. Constructivism and Its Failings: Anti-Realism and Individualism. In: *British Journal of Educational Studies*, (1996: 44/3), pp. 275-295.

[39] *Parkinson's disease: Medline Plus.* `http://www.nlm.nih.gov/medlineplus/parkinsonsdisease.html`, accessed 11th Jan 2010.

[40] Piaget, Jean. *Science of Education and the Psychology of the Child.* Harlow 1969: Longman.

[41] Pólya, George. *Vom Lösen mathematischer Aufgaben* (Vol. 1 & 2). Basel 1966/1967: Birkhäuser

[42] Rogoff, B. *Apprenticeship in Thinking: cognitive development in a social context.* New York 1990: Oxford University Press.

[43] Schubert, Sigrid; Schwill, Andreas. *Didaktik der Informatik.* München 2004: Spektrum Akademischer Verlag

[44] Sharron, H. *Changing Children's Minds.* Birmingham 1994: The Sharron Press.

[45] *Squeak Smalltalk.* `http://www.squeak.org/`, accessed 11th Jan 2010.

[46] Terhart, Ewald. Konstruktivismus und Unterricht. In: *Zeitschrift für Pädagogik*, (1999: 45/5), pp. 629-647.

[47] Tharp, R.; Gallimore, R. *Rousing Minds to Life: teaching, learning and schooling in social context.* New York: Cambridge University Press.

[48] Tolman E. C.; Honzik C. H. Insight in Rats. In: *University of California Publications in Psychology*, (1930: 4), pp. 215-232.

[49] Watson, John B. *Behaviorismus.* Eschborn bei Frankfurt am Main 1997: Klotz.

[50] Wertheimer, M. *Productive Thinking.* New York 1945: Harper & Row.

[51] Wertsch, J. V. *Vygotsky and the Social Formation of Mind.* Cambridge MA 1985: Harvard University Press.

[52] Wadsworth, Barry J. *Piaget's Theory of Cognitive and Affective Development: Foundations of Constructivism.* New York 1996: Longman.

9 Appendix

9.1 Abstract - English

This thesis presents an unconventional approach to didactics of computer science - music and musical structures applied in programming instruction. After looking at several learning theories with regard to their relevance for teaching computer science, a few methods vital for good CS teaching are presented. The thesis' central hypothesis is that essential concepts for programming such as algorithms and data structures can be taught by using music as a starting point and musical structures as models for abstract data structures. For this purpose a workbook for students is created and a teaching sequence is proposed that implements the hypothesis for high school or university teaching courses. Finally an evaluation carried out with a sample group of university students tests the hypothesis in practice.

9.2 Abstract - German

Diese Diplomarbeit präsentiert einen unkonventionellen Zugang zur Didaktik der Informatik - Musik und musikalische Strukturen und deren Anwendung im Programmierunterricht. Nach der kritischen Betrachtung einiger Lerntheorien und deren Relevanz im Informatikunterricht werden diverse Methoden präsentiert, die für den Informatikunterricht von besonderer Bedeutung sind. Die Diplomarbeit beruht auf der Hypothese, dass sich Datenstrukturen und Algorithmen, essentielle Konzepte der Programmierung, durch Musik bzw. musikalische Strukturen darstellen lassen. Zu diesem Zweck wird auch ein Unterrichtskurs sowie ein begleitendes Arbeitsheft erstellt, sowohl für höhere Schulen als auch für Universitätskurse. Weiters wird die Hypothese in einem praktischen Unterrichtsprojekt mit Studenten auf die Probe gestellt. Hierbei stellt sich heraus, dass tatsächlich musikalische Strukturen hervorragend geeignet sind, fundamentale Konzepte des Programmierens zu erklären.

9.3 Workbook

An Introduction to Programming

with Music and MidiCSD

by Rainer Dangl

Introduction

This workbook shall offer an introduction to the world of programming. The approach used here is quite a bit different from other commonly used methods: before we actually program something in a particular programming language, we will look at the theoretical concepts behind programming first: algorithms and data structures, which are absolutely necessary to be familiar with before starting to program. Algorithms and data structure may be terms that you might not know at all, but this is going to change soon! This workbook is most efficient when you work in groups, as the final project you will be working on is also a group project. Still, it is also possible to work through the exercises on your own. Your teacher will exactly tell you how to proceed. As already noted, at the end of this workbook stands a project in the course of which you will compose a song ... but we will come to this point later on. First of all we need to make sure that you have all the programs installed that are necessary for this course.

Maybe the program is already installed on the computer at school or at home. If not you find MidiCSD, an Add-on for MS Excel here: `http://www.sunsite.univie.ac.at/musicfun/MidiCSD/`. Please note that you will have to have MS Excel installed. Furthermore, the Add-on only works on Windows-PCs, not on Mac OS versions of MS Office.

While working through this workbook, please keep the following points in mind:

- It is very important that you work through this book in the intended order. Your teacher will give you some guidance anyway, but when you work on your own, please do not skip over some exercises. Of course you can revise former chapter is you feel the need to.
- This project is neither a test nor a revision. Therefore, you have enough

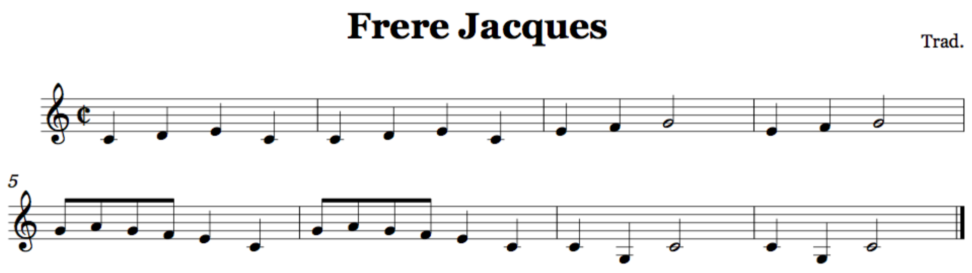
time! There is absolutely no need to rush through the exercises. It is very important to understand all the sections of this workbook, so take your time. As this is a project, there is of course a certain timeline, but there will be plenty for a successful completion.

- This workbook is intended as a group project. You will see that working in a group especially in computer science is a good way to work and learn very quickly. Therefore please work concentratedly and try to solve as much as you can without the help of the teacher. Of course you can, if you are really stuck with a problem and you cannot figure it out, ask the teacher for help.
- Please do not use the internet while you work through this book. At the end, when you need to compose songs and you need to find scores, you can of course use the internet to find songs. But especially during the first chapters avoid looking for answers on the internet. As this is not a test, you do not get points deducted if you give an incorrect answer. The aim is that if you don't know the answer, you should keep looking for the solution until it is clear to you. As you work in a group, this really should not be a problem.

If you are clear about these points we can start!

What does music consist of?

This is the first question we may ask ourselves. You will see that there are many connections between the way a piece of music is structured and a data structure on the computer! After all, sheet music is also only a way to structure musical data. Let us look at the following piece:

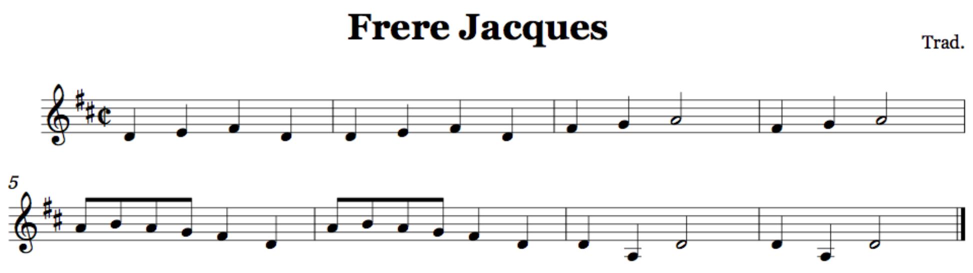


This is a very simple yet famous song. Surely you already know several terms from music theory, therefore let us analyze this song. We can firstly say that every songs consists of notes. Notes have certain properties, for example length and pitch. Once these two properties are given, we can identify a tone without doubt. We can now ask ourselves what other structures we can identify when thinking about a song, apart from notes. Discuss within your group, find suggestions and write them down, you will find the solution on the next page.

Which solutions did you find? For once, we can assume notes to be the smallest unit of a musical structure. Accordingly, the next bigger entity would be the measure, which consists of several notes, depending on the time signature and the note length. In this case we have a 4/4 measure, which means that one measure contains four quarter note or eight eighth notes. Several measures then constitute a musical phrase. This phrase might be identical to the piece of music as such (as in our example case), but it might also be that a piece of music contains several phrases (a symphony for example usually contains 4 movements, so we could classify a movement as a phrase).

Summing up, we can say that notes are the smallest units of a piece of music which can be summarized in measures which in turn constitute a phrase of the whole piece.

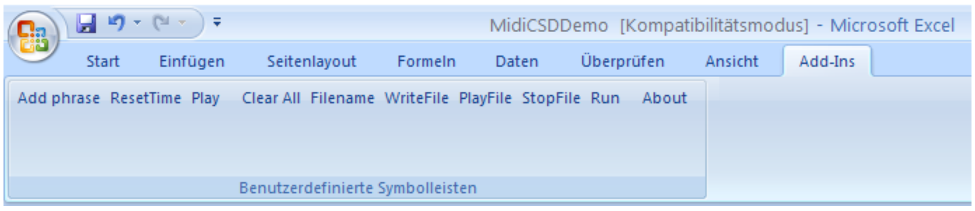
With notes and phrases one can of course do several things. You might be familiar with the process of transposing music, which changes the key signature of the piece. For example, if we would want our example song to be a tone higher we need to transpose it from c-major to d-major. This looks like this:



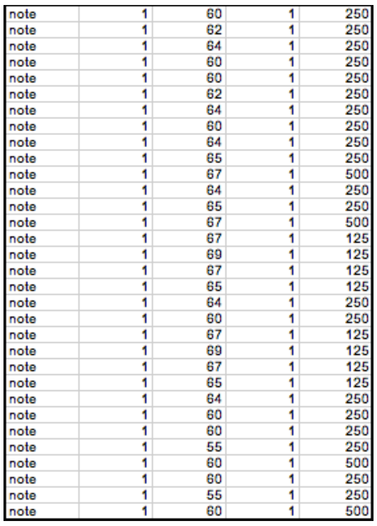
We could modify the piece further; we could insert a repetition bar at the end, which would mean that the piece will be played twice before it is finished. These are only two examples of operations that can be performed on musical phrases.

Data structure in MidiCSD

We have now clarified that there are a number of possibilities of how to change the structure of a piece of music. We also know how to describe the structure of a piece of music. We shall now look at the implementation of this structure on the computer. Please open Excel and add the MidiCSD Add-on, if necessary. You then should see a new toolbar:



This is how the toolbar looks in Excel 2007. If you still have Excel 2003 installed you might see a different picture, but you should in any case see the buttons *Add Phrase*, *ResetTime*, and so forth. Additionally, you will see the following table on the first sheet:

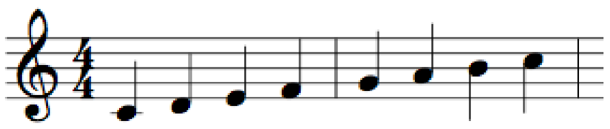


note	1	60	1	250
note	1	62	1	250
note	1	64	1	250
note	1	60	1	250
note	1	60	1	250
note	1	62	1	250
note	1	64	1	250
note	1	60	1	250
note	1	64	1	250
note	1	65	1	250
note	1	67	1	500
note	1	64	1	250
note	1	65	1	250
note	1	67	1	500
note	1	67	1	125
note	1	69	1	125
note	1	67	1	125
note	1	65	1	125
note	1	64	1	250
note	1	60	1	250
note	1	67	1	125
note	1	69	1	125
note	1	67	1	125
note	1	65	1	125
note	1	64	1	250
note	1	60	1	250
note	1	60	1	250
note	1	55	1	250
note	1	60	1	500
note	1	60	1	250
note	1	55	1	250
note	1	60	1	500

The question now of course arises what this table should represent. Right next to the table you find a few instructions. Carry them out and observe what happens!

You will hear a song, which we already know in its sheet music form. It seems that here we have the song in described by means of a table. Have a close look at the columns. Admittedly, the columns containing only the number 1 are not really exciting. But that does not mean that they are not important! We will come back to them in a moment. First try to find out which information is encoded in the third and fifth column. Only read on when you have found out!

Surely you have recognized what information is to be found in the two columns: pitch and duration. What else did you note? For example, you may have observed that each line describes a note (which is not all that hard to guess, the word *note* stands at the beginning of each line). To complete the picture: the other two columns that contain the number 1 are the channel and the volume. This piece of music uses only one channel (there is only one voice). Of course it would be possible to use several channels to play several voices at once. The volume is always the same, the standard volume 1 is therefore used. Try to change some of the values and see what happens! What can you conclude from the pitch numbering? Try to assign the correct numbers to the scale below:



The MIDI Format

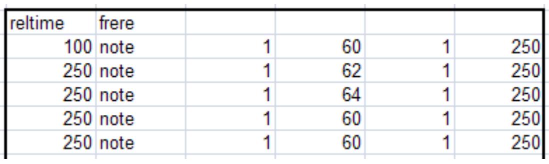
The correct solution to the scale on the previous page is: 60, 62, 64, 65, 67, 69, 71 and 72. Why? You may have found out yourself: of course the semitones also need to have a number, therefore in the scale above the numbering skips a number several times. The semitones in-between therefore have the following numbers: 61, 63, 66, 68 and 70. The lowest possible note has the value 0 and the highest possible note has the value 127. When looking at a piano keyboard, you will see that it usually has 88 keys. Therefore we here have an even greater range available! The duration on the other hand is given in milliseconds. Together with the channel and the volume, these four parameters represent a tone that the computer can process. This format is called MIDI (=musical instrument digital interface). What is particularly special with this format? Usually music is recorded and transmitted in waves. The term *sound wave* might ring a bell. In MIDI it is entirely different: the tone is defined by integers, exactly as in the table above. This results in the fact that the computer does not store the actual tone, it stores a description of the tone. The interpretation of this tone then can be arranged on the computer (which instrument to use, which reverb or other sound effects to apply).

We therefore keep in mind: the MIDI format needs 4 parameters in order to produce a tone: channel, pitch, volume and duration.

Now have a look at the next spreadsheet. At the beginning of the table you see that a new column has appeared - *reltime* (relative time). Yet when playing the song there is no difference. Which purpose could *reltime* then have? Try to change values and see what happens. Discuss in your group.

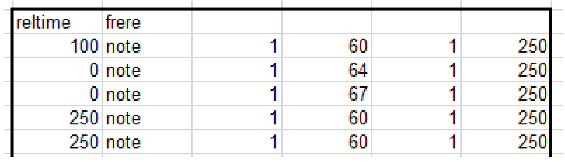
Relative Time

By means of relative time a note refers to its predecessor. Have a look at the following code excerpt:



reltime	frere				
100	note	1	60	1	250
250	note	1	62	1	250
250	note	1	64	1	250
250	note	1	60	1	250
250	note	1	60	1	250

The first note (the c) is 250 ms long. The the d follows with exactly the same duration. Without the relative time, the d would always start after the c ends. With reltime however, you can specify, when the d should start with regard to its predecessor, the c. Now, the d starts 250 ms after the c starts, which allows the c to finish in this exact time. If you would set the reltime value of d to 0, the c and the d are played simultaneously! Therefore the following code



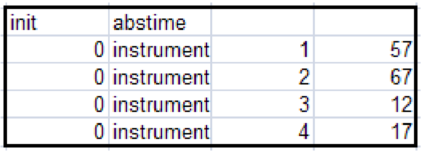
reltime	frere				
100	note	1	60	1	250
0	note	1	64	1	250
0	note	1	67	1	250
250	note	1	60	1	250
250	note	1	60	1	250

will result in a triad (c, e and g, the pitches also have been modified). The second and the third note now do not wait for the first note before to finish, they start at the same time.

Now you might argue that previously we stated that channels can be used to play several voices at the same time. This is correct, but note that a chord is not the same as voices. A voice is an entire phrase of its own, while a chord is within a phrase. We will work with phrases in a moment, for now we keep in mind that the relative time is a convenient means to play several notes at once.

Phrases and Objects

The table we have worked with so far is, according to the above definition, a phrase. When we have such a phrase, a number of things can be done with it. You might remember, we discussed this during the first pages, transposing, copying and so on. Look at the third spreadsheet (*FrereStampedParam*). There you see, next to our song, a new box:



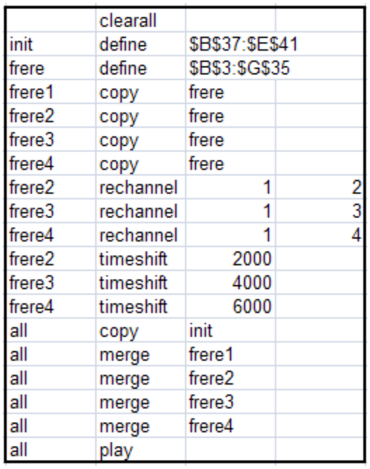
init	abstime		
0	instrument	1	57
0	instrument	2	67
0	instrument	3	12
0	instrument	4	17

Again, look at the instructions on the sheet and execute them! Now you see that there is a canon. What happened? First of all, we copied the phrase four times and assigned each copy a new channel. Therefore the four phrases end up in channels 1, 2, 3 and 4. Furthermore, you may have noticed that for the channels 2, 3 and 4, the first note starts with an increasing delay. When all phrases are then added to the memory and played at the same time, the timeshift results in a canon. The new box pictured above finally assigns an instrument to each channel. The numbers in the last column indicate which instrument to choose. This whole process results in a canon with different instruments.

Let us repeat this: a musical phrase can be treated as an object. As such it has certain properties, like for example the instrument that is assigned to it. It is then possible to do various things with this object. As we have just seen, we can copy such an object, or modify it (by implementing the timeshift) or transpose it. Therefore we see that an object is a very flexible unit.

Assembler - the Codeworkshop

We have now come already quite far in terms of understanding a data structure. If you think back about objects, phrases, notes and how to describe them, the properties they have and how notes are organized in a musical structure, you already know quite much about data structures on the computer. Now is also the time to introduce a new term, the *algorithm*. This sounds more complex than it is. Essentially, an algorithm describes the way from a given problem to a particular solution. The first half of the way is already behind us - we wanted to play music on the computer. We made up our mind on how to structure the data. The other half is the implementation on the computer. One could also say that an algorithm is as set of rules that leads from the problem to a solution. That is where we finally arrive at programming. Have a look at the fourth sheet (*FrereFullAssem*). Again, you will find the song, but now, right next to it, there is this box:



	clearall		
init	define	$B$37:$E$41	
frere	define	$B$3:$G$35	
frere1	copy	frere	
frere2	copy	frere	
frere3	copy	frere	
frere4	copy	frere	
frere2	rechannel	1	2
frere3	rechannel	1	3
frere4	rechannel	1	4
frere2	timeshift	2000	
frere3	timeshift	4000	
frere4	timeshift	6000	
all	copy	init	
all	merge	frere1	
all	merge	frere2	
all	merge	frere3	
all	merge	frere4	
all	play		

If you execute the instructions, you again will hear the canon. This box represents an algorithm (i. e. a program) to make a canon out of the musical phrase. Discuss in the group what happens to the phrase when these commands are executed.

Assembler - the Codeworkshop, Part 2

This is not an easy exercise, let us go through the program step by step.

- the first and the second line define the phrases and give them names. It seems that there is one phrase named *frere* and another one named *init*. *Init* obviously has the same purpose like in the example from two pages ago - it assigns instruments to channels. *Frere* is of course the song.
- the phrase *frere* is then copied four times. The new phrases receive the names *frere1*, *frere2*, *frere3* and *frere4*.
- all of these four copied phrases are still in channel 1. We need to change this for a canon. Therefore the latter three are moved to other channels - channel 2, 3, and 4.
- the latter three phrases also need to begin later, otherwise we do not get a canon. Therefore a timeshift of 2000, 4000 and 6000 ms is performed.
- at the end, also the phrase *init* is copied and named *all*. To this phrase we add the four *frere*-phrases and the final product is played.

You might have noticed the structure of the commands: on the left there is always the name of the phrase, then comes the command and thirdly comes an input, if the command requires it to work. For example:

```
frere2     -     rechannel     -     1     -     2
```

puts *frere2* into a new channel, and in order for the command *rechannel* to work it needs to know in which channel to find the phrase *frere2* and in which channel it should put it.

This way, we have rendered a piece of music, using a data structure!

Have a look at the other spreadsheets. Here you can see other interesting sound effects implemented in the same way as the previous examples.

Compiling and Interpreting Code

There are two big families of programming languages: compiler-based languages and interpreter-based languages. You already know one of them: the program you just ran is was an interpreter program. What does this mean?

Interpreter languages read the source code (i. e. the commands that you see in the box) while executing the program and translate it into machine code that the computer can understand. You can compare this approach to a musician who can sight-read. While playing on the piano, he reads the score and turns it into music on the piano. The computer does the exact same thing: it starts the program without knowing what commands if will have to execute. It reads them, translates it to machine code and executes them instantly.

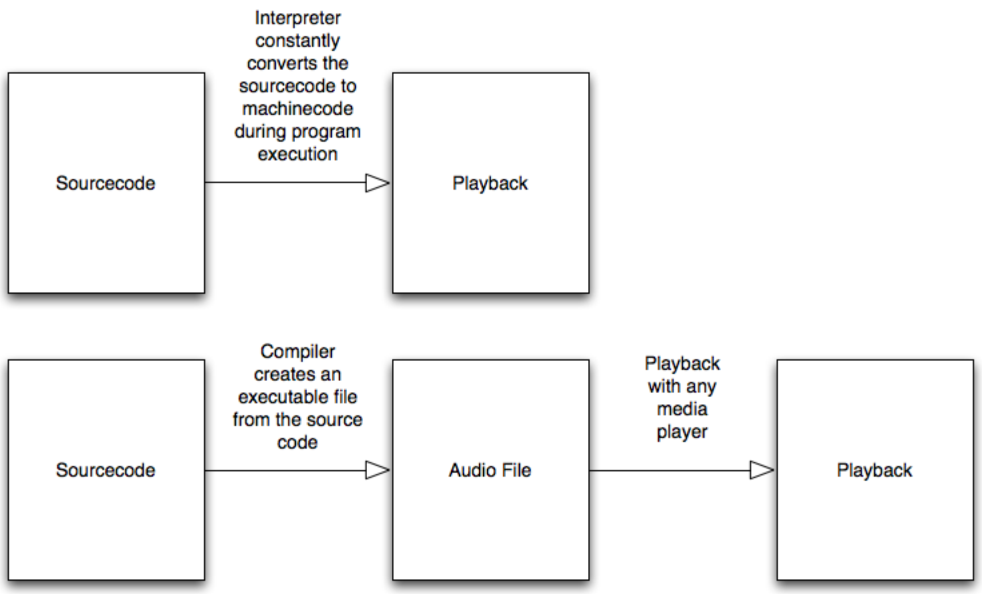
The second major family of programming languages are compiler languages. They have another approach. You can see what they are doing different by going back to the first sheet. Place the cursor in the phrase as usual and then press *Clear All*, *Add Phrase* and then *WriteFile*. Have a look in the MidiCSDxls folder. What happened? Discuss in your group what the different approach of compiler based languages could be. If you have some theories, read on on the next page.

Compiling and Interpreting Code, Part 2

You may have discovered that in the MidiCSD folder you find an audio file. This file was generated and can be opened with any media player.

Compiler based languages therefore operate as follows: before you can start the program, the code has again to be brought into machine-readable form, i. e. translated into machine code. The compiler of a programming language (this is a special program) therefore reads the code and converts it into a file that you can then execute. If we again draw a parallel to the musician on the piano, he would now first study the song until he knows it by heart. Then he goes to the piano and plays it.

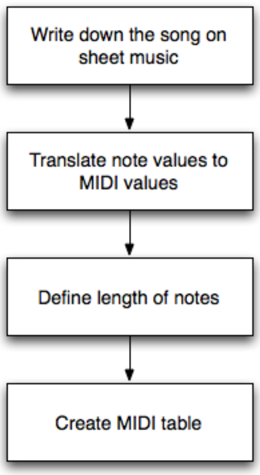
The following image shows the difference between the two approaches:


Sourcecode
Interpreter constantly converts the sourcecode to machinecode during program execution
Playback
Sourcecode
Compiler creates an executable file from the source code
Audio File
Playback with any media player
Playback

Discuss advantages and disadvantages the two approaches could have. Which is the ”better” one?

Programming in MidiCSD

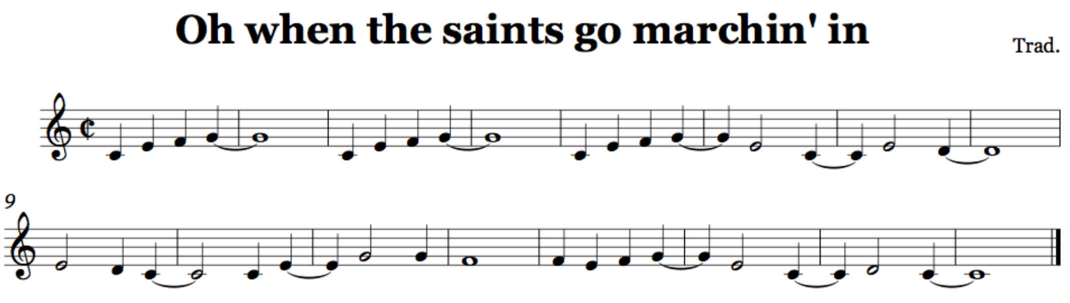
Now we want to program something ourselves. For this purpose you need a song. You can either compose one, or choose an existing song, but be aware that it should be a very simple tune, like the song *Frere Jacques*, as an implementation of a more complex song might be a bit too difficult for the beginning. You will get the chance to compose more complex songs soon, so for now we rather choose a simple song. The process of how you arrive at a MIDI file looks like this:


Write down the song on sheet music
Translate note values to MIDI values
Define length of notes
Create MIDI table

This means that you take your composed song (or existing song) and translate it into the MIDI format. Thus you first find the pitch values for the notes and write them down. Keep in mind, the middle-c has the value 60. This is your starting point. After that, you need to choose the duration. This means that you need to define how long a quarter note shall be in your piece. If it is 250 ms, it is fairly fast. Consequently the half note the is 500 ms long and the eighth note is 125 ms. If you double all those values, you get a song twice as slow. This depends on the song.

Next you need to define the channel and the volume. If it is a simple song with just one voice you can set these two columns to 1, as in the example in MidiCSD. This is all the information you need! With it you can create the table and enter in in Excel. If you then, as usual, press *Clear All*, *Add Phrase* and *Play* (while positioning the cursor somewhere in the table) you can listen to your song! Furthermore, with *WriteFile* you can also compile your song and get a file that you for example can upload to your cellphone as a ringtone!

Try this process with a song in your group! Does it work? If not check the following pages, we shall go through the whole process with an example song. This is the score we start out with:

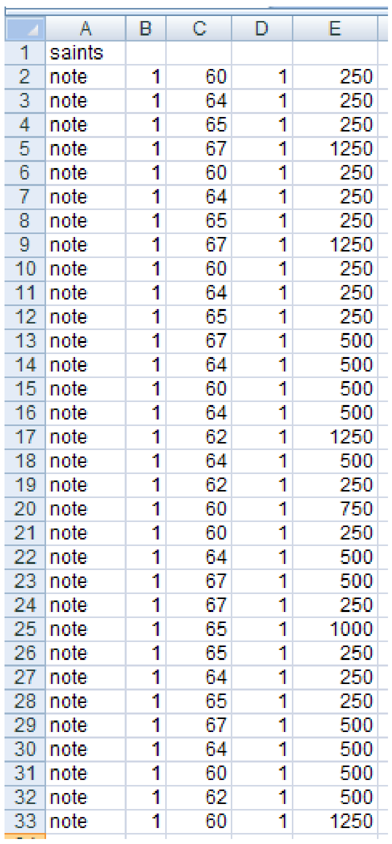


The first step is to translate the note values into MIDI values. It is quite simple, the first two measure translate as follows:

Note	Pitch (MIDI)	Duration (MIDI)
c (quarter)	60	250
e (quarter)	64	250
f (quarter)	65	250
g (quarter + whole note)	67	1250

All the other pitches and durations should not be a problem. For this example, the final MIDI table looks like you see it on the next page.

That was not too difficult, was it? Now comes the hard part: you now have a simple tune implemented in MidiCSD. What if you want to turn this into a

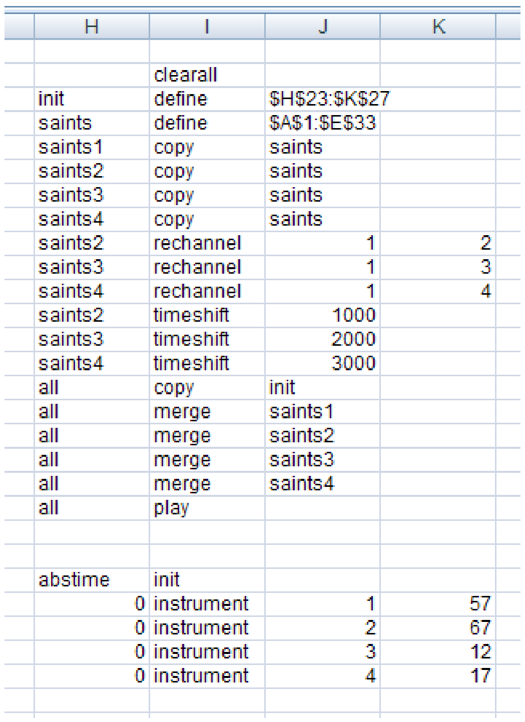


	A	B	C	D	E
1	saints				
2	note	1	60	1	250
3	note	1	64	1	250
4	note	1	65	1	250
5	note	1	67	1	1250
6	note	1	60	1	250
7	note	1	64	1	250
8	note	1	65	1	250
9	note	1	67	1	1250
10	note	1	60	1	250
11	note	1	64	1	250
12	note	1	65	1	250
13	note	1	67	1	500
14	note	1	64	1	500
15	note	1	60	1	500
16	note	1	64	1	500
17	note	1	62	1	1250
18	note	1	64	1	500
19	note	1	62	1	250
20	note	1	60	1	750
21	note	1	60	1	250
22	note	1	64	1	500
23	note	1	67	1	500
24	note	1	67	1	250
25	note	1	65	1	1000
26	note	1	65	1	250
27	note	1	64	1	250
28	note	1	65	1	250
29	note	1	67	1	500
30	note	1	64	1	500
31	note	1	60	1	500
32	note	1	62	1	500
33	note	1	60	1	1250

canon? Take a look at the sheet *FrereFullAssem* again, this should help you. The only problem of which you should be aware from the beginning is that, like in the image above, the title of the phrase *has to be* at the top. Otherwise you will get an error message.

As you might have guessed, you need the box with the commands that operate on the phrase (i. e. the assembler, as the name already says). Try to solve this problem in your group. You can then check on the next page how the assembler has to look like for this example here.

Generally, this assembler is almost the same as with *FrereFullAssem*. This is not much of a surprise, as the songs are structurally quite similar and the same solution was desired for both examples. You might have noticed that the *init* table



H	I	J	K
	clearall		
init	define	$H$23:$K$27	
saints	define	$A$1:$E$33	
saints1	copy	saints	
saints2	copy	saints	
saints3	copy	saints	
saints4	copy	saints	
saints2	rechannel	1	2
saints3	rechannel	1	3
saints4	rechannel	1	4
saints2	timeshift	1000	
saints3	timeshift	2000	
saints4	timeshift	3000	
all	copy	init	
all	merge	saints1	
all	merge	saints2	
all	merge	saints3	
all	merge	saints4	
all	play		
abstime	init		
0	instrument	1	57
0	instrument	2	67
0	instrument	3	12
0	instrument	4	17

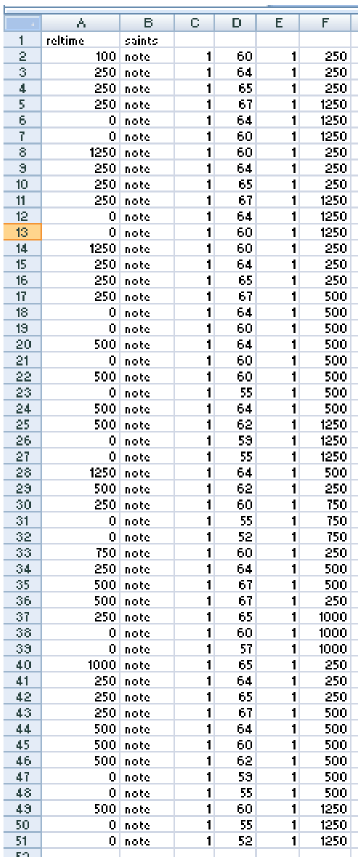
is necessary which, as we know, assigns an instrument to a channel. What has changed are of course the phrase names and the ranges. The timeshifts need to be adjusted to fit the song - it depends where the canon fits best. In this example, 1, 2 and 3 seconds are a good choice.

You might recall that we tried to produce a canon with another method as well: we did all the commands we just performed automatically in the assembler ourselves (*FrereStampedParam*). Try this as well in your group!

Special options in MidiCSD

It is, as already noted above, possible to put chords into the piece. Try if you can do that! For this purpose you need to remember what we discussed regarding the relative time - we used it to determine when a tone starts. You might want to look at the section about reltime again before trying to modify your phrase.

One possible way to do this for our example here would look as follows:



	A	B	C	D	E	F
1	reltime	saints				
2	100	note	1	60	1	250
3	250	note	1	64	1	250
4	250	note	1	65	1	250
5	250	note	1	67	1	1250
6	0	note	1	64	1	1250
7	0	note	1	60	1	1250
8	1250	note	1	60	1	250
9	250	note	1	64	1	250
10	250	note	1	65	1	250
11	250	note	1	67	1	1250
12	0	note	1	64	1	1250
13	0	note	1	60	1	1250
14	1250	note	1	60	1	250
15	250	note	1	64	1	250
16	250	note	1	65	1	250
17	250	note	1	67	1	500
18	0	note	1	64	1	500
19	0	note	1	60	1	500
20	500	note	1	64	1	500
21	0	note	1	60	1	500
22	500	note	1	60	1	500
23	0	note	1	55	1	500
24	500	note	1	64	1	500
25	500	note	1	62	1	1250
26	0	note	1	59	1	1250
27	0	note	1	55	1	1250
28	1250	note	1	64	1	500
29	500	note	1	62	1	250
30	250	note	1	60	1	750
31	0	note	1	55	1	750
32	0	note	1	52	1	750
33	750	note	1	60	1	250
34	250	note	1	64	1	500
35	500	note	1	67	1	500
36	500	note	1	67	1	250
37	250	note	1	65	1	1000
38	0	note	1	60	1	1000
39	0	note	1	57	1	1000
40	1000	note	1	65	1	250
41	250	note	1	64	1	250
42	250	note	1	65	1	250
43	250	note	1	67	1	500
44	500	note	1	64	1	500
45	500	note	1	60	1	500
46	500	note	1	62	1	500
47	0	note	1	59	1	500
48	0	note	1	55	1	500
49	500	note	1	60	1	1250
50	0	note	1	55	1	1250
51	0	note	1	52	1	1250

With this, you can write very impressive MIDI music pieces! If you then add the phrase with *Clear All* and *Add Phrase* to the memory and compile it with *WriteFile*, you have your self-made MIDI file!

Project: Composition Contest (school)

You now know enough about MidiCSD to start a project on your own. In your group try to create a particularly impressive song. You can first write it down on sheet music, if you feel that you can do it directly in MidiCSD, go for it!

The target of this contest is to produce a song on the computer, as complex and creative as possible. Complexity can for example be measured by the number of voices used, by the number of instruments or the polyphony. In theory it is possible to implement a small orchestral piece in MidiCSD. Open the file MidiCSDDocs for reference. You find all the commands you can use in the Assembler (on the sheet *Macro Language*), or the phrase language syntax (we used only channel/pitch/volume/duration), but there is even more! You will also find the list of all instruments available, along with the numbers assigned to them.

The best compositions might (if they are well made) make it onto the school website as background music. Therefore have fun programming your contribution!

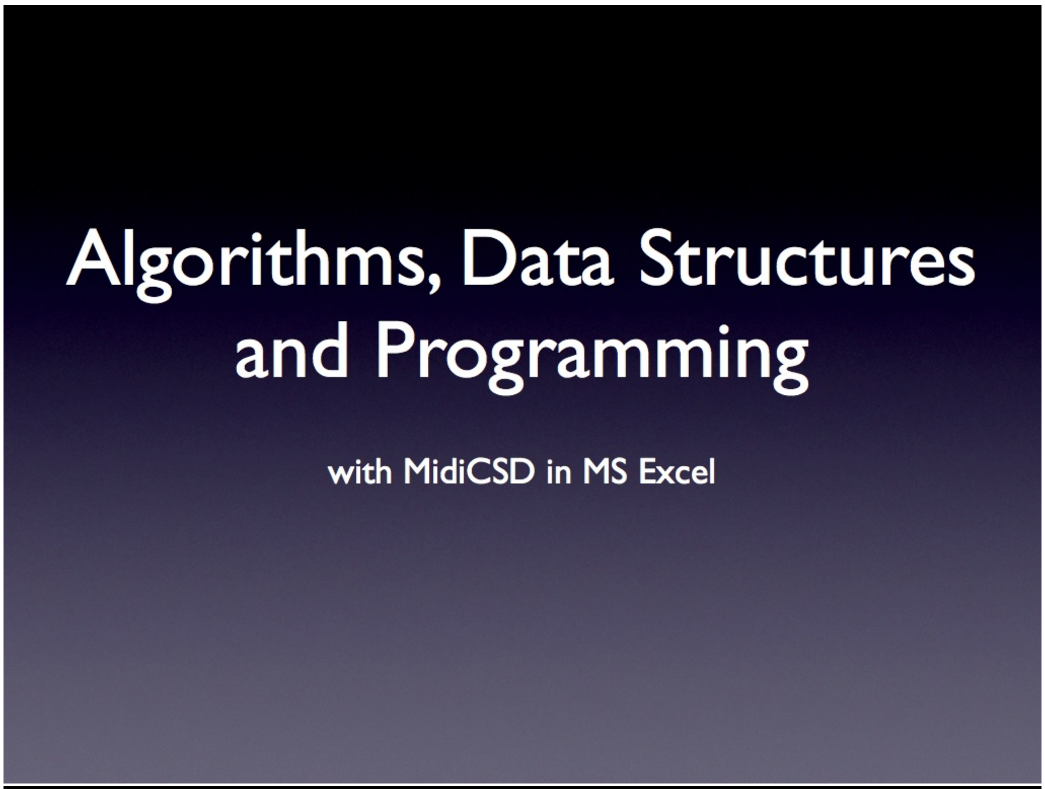
Assessment: Composition of a song (university)

The final project for this course is the implementation of a song in MidiCSD. For this purpose you need to compose a song of your own. This means that no implementation of an existing song is allowed - it needs to be solely your creation. This allows you to freely construct a song you like, yet a certain level of complexity has to be maintained. Consider the following points:

- the song should at least have a two voices (i. e. a piano piece). Of course you can implement as many voices as you like (e. g. four for a choir, or even more for an orchestra), but there need to be at least two.
- make use of the possibilities *reltime* offers and use polyphony. That means you should incorporate chords. It cannot be exactly defined how many, as this depends on your piece, but a few instances should be there.
- use the macro language to automatically assemble your piece.
- compile your song, so you have an audio file.

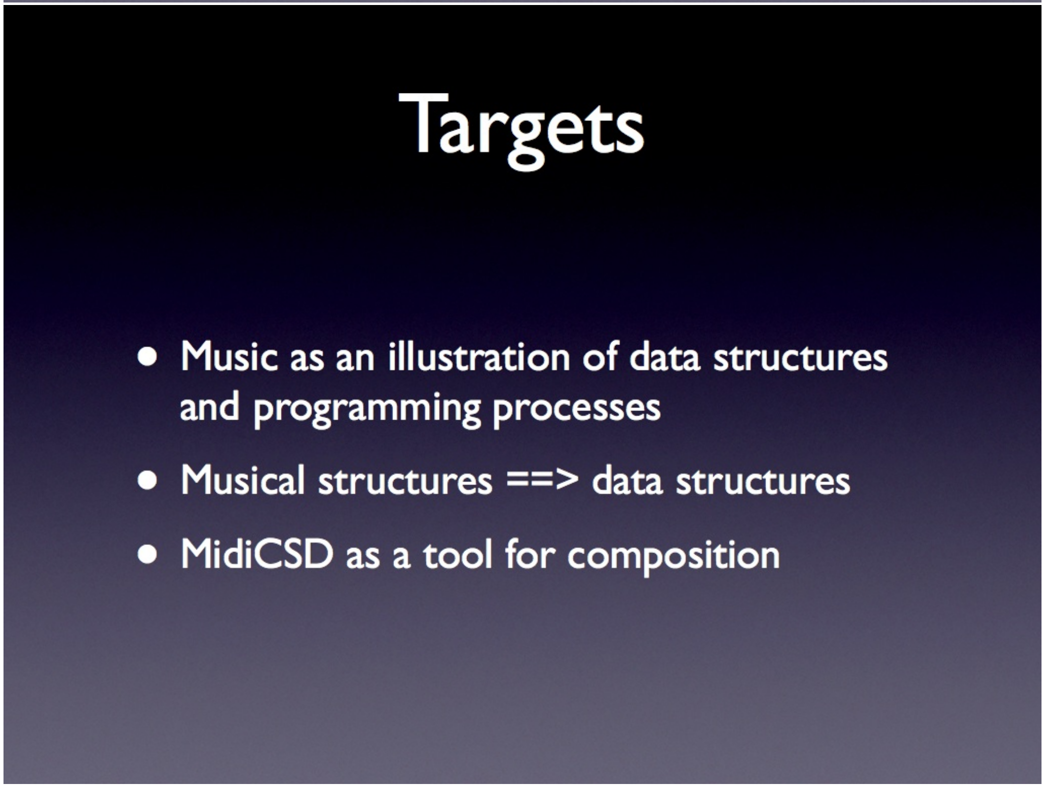
When you are finished, hand in a print version of the phrases and the macro language code you created. Also hand in the audio file.

9.4 Presentation for the evaluation



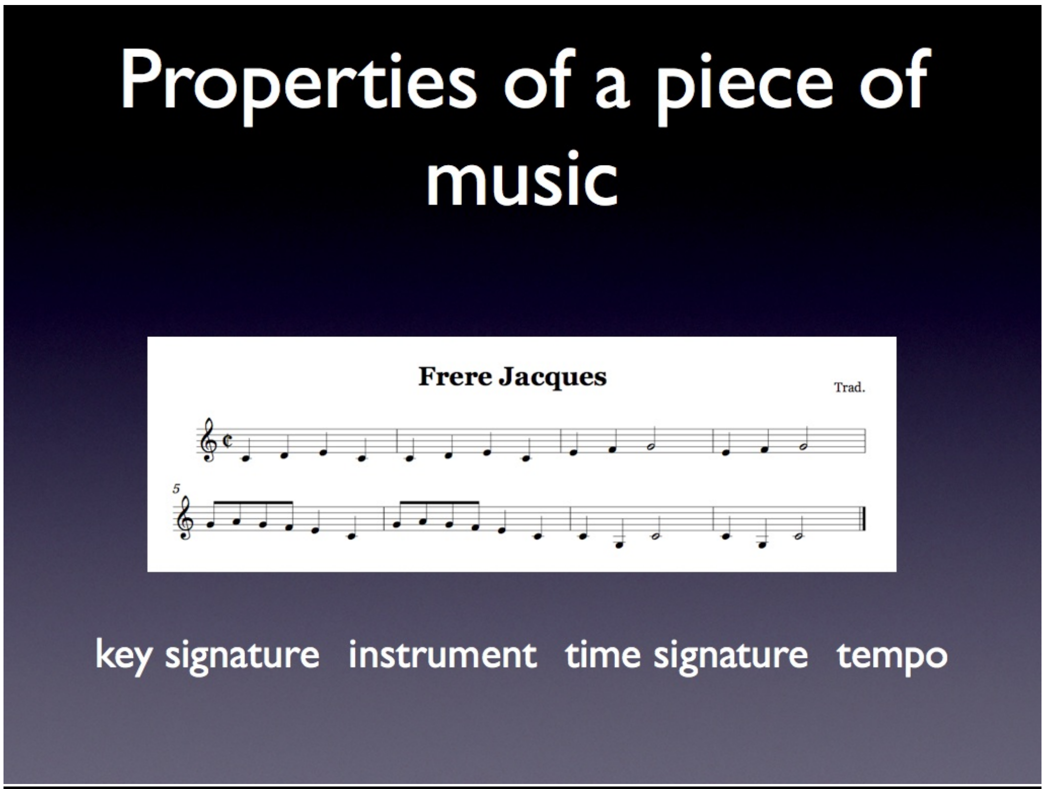
Algorithms, Data Structures and Programming

with MidiCSD in MS Excel

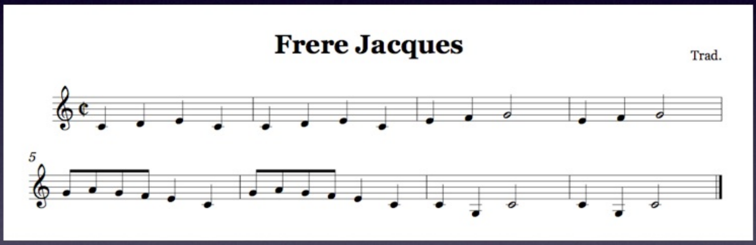


Targets

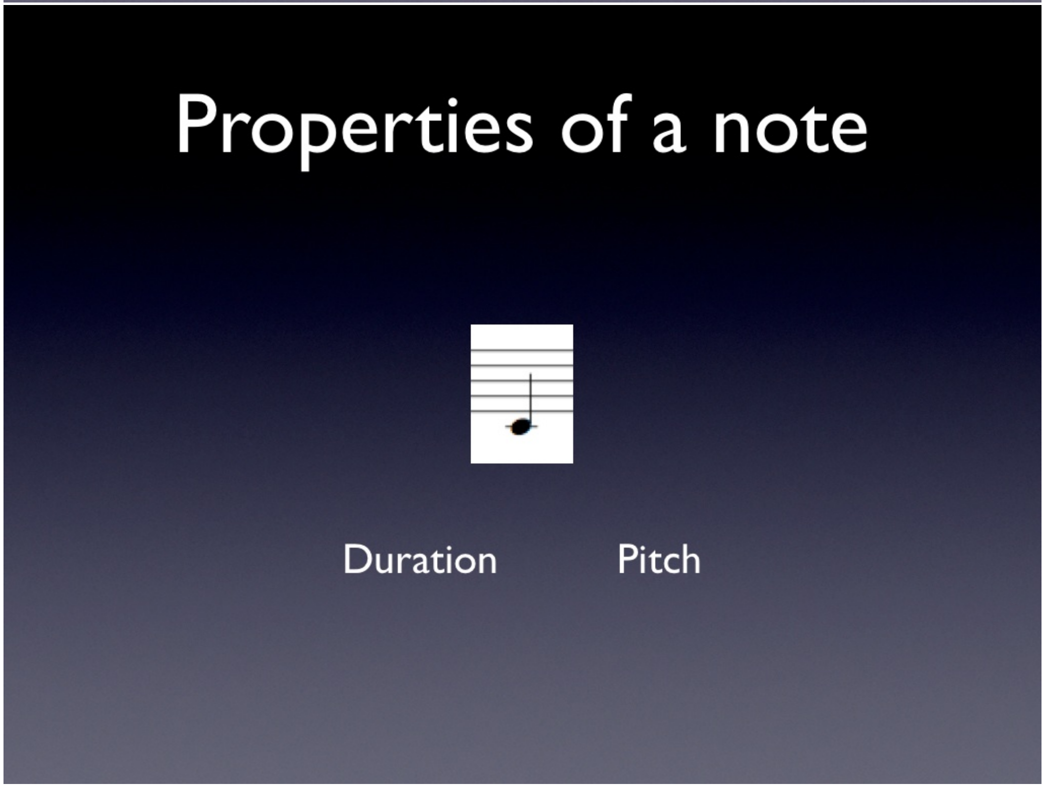
- Music as an illustration of data structures and programming processes
- Musical structures ==> data structures
- MidiCSD as a tool for composition



Properties of a piece of music

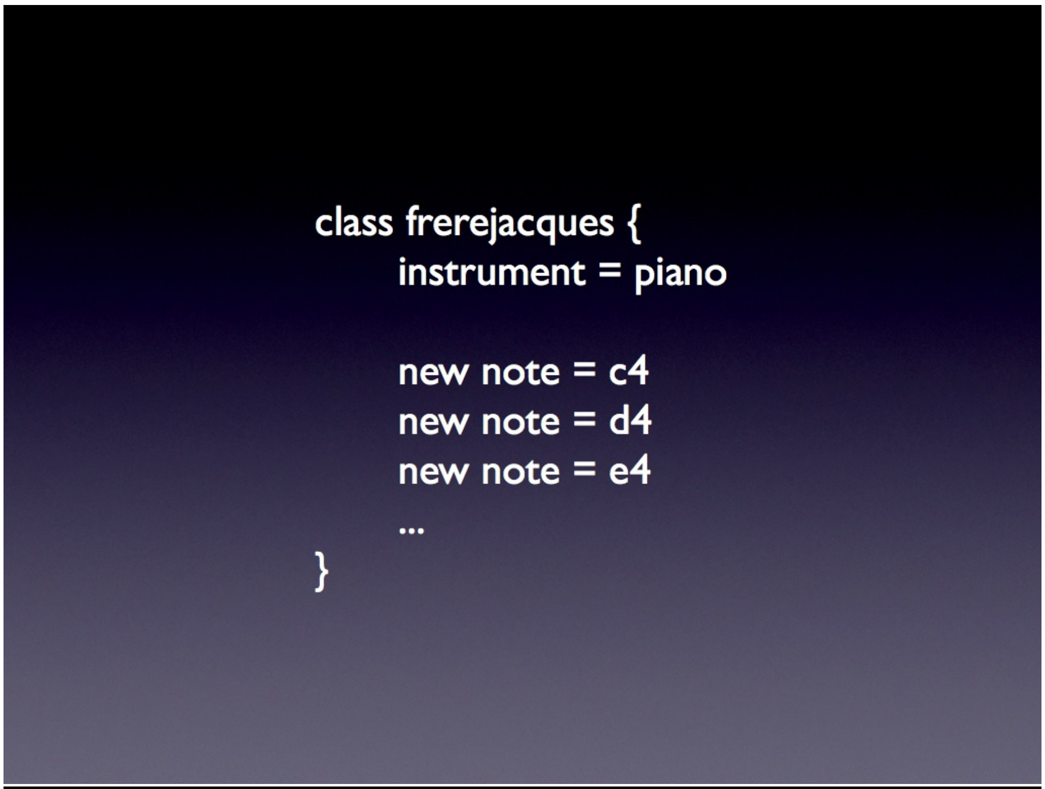


key signature instrument time signature tempo

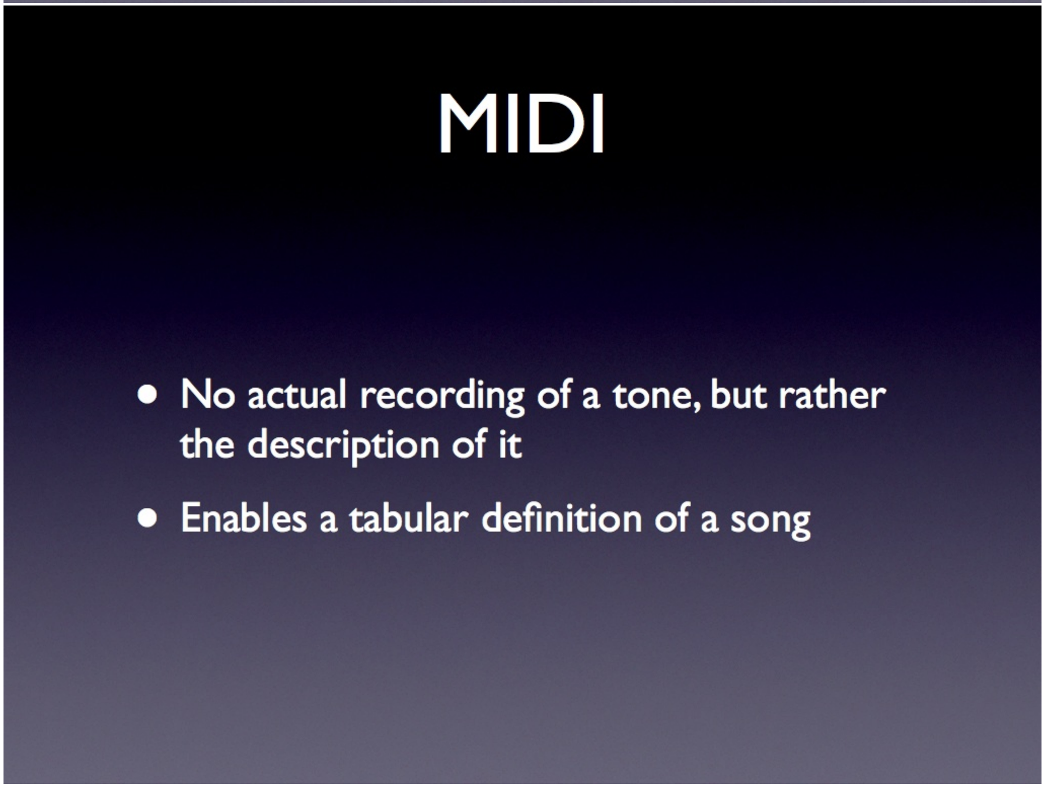


Properties of a note

Duration Pitch

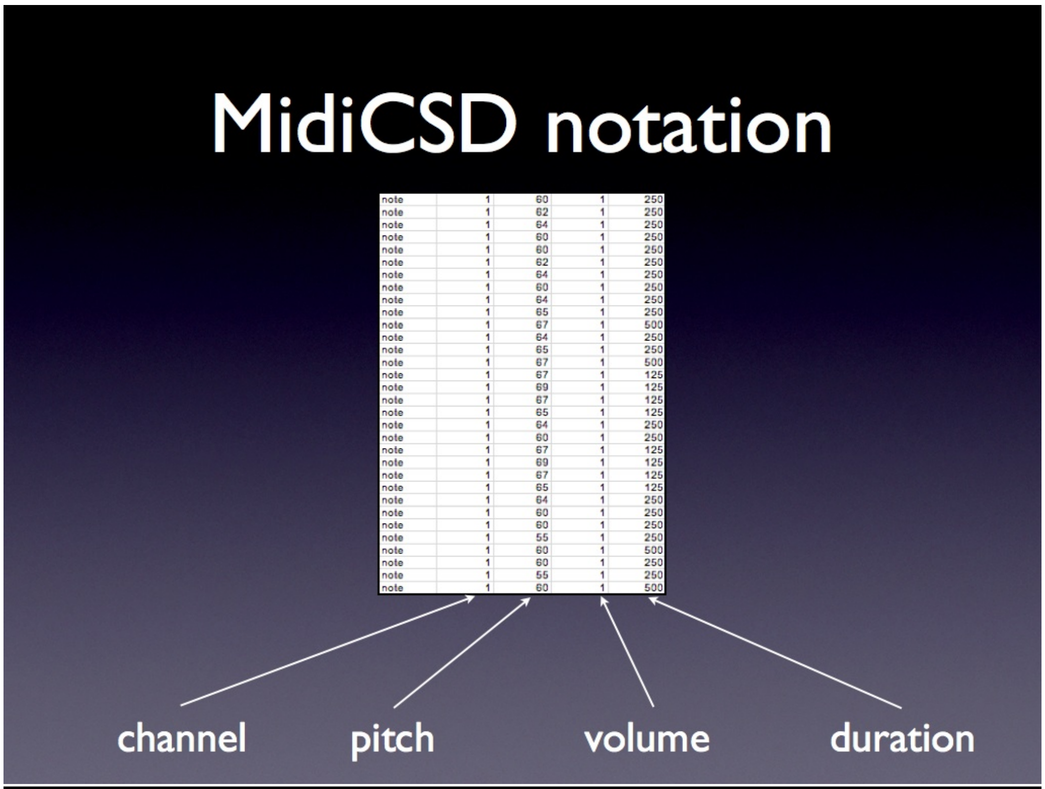


```
class frerejacques {
        instrument = piano

        new note = c4
        new note = d4
        new note = e4
        ...
}
```



MIDI

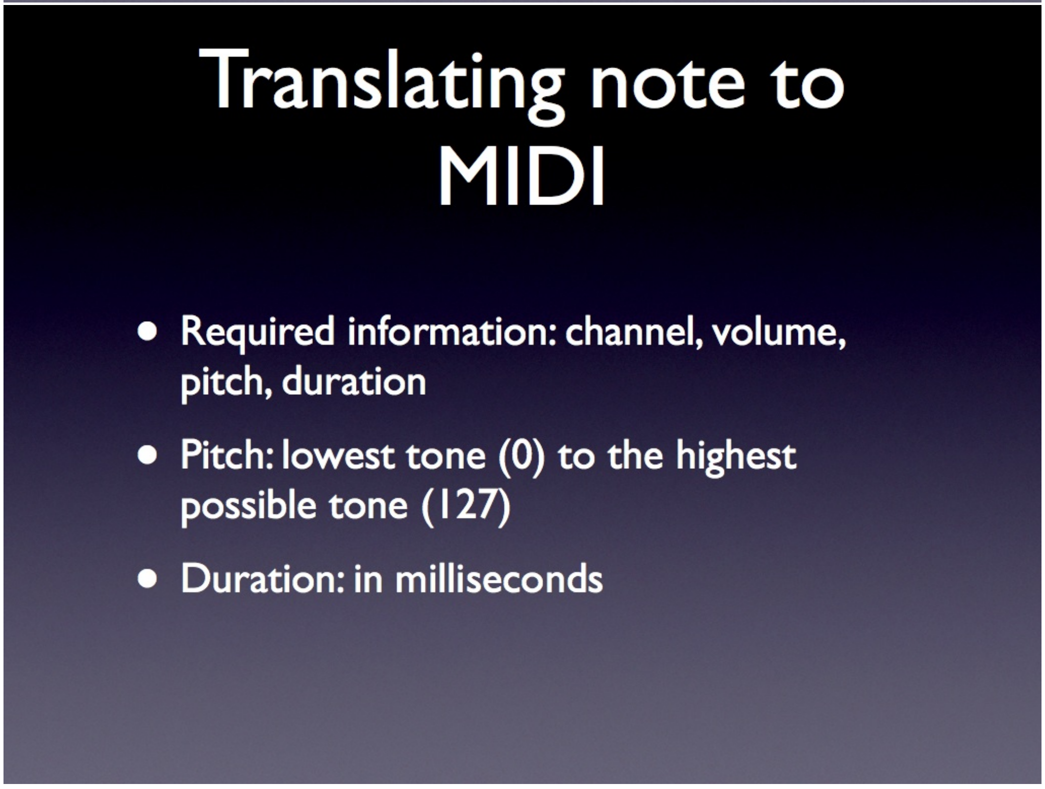
- No actual recording of a tone, but rather the description of it
- Enables a tabular definition of a song



MidiCSD notation

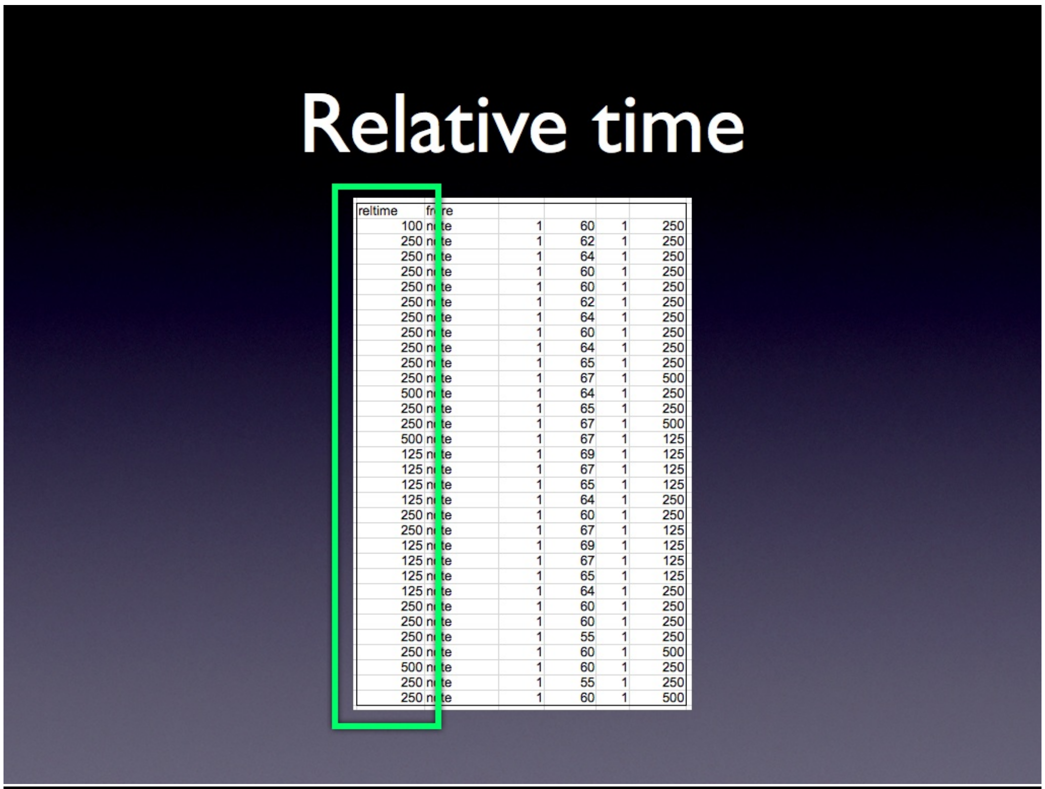
note	1	60	1	250
note	1	62	1	250
note	1	64	1	250
note	1	60	1	250
note	1	60	1	250
note	1	62	1	250
note	1	64	1	250
note	1	60	1	250
note	1	64	1	250
note	1	65	1	250
note	1	67	1	500
note	1	64	1	250
note	1	65	1	250
note	1	67	1	500
note	1	67	1	125
note	1	69	1	125
note	1	67	1	125
note	1	65	1	125
note	1	64	1	250
note	1	60	1	250
note	1	67	1	125
note	1	69	1	125
note	1	67	1	125
note	1	65	1	125
note	1	64	1	250
note	1	60	1	250
note	1	60	1	250
note	1	55	1	250
note	1	60	1	500
note	1	60	1	250
note	1	55	1	250
note	1	60	1	500

channel
pitch
volume
duration



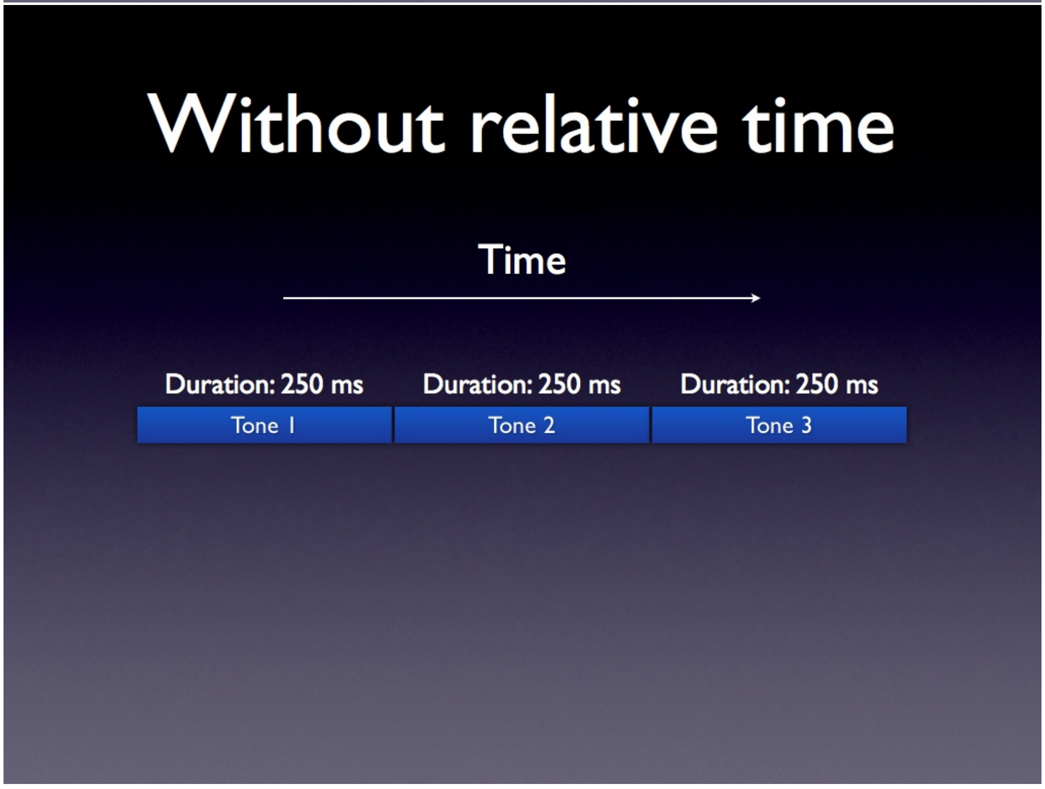
Translating note to MIDI

- Required information: channel, volume, pitch, duration
- Pitch: lowest tone (0) to the highest possible tone (127)
- Duration: in milliseconds

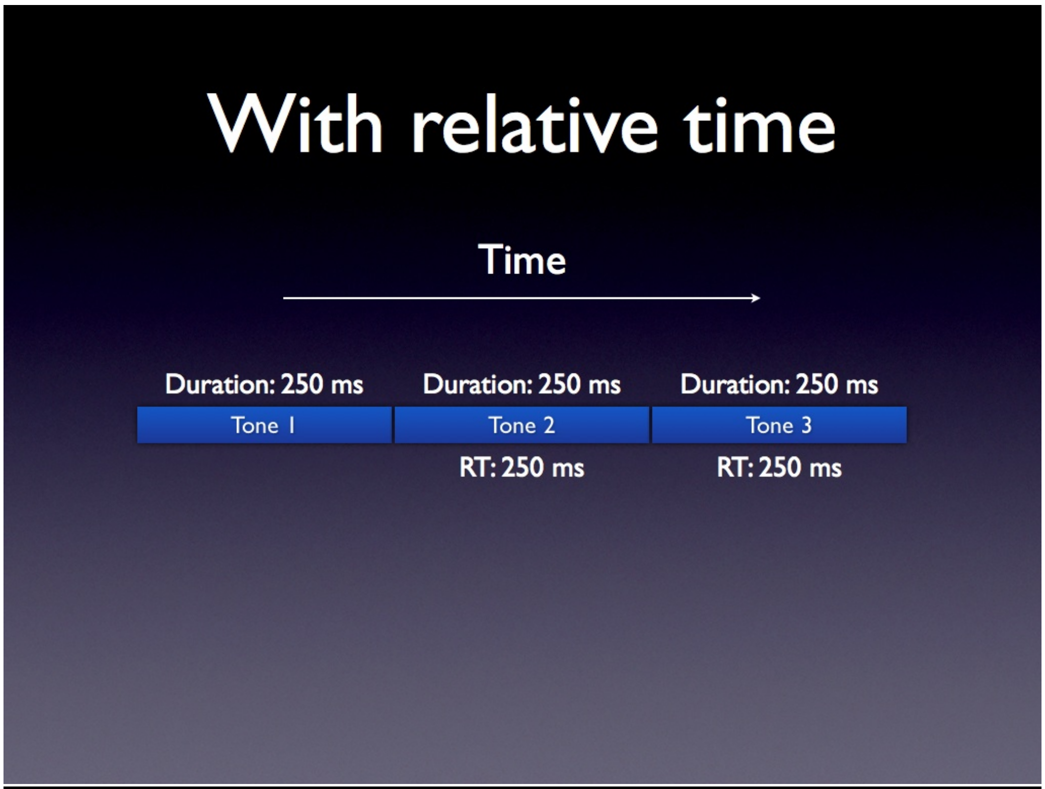


Relative time

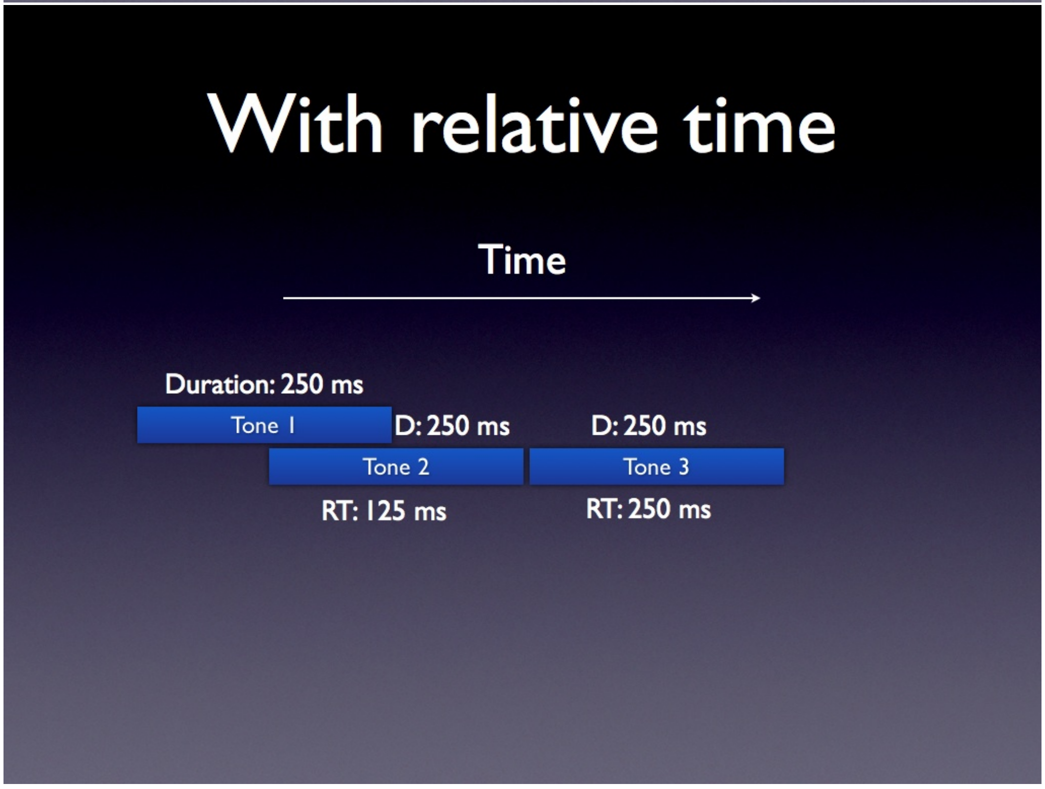
reltime	fr re				
100	n te	1	60	1	250
250	n te	1	62	1	250
250	n te	1	64	1	250
250	n te	1	60	1	250
250	n te	1	60	1	250
250	n te	1	62	1	250
250	n te	1	64	1	250
250	n te	1	60	1	250
250	n te	1	64	1	250
250	n te	1	65	1	250
250	n te	1	67	1	500
500	n te	1	64	1	250
250	n te	1	65	1	250
250	n te	1	67	1	500
500	n te	1	67	1	125
125	n te	1	69	1	125
125	n te	1	67	1	125
125	n te	1	65	1	125
125	n te	1	64	1	250
250	n te	1	60	1	250
250	n te	1	67	1	125
125	n te	1	69	1	125
125	n te	1	67	1	125
125	n te	1	65	1	125
125	n te	1	64	1	250
250	n te	1	60	1	250
250	n te	1	60	1	250
250	n te	1	55	1	250
250	n te	1	60	1	500
500	n te	1	60	1	250
250	n te	1	55	1	250
250	n te	1	60	1	500



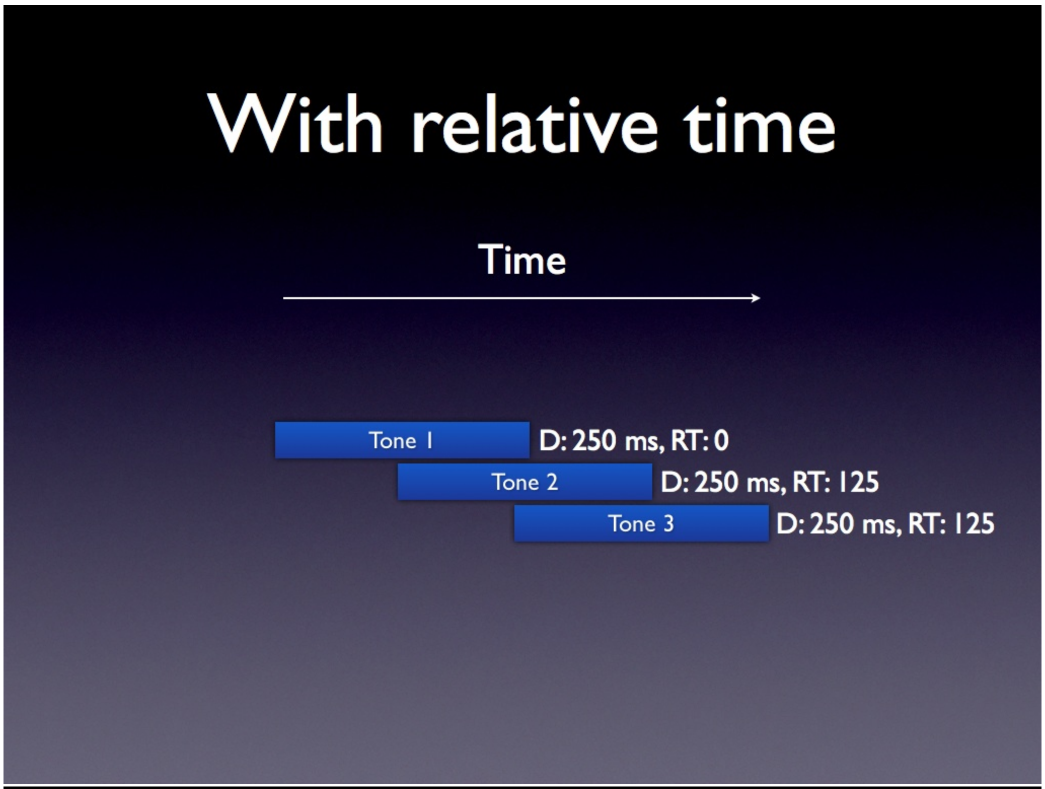
Without relative time



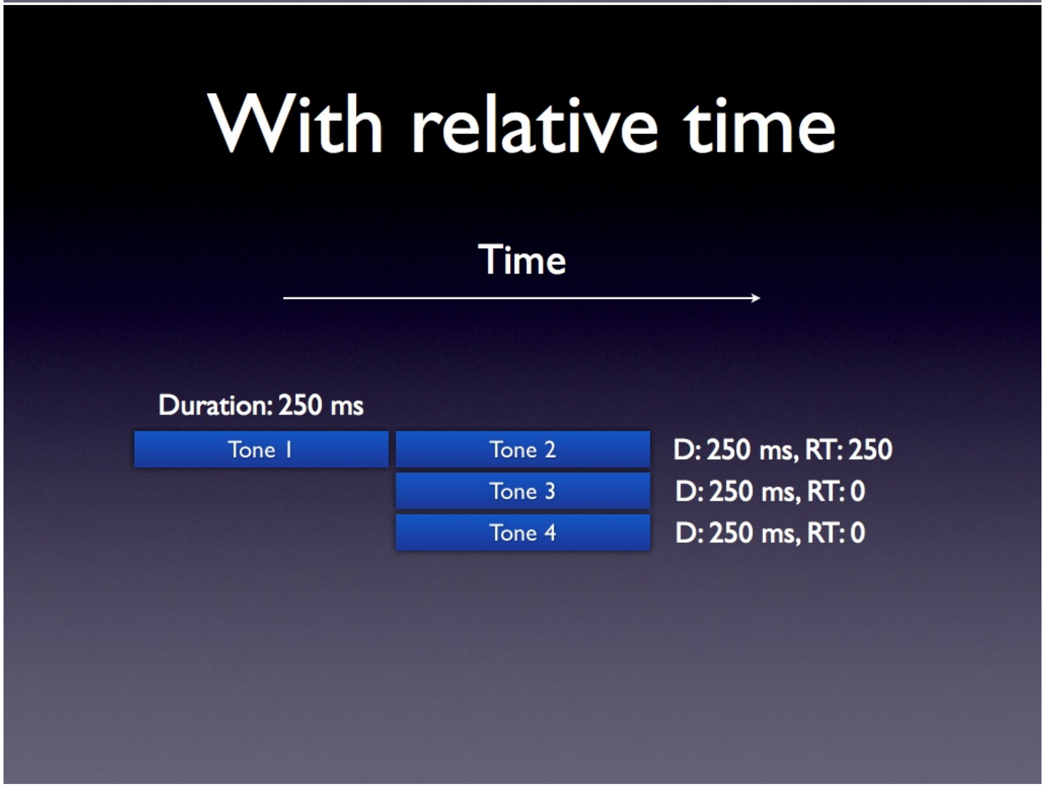
With relative time



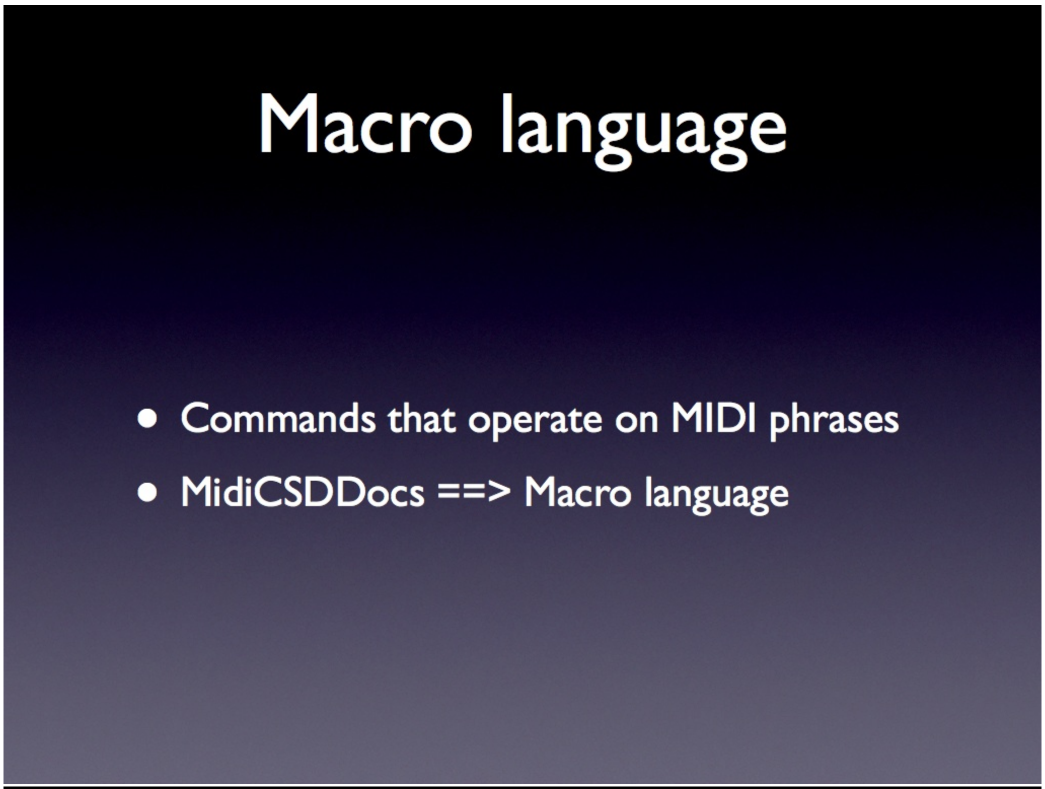
With relative time



With relative time

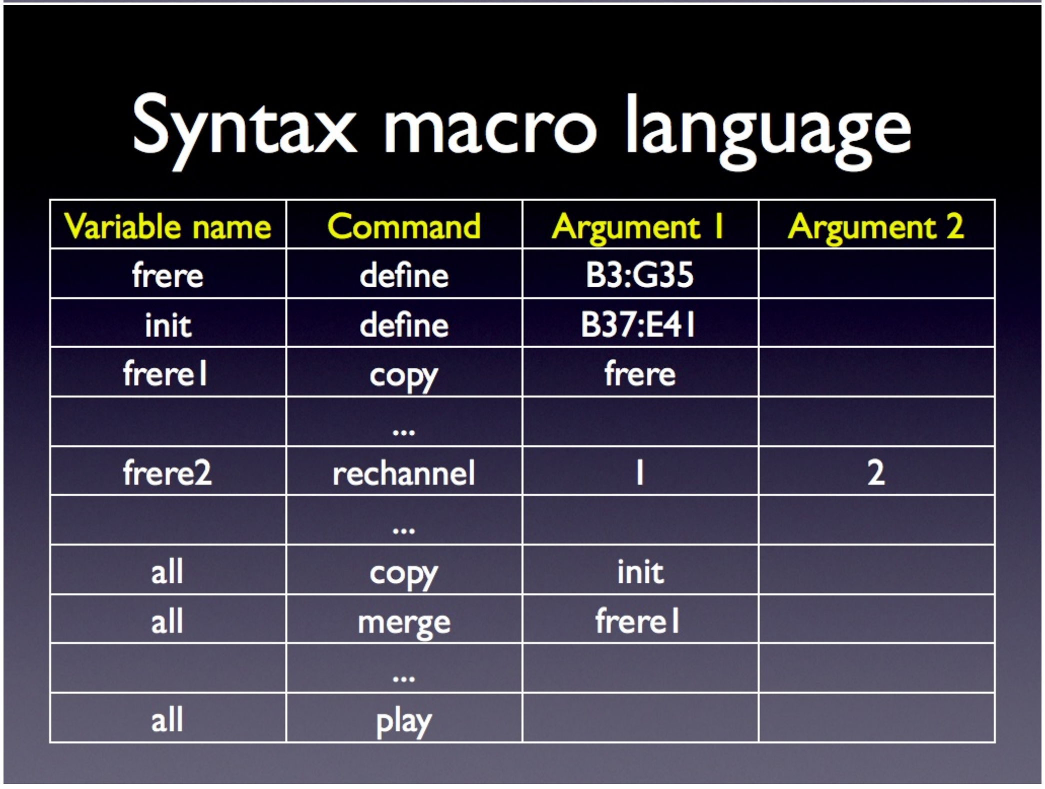


With relative time



Macro language

- Commands that operate on MIDI phrases
- MidiCSDDocs ==> Macro language



Syntax macro language

Variable name	Command	Argument 1	Argument 2
frere	define	B3:G35	
init	define	B37:E41	
frere1	copy	frere	
	...		
frere2	rechannel	1	2
	...		
all	copy	init	
all	merge	frere1	
	...		
all	play		

Let’s get going!

9.5 Task sheet for the evaluation

Diploma Thesis Project: Composition with MidiCSD

Data for statistical purposes:

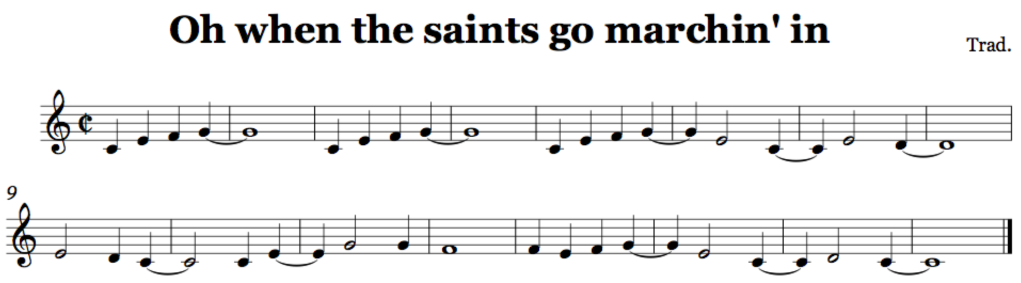
Age:

Degree:

Gender:

Tasks

It is absolutely necessary to complete task 1 in order to be able to fulfill the rest of the exercises. After completion of task one please select at least 2 other tasks to solve. For exercises 1 and 2 no macro language is necessary, but for all other tasks it is required.



Task 1: Song translation

In Excel, create the MIDI table that corresponds to this song. Find out, how the pith numbering has to look like and how the note values shall be represented in terms of their duration in milliseconds. Include the relative time in the table as well.

Task 2: Constructing chords

Ornament the song by adding chords! Note, chords do not necessitate the development of an additional voice, they merely add notes to the same voice.

Task 3: Going backwards

Which macro code plays the song backwards?

Task 4: Going up

Transpose the song upwards by 2 tones.

Task 5: Instrument parade

Create the macro code which plays the song three times in a row, but with different instruments.

Task 6: Strange sounds

Invert the song (i. e. the exact opposite of the pitch structure - if the note in the piece is one tone higher as the last one it should be lowered in the inverted version). Play it simultaneously with the 'correct' song.

Task 7: Polyphony

Add a second voice to the song (it does not necessarily need to contain chords - one tone is fine). How do you realize this and how does the macro code look like?

Task 8: Orchestra composition

Combine tasks 2 and 7. Assign different instruments to the voices and you will receive an orchestra version of the song!

Task 9: Free composition

Either use the example song or a composition of you own to create a free version of the song. You can freely choose which effects to use in the process.

Task 10: Create MIDI file

Use the button 'WriteFile' to create a MIDI file of your composition.

9.6 Curriculum Vitae of Rainer Dangl

Personal Data

Birth date and venue:	26th July 1984, Krems an der Donau
Marital status:	unmarried
Citizenship:	Austria

Education

2004-2010:	Studies of Computer Science and English at the University of Vienna and the University of Technology of Vienna (Special degree for teachers). Joint Study Exchange in 2008 at Macquarie University in Sydney, Australia
1998-2003:	Business school in Waidhofen/Thaya, Focus information management and multimedia
1990-2008:	Primary and secondary school in Waidhofen/Thaya

Military service from September 2003 to April 2004

Job Experience

Since 2008:	Computer Science Teacher at GRG Zirkusgasse in Vienna