

COMS W4115 Programming Languages and Translators

CHRONOS

Chronologically Organized Scheme

Stephen Somuah – Project Manager – sys2111 @ columbia.edu

Lishan Made – Language & Tools Guru – lba2125 @ columbia.edu

Tikue Anazodo – System Architect – tca2111@columbia.edu

Richard Nwaobasi – System Integrator – rcn2105@columbia.edu

Shannon Lee – Tester & Validator – sil2116@columbia.edu

CHRONOS

CHRONOS: A simple, intuitive, imperative programming language for creating course schedules for college students. As college students, we are intimately aware of the difficulties involved in deciding on what array of classes to take in a given semester. CHRONOS seeks to enable students to easily program the way in which an input file of all the courses they are considering should be handled to output a possible schedule. With minor tweaks to the file listing the courses, or the program itself, a wide range of schedules can be created that meet specific and varying needs that most students would have.

**Scenario**

Sam is interested in CS, Architecture, Statistics, Economics, and Music. Basically he has so many interests that he cannot even begin to think of a concrete and feasible class schedule. He would therefore like to write a program that would take all the classes he is contemplating taking this semester, and would be able to compute the various schedules that he could end up with, give a maximum number of classes or a maximum number of credit points as a constraint.

## Predictable / Structured

One of CHRONOS’s aims is to allow students with minimal programming experience to be able to write a program. Therefore, the language is designed to be very structured to allow students to follow a fixed pattern to write code that they can easily understand and can be learned by looking at a few sample programs. The structured nature of the programs, together with the specific use of the language also ensures that from semester to semester, much of the same code is re-used because the underlying organization is very structured, even if your interests have changed drastically from the previous program

**Powerfull**

CHRONOS does not allow the simplicity and structured nature of its programming to make it a

**Simple**  
Because the entire essence of our language is to simplify the task of creating class schedulers, we made the syntax and grammar at the core of our language as simple as possible. The goal is to make the task of scheduling much easier by allowing users to create input files of the classes they are deciding between in the order of most preferred to least preferred. This separation allows the programmer to easily change the ranking of courses, or to even re-use the program in other semesters, simply by changing the courses in their input file.   
  
We have basic programming language concepts similar to those used in the more mainstream programming languages such as the concept of loops and conditionals at the core of our language. This makes for a smooth transition from any language that a user might have previous experience with to our language. Even in the case that the developer using our languages has no previous programming experience, the syntax of the language is intuitive enough that the user is able to fully understand and create their own programs merely by looking at a few sample programs.

Summary

Appendix

Sample code to create a schedule with a minimum number of courses and credits:

new schedule sched;  
new courseList CL = generateCourses(infile);  
  
/\* Creating some int variables \*/  
int courseMax = 6; // maximum number of courses  
int courseMin = 4; // minimum  
int creditMax = 20; // maximum number of credit points  
int courseMaxR = 2; // maximum number of courses on Thursday  
  
/\* Add classes only if conditions are met \*/  
foreach course c in CL  
{

if (sched.numCourses() <= courseMax

&& sched.numCredits() + c.numCredits() <= 20)

{

if (c.day() == R && sched.numCourses(R) >= courseMaxR)

break;

else

sched.addCourse(c);

}  
}   
  
/\* Print some string output \*/  
if (sched.numCourses() < courseMin)  
print(“Four non-conflicting classes could not be found. Add more potential courses!”);