A C-based Introductory Course Using Robots

David Cowden
Grinnell College
Grinnell, Iowa 50112 U.S.A.
+1 215-326-9336
cowdenda@grinnell.edu

April O'Neill
Grinnell College
Grinnell, Iowa 50112 U.S.A.
+1 641-269-9059
oneilla@grinnell.edu

Erik Opavsky
Grinnell College
Grinnell, Iowa 50112 U.S.A.
opavskye@grinnell.edu

Dilan Ustek
Grinnell College
Grinnell, Iowa 50112 U.S.A.
ustekdil@grinnell.edu

Henry M. Walker
Grinnell College
Grinnell, Iowa 50112 U.S.A.
+1 641-269-4208
walker@cs.grinnell.edu

ABSTRACT
Using robots in introductory computer science classes has recently become a popular method of increasing student interest in computer science. This paper describes the development of a new curriculum for a CS 2 course, *Imperative Problem Solving and Data Structures*, based upon Scribbler 2 robots with standard C. The curriculum contains eight distinct modules with a primary topic theme, readings, labs, and project at the end. Each module resulted from collaboration among former CS 2 students and a faculty member, utilizing an iterative process with revisions. Each lab includes a survey to obtain student feedback that will allow the course to evolve and better fit the needs of future CS 2 students. All materials discussed here are available online for use by others.

Categories and Subject Descriptors
*K.3.2 [**Computers and Education**]: Computer and Information Science Education – computer science education, curriculum*

General Terms
Design, Experimentation, Languages

Keywords
Robots, course materials, lab-based course, collaborative learning, modules

1. INTRODUCTION
This paper describes an innovative approach for incorporating robots within the context of a second course of a three-course multi-paradigm introductory sequence to computing. In summary, this work has at least four vital characteristics:

- Scribbler 2 robots are introduced in the context of imperative problem solving and C, in order to fit within a multi-paradigm sequence;
- The course is divided into eight modules, each with readings, labs, and a project; and each module highlights capabilities of the robots;
- Each module was developed by a faculty member working with upper-level students, so the instructor could provide pedagogical guidance, and the students could encourage applications that would interest introductory students; and
- For each class meeting, a feedback/evaluation process is tied to the readings and labs to help the refinement of course materials over several semesters.

Both the software environment (using C that links to C++ object code) and all course materials are publicly available (via launchpad.net [11, 12] and the course home page [15], respectively). Thus, faculty interested in experimenting with robots within C-based courses have a complete collection of materials to start with.

The rest of this paper describes this course and the process of developing it in some detail. Section 2 describes the multi-paradigm context of the course, the selection of C, and the motivation for an application-oriented theme. Since a standard C interface was not available for the Scribbler 2 robot, Section 3 outlines the work needed to adapt a Myro-C++ package from the University of Tennessee at Knoxville [12], so that students can write their CS 2 programs in standard C. Section 4 reviews the goals, pedagogy, and organization for this lab-based course, and Section 5 reviews the process of developing both the C interface and the course modules. Since the Myro-C++ package was already available from launchpad.net [12], the development group wanted the C-based materials (called MyroC) to also be available on launchpad.net [11], and Section 6 describes some resulting interactions with launchpad. The paper concludes with a consideration of next steps (Section 7), acknowledgments (Section 8), and references (Section 9).

2. CONTEXT: ROBOTS & C IN CS 2
In recent years, many computing educators have experimented with connecting problem-solving and programming with application themes in introductory computing courses. For example, [1, 2, 11] describe experiments using media computation to motivate students; [4, 6, 12] discuss the use of robots, and [10] describes music as an application area. These experiments regularly demonstrate considerable success in engaging students and encouraging them to try another computing course.

As another theme for introductory computing, many schools utilize a multi-paradigm approach for introductory computing, following curricular recommendations from ACM [1, sections 3.2.4 and 9.1.5] and the Liberal Arts Computer Science Consortium [7]. For example, [13] describes several approaches for functional problem solving within Scheme in the first course. In particular, within [13], Rebelsky and Davis describe the use of

SIGCSE 2012, February 29-March 2, 2011, Raleigh, NC, U.S.A.

media computation and image processing as an extremely successful application area for a CS 1 course that emphasizes functional problem solving and Scheme.

With this attention to application areas for CS 1 within a multi-paradigm context, relatively little discussion has focused on an application area for the second course in the sequence. Specifically, for the course described here, any application area needed to meet these criteria:

- CS 1 covers functional problem solving, Scheme, and media computation, so an application area for CS 2 should complement this prerequisite work.
- CS 2 focuses on imperative problem solving.
- CS 2 explores C and low-level details including data allocation, memory allocation, and data representation.

Altogether, the use of robots with programming in C naturally fits this framework. Further, the course should not place extensive emphasis on hardware construction, since building robots would likely detract from the central theme of imperative problem solving with C.

In reviewing available platforms, the Scribbler 2 robot seemed a natural choice, because it is widely used at the introductory level [4, 12] and extensive materials are available to support courses using Python [4, 6] and C++ [8]. Unfortunately, neither the Scribbler 2 or other platforms seemed to have mature environments for programming in C.

A natural development path, therefore, seemed to base software development on the Myro-C++ materials from the University of Tennessee at Knoxville [12] and develop a C interface that could call compiled C++ materials that utilize 'extern "C"' directives.

With a clean, clear C library, work then could focus on the develop of course materials.

3. USE OF C

Out of the box, the robots are not ready for classroom use. Our group extended and refined the available material so the robots function in alignment with the desired course curriculum. The following subjects were considered when revising the robot functionality.

3.1 Choice of the C Programming Language

We chose to use C with the robots in order to constrain students to a strict imperative problem solving approach as indicated by the course title and description. This is highly compatible with previous courseware since C was the language of choice for our prior CS 2 course. Although it would have been possible to use the C subset of C++, we used strict C to retain the pure nature of the standard C specification and to remove extra features that might distract students from the imperative nature of standard C.

3.2 Drawing Upon Existing C++ Material

The de facto implementation of the IPRE Scribbler communication protocol is provided by roboteducation.org in the Python programming language [4]. To adapt the robots for use in an imperative programming course using C, we had two options: We could write our own implementation of the library, or we could adapt the publicly available C++ version from the University of Tennessee at Knoxville [3]. Since C++ supports 'extern "C"' function declarations, the most logical approach was to simply wrap existing C++ member functions of the robot class with C-style functions that mask all object-oriented interactions from the user.

3.3 Definition of MyroC

Using the 'extern "C"' keyword, it is possible to write a C header for a C++ library. When the header containing 'extern "C"' declarations is included, the compiler creates function symbols C can link against. Although our header file allows students to call procedures in standard C, the underlying library is implemented in C++ and can reference C++ objects in other C++ libraries. For example, if we want to enable the user to get the name of the robot, we simply provide an 'extern "C"' function that calls the robot object's getName member function. Our function then converts the C++ string returned by robot.getName() to a C string and returns a pointer to the first character in the string.

In certain cases is was necessary to discard the extra level of abstraction provided by a higher level language such as C++ in order to make functionality compatible with standard C. As shown by the previous example, users have no notion of objects or that a robot has the function getName, etc. The C++ library also utilizes smart pointers. Smart pointers, the C++ solution to memory management, delete the data they point to if they are the last reference to that data when they fall out of scope. Since the smart pointers are implemented as objects that override the unary dereference operator, it was necessary to discard the notion of smart pointers altogether and simply return normal pointers to the C caller.

Finally, it is possible to retain objects on the C++ side and pass pointers to those objects back to the C caller. Rather then re-write all the Picture class member functions to operate on raw picture data, any time a picture was created we kept the entire class on the heap and simply passed its location on to the caller. Later, the caller can pass that pointer back to the library and it can be treated as the C++ class that it is. This method of interchanging objects between C and C++ is accomplished with a simple typedef. Because the size of a pointer is consistent between the two languages, a struct with the same name as the C++ class can be declared. The C compiler now knows there is a stuct named Picture (in our case) but does not know how big that struct is. This is acceptable because we are only passing pointers to the struct; something the compiler does know the size of. .

4. COURSE PLANNING

Initial explorations with the Scribbler 2 suggested that the robots would interest many students and be a valuable addition to the CS 2 course. Subsequent course planning considered how the inclusion of robots would fit with curricular goals, course organization, and pedagogical considerations.

4.1 Curricular Goals and Course Topics

The initial work of integrating the Scribbler 2 into the course required identifying the following course goals:

- introduce imperative problem solving and data structures;
- teach the fundamentals of programming robots;
- provide students with gain and practice knowledge of the C programming language;
- increase interest in continuing the study of computer science;
- describe elements of the representation of integers, floating-point numbers, characters, and other data structures; and
- introduce concepts and skills related to a Linux environment.

Some main topics for the course are:

- *imperative problem solving:* top-down design, common algorithms, assertions, invariants;
- *C programming:* syntax and semantics, control structures, functions, parameters, macro-processing, compiling, linking, program organization;
- *concepts with data:* data abstraction, integer and floating-point representation, string representation, arrays, unions, structures, linked list data structures, stacks, and queues;
- *machine-level issues:* data representation, pointers, memory management; and
- *GNU/Linux operating system:* commands, bash scripts, software development tools.

4.2 Use of Modules

Course organization and development proceeded in two stages, the second of which was successful. At first, we tried appending Scribbler 2 activities to existing labs on each topic, but this approach did not provide a coherent structure for the robots or fully utilize their potential.

Instead, we organized topics required for CS 2 course into eight groups, called modules. For each module, we identified readings for each topic, outlined laboratory activities, constructed possible examples (i.e., C programs), and devised a project that would integrate the topics of that module. Next we arranged the topics within each module and the order of the modules from foundational concepts to more advanced uses of imperative problem solving. With this framework, we considered how each module would fit within an overall day-by-day semester schedule.

Each module has a unifying theme, lasts six or seven class days, and utilizes a common format. On the first day of each module, the instructor introduces the topics to be covered over the course of the module, demonstrates examples of programs using those concepts, and may begin discussing the fundamentals of one or more topics. For each of the next few days, students are assigned to do the readings on the topic for the next lab, and the class periods are a mixture of lectures and laboratory practice of the topics introduced on each day. Near the end of the module, students have one to two class periods to work on an assigned project that integrates most or all of the concepts discussed in the module as well as concepts introduced in previous modules. Projects for several modules are closely related to previous ones, in order to reinforce the connectedness of the topics and review previous material. Overall, the modules are designed to introduce course topics to the students, discuss and practice the topics, and then review and apply the knowledge gained.

4.3 Pedagogical Considerations

Much course planning and materials development was guided by considerations of how and what students learn. Some important elements of pedagogy include:

- lab-based approach: the overall course includes about 35 laboratory sessions;
- student practice in solving problems: numerous exercises build experience with imperative problem solving;
- sequencing of topics: complex topics are spread over several days;
- emphasis on pair programming: the course encourages collaboration on lab activities and projects (in and out of class);
- range of lab activities: laboratory exercises include an introduction to the topic, simple usage (including student descriptions/commentaries), writing code, testing, comparing approaches, and identifying advantages and disadvantages of various implementations;
- readings: in-class lab activities are supported by readings from a textbook and/or Web-based materials; and
- projects: a final activity integrates previous work.

4.4 Semester Outline

Table 1 illustrates how central topics in imperative problem-solving were mapped into new modules that also allow exploration of the Scribbler 2 robots. In addition, the course includes three in-class hour tests, a day for end-of-course evaluations, and a three-hour final exam. A more complete outline may be found at

http://www.cs.grinnell.edu/~walker/courses/161.fa11/semester-outline.shtml

5. PROJECT DEVELOPMENT

The development process included initial experimentation, development of a C interface, writing C programs, designing modules, and writing the course materials. As part of the development work, we presented our work to colleagues throughout the summer to get feedback on our work.

The development of materials represented a collaboration between students and faculty. Two students had recently taken the course, so they understood perspectives of the target audience. Two students had more technical experience. The faculty member contributed technical background, coordinated the project, and provided pedagogical and curricular perspectives.

5.1 Initial Experimentation

After conducting experiments in Python with the Scribbler 2 robots, we concluded that some important capabilities included obtaining sensor data, beeping in different frequencies, moving back and forth, turning, taking photos with the built-in camera on the robot, and manipulating these photos.

5.2 Development of the C Interface

Given that Scribbler 2 robots had a library in C++, and CS 2, *Imperative Problem Solving and Data Structures*, would be in C, our next step was to make a wrapper library that translated C++ into C so that the CS 2 students could write programs in C, and the robot could execute them. This was the first draft of MyroC.cpp and its header file MyroC.h. MyroC was a continuous effort throughout the project. We kept track of the revisions that were released of Myro-Cpp and if there was something that was of use to us, we updated our MyroC. This sometimes was a problem because some parts of the revised Myro-Cpp did not compile with our MyroC. The final version of MyroC was not reached before we were completely done with the project, because Myro-Cpp was updated several times with new functions that were beneficial for our use, and documentation was improved many times before the end.

5.3 Writing C Programs

After we were able to compile C programs for the robot to follow, we tested MyroC for every function to make sure they all worked the way we wanted them. We then started to write demonstration programs for range of robot capabilities such as moving to avoid objects, dancing, playing various songs, and controlling the robot using command line arguments.

Table 1: Semester Outline for Imperative Problem Solving and Data Structures

Module	Days Allocated	Overall Theme	Summary and Main Topics
0	7	*Introduction: Play a song*	the Linux environment, elements of a C program, introduction to Scribbler 2
1	6	*Motion (+song, motion)*	variables, types, conditions, loops, Scribbler 2 motion
2	6	*Motion and sensors*	1-dimensional arrays, testing, functions, parameters, values, addresses, & operator, Scribbler 2 sensors
---	3	*Data representation*	integer and floating point representation
3	6	*Deepen understanding of imperative problem solving and C*	characters, strings, input and output
4	6	*Image Processing*	2-dimensional processing, transforming a pixel, transforming a picture
5	6	*Music Composition*	musical notes as frequency/duration structs, pointers, linked lists
6	6	*Storage/retrieval Patterns*	stacks, queues, Bash scripts
7	6	*Integrated Robotics*	files, command-line arguments, robot motion with logging

5.4 Initial Approach

When we had a good idea of what the Scribbler 2 robots could do, and had written various programs that were aligned with the topics of the course, we tried rewriting the previously existing labs to include the robots. The structure of this course alternated lecture, lab, lecture, lab, etc. However, this approach did not give a view of the capabilities of the robots. The use of robots seemed forced rather than natural.

5.5 A Better Approach: Modules

As a second approach, we organized the course into 8 modules. A module, as we defined it, would each last approximately two-weeks, and would comprise a part of the course that would begin with lecture and examples, continue with a few days of labs, and end with more extensive projects that would require the students to use cumulative knowledge to complete.

After deciding on the structure of the course, and making a day-by-day schedule of the topics and their labs, we wrote the various components of the modules in pairs; all of the readings and labs were written in HTML format.

5.6 Writing the Course Materials

Students worked in different pair groupings to develop multiple iterations of each module. The approach to writing modules included 4 steps:

1. A pair made a module outline containing readings that corresponded to each topic, links to the labs and a final project.
2. The pair wrote a first draft of each lab, project, and reading.
3. Each pair reviewed their work with the faculty leader.
4. The authoring pair then gave the module to others, who had not seen it, for a test run. The test run was done by recording the time each lab took, and giving the authors comments and suggestions the testers had on each aspect of the module.
5. The authors performed the final revisions, resulting in a final draft of the module.

By the end of development, modules, readings, examples, labs, and projects were all collected into one course website,

http://www.cs.grinnell.edu/~walker/courses/161.fa11/

These materials are available to students in the course as well as to other computer scientists doing work in related fields, and educators wanting to teach the same course with these robots.

6. INTERACTIONS WITH LAUNCHPAD

To benefit as many people as possible, our work, consisting of the MyroC library, header specification, Makefile, and README, is publicly available through Launchpad.net on the world wide web, though a secure connection, at the location:

https://launchpad.net/myro-c

References to the course material can also be found there as well.

Launchpad was chosen because it is the home of the Myro-C++ project, which MyroC utilizes. Launchpad also boasts support for a variety of bug tracking solutions and effectively keeps track of all users' contributions to a project. Further, Launchpad's ease of use and welcoming nature factored into the decision on where to host our code.

7. CONCLUSIONS/NEXT STEPS

Previous work [4, 6, 8, 14] has shown that robots can raise student interest in computer science and potentially bring more students into the major. Our curriculum shows that the course can be structured around the use of robots and still cover all of the fundamental concepts of imperative problem solving, supported by the C programming language.

We have developed programming support and curricular materials for the use of robots in the teaching of CS 2 *Imperative Problem Solving and Data Structures* with the C programming language.

Programming support includes a rich C-callable header file and corresponding implementation (in C++) that allows students to write imperative programs in standard C and link to a C++ library. This C-based header and implementation is packaged as a MyroC project that is available online on Launchpad and provides support for most features available with Scribbler 2 robots using the IPRE fluke dongle.

The CS 2 curriculum with robots, which includes modules, readings, labs, and projects, is also available online, and is currently being field test in a classroom setting. The full package of 55 C programs and 57 Web pages is available from the authors for use for any instructors who wish to teach an introductory course in computer science with robots in the C language.

The innovative development process for these materials demonstrates the effectiveness of collaboration among former CS 2 students and a faculty member. The former CS 2 students provide deeper insight into the needs of future CS 2 students. The faculty member provided guidance and experience in the development and writing of a course. The full collaboration allowed an iterative review of all materials, including extensive testing of the MyroC package.

As current students field test the labs and projects, they will submit reviews on the material describing whether it is too long, too short, too confusing, etc. The material will then be revised by former CS 2 students and the faculty member to better fit the needs of future CS 2 students. Future iterations of the curricular materials will also be made available online as they are developed.

Field testing will also allow the MyroC package to continually evolve, and updates to our Launchpad materials will be made accordingly. For example, there are several features left to implement, such as a graphics library and greater control over the various sensors on the Scribbler 2 and fluke dongle.

Overall, we believe that our existing MyroC materials provide a solid base for CS 2 which uses robots in the context of imperative problem solving and C. Through an integrated feedback mechanism, we expect our publicly-available materials will continue to evolve.

8. ACKNOWLEDGMENTS

We would like to thank Dr. John Stone, who provided technical assistance with the interaction of the robots and the network, and Mr. John Hoare of the University of Tennessee at Knoxville, who developed the MyroC++ library. Finally, none of the work over the summer would have been possible without the funding provided by Grinnell College.

9. REFERENCES

[1] Guzdial, Mark, "A Media Computation Course for Non-Majors", *ACM SIGCSE Bulletin*, 35 (3), September 2003, 104-108.

[2] Guzdial, Mark, et al, "Variations on a theme: Role of Media in Motivating Computing Education", *ACM SIGCSE Bulletin/SIGCSE 2010*, 42 (1), March 2010, 66-67.

[3] Hoare, John, Myro-cpp, home page at http://web.eecs.utk.edu/~jhoare/Myro-cpp/Myro-cpp, retrieved August 21, 2011.

[4] Institute for Personal Robots in Education (IPRE), *Main Page*, http://wiki.roboteducation.org/Main_Page, accessed August 21, 2011.

[5] Interim Review Task Force, *Computer Science Curriculum 2008: An Interim Revision of CS 2001*, ACM/IEE, December 2008.

[6] Kumar, Deepak, *Learning Computing with Robots (Uses Python + Scribbler or Scribbler* 2), textbook available at http://wiki.roboteducation.org/Introduction_to_Computer_Science_via_Robots, Fall 2011.

[7] Liberal Arts Computer Science Consortium, "A 2007 Model Curriculum for a Liberal Arts Degree in Computer Science", *Journal on Educational Resources in Computing (JERIC)*, 7 (2), June 2007.

[8] MacLennan, Bruce, *Learning Computing With Robots in C++*, (based on Kumar's Python-based text), textbook available at http://web.eecs.utk.edu/~mclennan/Classes/102/LCRcpp/index.html, January 2011.

[9] Metaxas, Panagiotis, *Applied Ancient Wisdom: Lessons from an experiment in CS2 (and not only)*, slides with Stella Kavavouli, presentation to the Liberal Arts Computer Science Consortium, Holland, MI, July 23, 2011.

[10] Misra, Ananya, Douglas Blank, and Deepak Kumar, "A Music Context for Teaching Introductory Computing", ACM *SIGCSE Bulletin/ ITiCSE 2009*, 41 (3), September 2009, 248-252 .

[11] Myro C Developers, *Myro C: A C Implementation for Scribbler 2 Robots*, https://launchpad.net/myro-c, August 21, 2011.

[12] Myro C++ Developers, *C++ Port of Myro*, https://launchpad.net/myro-c++, August 21, 2011.

[13] Rebelsky, Samuel, Janet Davis, Richard Brown, and Brian Harvey, Whither scheme?: 21st century approaches to scheme in CS1", ACM S*IGCSE Bulletin/SIGCSE 2009*, 41 (1), March 2009, 551-552.

[14] Summet, Jay, et al, "Personalizing CS1 with Robots", *ACM SIGCSE Bulletin/SIGCSE 2009*, 41 (1), March 2009, 433-437.

[15] Walker, Henry M., *CSC 161: Imperative Problem Solving and Data Structures Home Page*, http://www.cs.grinnell.edu/~walker/courses/161.fa11/, Fall, 2011.