Project: *SCAPES*

by Christine Laurendeau

1. Problem Description

The **S**chool of **C**omputer science **A**ssembly **P**rogramming **E**nvironment **S**ystem (*SCAPES*) is a development environment that allows a user to write, compile, and execute basic programs written in the **S**chool of **C**omputer science **A**ssembly **P**rogramming **L**anguage (SCAPL) language. The SCAPL language is a simplified subset of assembly language instructions, and it is described in section 3 of this document. Users of the *SCAPES* system can write programs in the SCAPL language, they can compile those programs into an internal format suitable for execution, and they can run the compiled programs.

The main features of the *SCAPES* system include:

- the ability to write a program in the SCAPL language
- the ability to save a program to a file, or load a program from an existing file
- the ability to compile an existing program; for our system, this means translating the program from SCAPL language instructions into internal data structures that will enable its execution
- the ability to run an existing, compiled program; and
- the ability to manage system preferences

2. Features

The *SCAPES* system supports two categories of users: programmers and system administrators.

Programmer users can create new programs, and edit them. They can load an existing program from a file, or save a program to a file. They can also choose to compile an existing program, where the *SCAPES* system verifies the syntax of the program and translates it from the SCAPL language into an internal format. This internal format will be designed by each team, and it **MUST** consist of **objects** that could potentially be implementable in *any* OO language. For example, these objects could be organized as instructions, operations, and operands. **DO NOT** translate SCAPL instructions into C++ instructions (for example, do not translate SCAPL instruction “dci a” into C++ code “int a;”). You must translate SCAPL instructions into C++ **objects**.

Programmer users can choose to run a program, which means taking an already compiled program and executing its instructions.

System administrator users can do everything that programmer users can do, and they can change system preferences. System preferences include preferred programming language and storage directories.

3. The SCAPL Language

The SCAPL language is a simplified subset of assembly language. Each line of a SCAPL program contains one statement. Each statement corresponds to one instruction from the SCAPL instruction set, as shown in Table 1, and each instruction has a fixed number of operands, as indicated in the table. Any instruction may be preceded by a label, which is used for jumping to that instruction from somewhere else in the program. Labels consist of a string of characters followed by a colon (`:`).

3.1 SCAPL instruction set

The instruction set for the SCAPL language, which your implementation of the *SCAPES* system must fully support, is shown in Table 1. Array variables are declared using the **`dca`** instruction, and array elements are accessed by referencing the array variable preceded by a dollar sign (**`$`**), followed by the addition symbol (**`+`**) and the index value, which can be either an integer variable or a literal. For example, given an array variable **`arr`** and index **`i`**, the corresponding element would be accessed using the expression: **`$arr+i`**

Table 1: SCAPL instruction set

Instruction	Purpose	Operand 1	Operand 2
`dci`	declares an integer variable	name of the variable	not used
`dca`	declares an array variable	name of the variable	maximum size of the array
`rdi`	reads an integer value from the user and stores it in a variable or array element	variable or array element where the value will be stored	not used
`prt`	prints out the value of a variable or array element or literal (may be a string)	variable or array element or literal to be printed	not used
`mov`	copies a value from a source to a destination	the source of the copy (can be a variable or array element or literal)	destination of the copy (must be a variable or array element)
`add`	adds one value to another	one value to be added (can be a variable or array element or literal)	other value to be added, and destination of the addition (must be a variable or array element)
`cmp`	compares two values to test if the first value is greater than, less than, or equal to the second value	first value	second value
`jls`	jump to the specified label if the previous comparison instruction resulted in the first value being less than the second value	destination label	not used
`jmr`	jump to the specified label if the previous comparison instruction resulted in the first value being greater than the second value	destination label	not used
`jeq`	jump to the specified label if the previous comparison instruction resulted in the first value being equal to the second value	destination label	not used
`jmp`	unconditional jump to the specified label	destination label	not used
`#`	indicates that the line is a comment	comment text	
`end`	indicates the end of the program	not used	not used

3.2 SCAPL sample program

For example, the SCAPL program in Figure 1 declares two integer variables **a** and **b**, reads two values from the user, and stores those values in the declared variables. It then compares the two values and prints out the maximum of the two.

```
dci a
dci b
rdi a
rdi b
cmp a b
jmr L1
prt b
jmp L2
L1: print a
L2: end
```

Figure 1: Sample SCAPL program

4. Technical Specifications

The Linux Ubuntu platform, as provided in the official course virtual machine (VM), will be used as the test bed for evaluating the *SCAPES* system. All source code must be written in C++, and it must be designed and implemented at the level of a 3rd year undergraduate Computer Science student, following standard UNIX programming conventions and using advanced OO programming techniques such as polymorphism and design patterns. Only libraries already provided in the course VM may be used.

4.1 User Interface (UI)

The *SCAPES* user interface (UI) will preferably be graphical in nature. A console-based UI will be an acceptable alternative, but will not earn full marks for any of the coding deliverables. In general, user features should be easily navigable, either as menu items and/or pop-up menus. The look-and-feel of the *SCAPES* system should be professional and consistent with commercially available UIs.

4.2 Data Storage

The *SCAPES* system will run on a single host, with all its data stored on that same host. Data storage, both in memory and in persistent storage, must be organized for ease of retrieval and efficient use of space. There should be no duplication of information anywhere. Persistent data may be stored in flat files, or any other mechanism available in the VM provided, including the Qt SqlLite library. The *SCAPES* system must be delivered already configured with a *minimum* of two (2) complete SCAPL-language programs. One of the programs can be basic with at least one (1) loop, and the other program must be more complex with at least two (2) loops (including at least one (1) nested loop), and at least one (1) compound condition. Both programs must be fully documented using inline comments.

4.3 Delivery and Deployment

Delivery and deployment of the *SCAPES* system must be *turnkey*. This means that the user must be able to install all the software for the project with one (1) command, build it with one (1) command, and then launch the project with one (1) command. The user must **not** be required to launch Qt in order to build or launch the project. Projects that do not conform to this requirement will not be graded.

5. Project Work

The project work will be divided into three (3) assignments to be completed by individual students, and two (2) project deliverables to be completed by registered teams. Projects submitted by non-registered teams will not be graded.

5.1 Individual Assignments

All assignments must be completed **individually**. Every assignment must be submitted as a **PDF** document. The documents must be typed and legible, and they must be **professional**, including a cover page, page numbers, as well as section numbers and names. All UML diagrams must be produced using a drawing tool, and not hand-drawn. Documents that do not conform to these specifications will not be graded.

- **Assignment #1:** The work for this assignment will consist of a document specifying:
  - the functional and non-functional requirements for the entire *SCAPES* project
  - the use cases (UML diagrams and table-based descriptions) for a selected subset of the project
- **Assignment #2:** The work for this assignment will consist of a document specifying:
  - the class diagrams and sequence diagrams for a selected subset of the *SCAPES* project
- **Assignment #3:** The work for this assignment will consist of a document specifying:
  - the subsystem decomposition for the entire *SCAPES* project
  - the design patterns used

5.2 Project Deliverables

All project deliverables must be completed by a **registered team**. Coding deliverables must be delivered as a single **tar** file consisting of all source code, data files, and configuration scripts, as well as installation, build, and launch instructions.

NOTE: Coding submissions that cannot be downloaded into a fresh directory and un-tar'd, then successfully built with **a single command** in the course VM, and successfully launched with a single command in the course VM, will **not** be graded.

- **Deliverable #1:** The work submitted for this deliverable will consist of:
  - an in-class presentation of a selected aspect of the *SCAPES* system design
  - a soft copy of the presentation slides, in PDF format
  - the implementation of selected feature(s)
- **Deliverable #2:** The work submitted for this deliverable will consist of:
  - the implementation of selected feature(s)