Semester Project – Part 1 – CSE244 – Spring 2001

Due: Tuesday, March 13, 2001, 12:30pm

The project for the Spring 2001 Semester will focus on developing a lexical analyzer, parser, and syntax-directed translator that can automatically generate a working and compilable graphical user interface when given an input “program”. For our purposes, the program will represent a specification of the makeup and components of the graphical interface, and also indicate all allowable actions as initiated by the mouse. The project will proceed in three parts as overviewed below:

1. Design and develop a CFG that allows programs that represent graphical user interfaces to be written.
2. Design, implement, and test the common CFG (which I will provide to you at a later date) using Lex (Flex) and Yacc (Bison).
3. Design, implement, and test a syntax-directed translator based on part 2 that automatically generates compilable Java code from a program which characterizes the structure and functionality of the interface.

The first two parts of the project must be done by each student individually. The third part of the project is for teams of two students. The first part of the project is divided into two major tasks:

- Develop a CFG for the project that allows arbitrary graphical user-interfaces (their structure and functionalities) to be specified.
- Demonstrate that your CFG works by describing and designing a graphical user interface and illustrating that your CFG can represent its components and functions.

Neither of these tasks require you to develop and/or implement code. This is a design project!!

To provide a background for the semester project, a number of figures that illustrate (and restrict) the capabilities and parameters of a graphical user interface have been provided. These figures are only available in hardcopy, and not on the web! In Figure 1, the major components of an interface are shown, and include a tool banner, screen banner, and menu panel (all of which are optional). The screen consists of a set of graphic functions (which may be empty) that is juxtaposed with a canvas (which is mandatory). The canvas allows drawings and/or other actions related to the interface to occur. The button panel is also required, and must contain, at a minimum, a quit button for exiting the interface. Figure 2 provides a sample user interface for understanding the aforementioned concepts. Notice that the tool is named “Steve’s Drawing Tool” and that the screen is called a “Layer”. Five key panel items (i.e., File, Environ, Shading, Font, and Size) have been provided and when invoked by the mouse result in pull-down menus (see Figure 3). Four graphic functions for drawing rectangles, lines, circles, and inputting text have been defined. Note that polygon is the other available option for a graphic function. Finally, four buttons to Print, Copy, Save, and Quit the interface have been provided.

Figures 3 to 6 indicate the other available components for constructing a user interface, and include: pull-down menus (to allow users to select an action from a list), buttons (to invoke a specific action), dialog boxes (to solicit input from user for performing and action), and browsers.
(to both view information and to allow actions to be selected based on dynamic runtime data). In Figure 3, pull-down and pull-right menus are shown. A pull-down menu is a static list of actions, grouped under a single name (e.g., File in Figure 3). Each of these actions may in turn cause a pull-right menu to be invoked, displaying a second level of static information to the right of the invoking action (Open in Figure 3). You may assume that only two levels (pull-down and pull-right) are available. Pull-down menus (with optional pull-right menus) can be invoked using a command from the menu panel, a button from the button panel, or as part of an action for a selection browser (see Figure 6). Figure 4 indicates that there are two valid buttons. The default button is pre-defined with the string “Quit” for exiting the tool. One or more action buttons may also be given in the button panel, where each is represented by a string.

In Figure 5, three varieties of dialog boxes are shown. In a string dialog box, a message is printed and an input string can be supplied by the user. When the user is satisfied as to their actions, the system defined accept or cancel button deactivates the dialog box. For example, in the sample interface of Figure 2, a string dialog box could be used whenever the graphic function Text is chosen to allow a collection of characters to be input. The message dialog box asks for feedback from the user to inform the user about the consequences of his/her actions, e.g., the message might be Exit without saving file?. The message string in both the string and message dialog boxes is defined as part of a user-interface specification program. Finally, the confirm dialog box is utilized for a final check on a user’s action. The message and buttons in this dialog box are system defined. In Figure 6, two types of browsers with elevator bars are shown. Each browser has a name. The difference between the two browsers is that the view browser only displays information while the select browser displays and then allows the user to select a displayed item and invoke an action (which may be another browser, a pull-down menu, or a dialog box).

The two remaining issues to discuss involve the interdependencies of components and the use of the mouse for selection. First, pull-down menus can occur by an action on the menu panel, as the result of a button being selected, or from the select browser. A dialog box can occur as the action for a menu-item selected from a pull-down (or pull-right) menu, when a button has been chosen, or from the select browser. Finally, a browser can also be used as the action for a menu-item selected from a pull-down (or pull-right) menu, when a button has been chosen, or from the select browser.

The last important issue relates to the interactions of the mouse with the different components of the user interface. The mouse is used to activate each of the required actions. For example, when the mouse cursor is on a key item in the menu panel and one of its three buttons are depressed, then the appropriate action will occur (e.g., a pull-down menu will be displayed). As another example, when the mouse cursor is positioned over a graphic option (or button) and one if its buttons are depressed, then an appropriate action occurs. For a graphic option, perhaps a rectangle or other object can be created. For a button in the button panel, a dialog or browser box, or a pull-down menu may be activated. All of these available options are the responsibility of you, the language designer, when writing the CFG for this problem.

To provide a look at the syntax and semantics of a language for specifying a graphical user interface, consider the following sample program, which is also posted on the web page for CSE244 as a text file:

```plaintext
Graphics Interface Spec;
Begin Main Interface;
   Tool Banner is: "Steve’s Drawing Tool" ;
   Screen Banner is: "Layer 1" ;
```
Panel Items are: ((File Left) Open New),
((Environ Right) Zoom Copy),
((Shading Right) None Gray Black),
((Font Right) Bold Italics Roman),
((Size Center) 10pt 12pt 18pt 24pt);
Graphic Functions are: (Rectangle Center), (Line Center),
(Circle Center), (Text Center);
Buttons are: (Print Left), (Copy Left), (Save Left), (Quit Left);
End Main Interface;

Begin Bind Buttons;
Print = PullDown;
   Items are: (Laser, Line);
End PullDown;
Copy = Dialog < String, Left, "Current Figure to File?" >;
Save = Dialog < Message, Right, "Overwrite Current File?" >;
Quit = Dialog < Confirm, Center >;
End Bind Buttons;

Begin Bind Functions;
Text = Dialog < String, Left, "Input Desired Text" >;
End Bind Functions;

Begin Bind Panel Items;
File.Open = Begin PullRight
   Items are: (Drive, Directory);
   Begin Bind Items;
      Drive = Browser < Select, "Available Drives", El:Center,
          [Left: "A-Drive" "B-Drive" "C-Drive"] >;
      Directory = Browser < Select, "Directories", El:Center,
          [Right: "System" "User" "Libraries"] ;
   End Bind Items;
End PullRight;

Shading.Gray = PullRight;
   Items are: (Light, Medium, Dark);
End PullRight;
End Bind Panel Items;

Begin Bind Browser Items;
Drive.A-Drive = Dialog < Confirm, Center >;
Drive.B-Drive = Dialog < Confirm, Center >;
Drive.C-Drive = Dialog < Confirm, Center >;
Directory.System = Browser < ... etc ... >;
End Bind Browser Items;

End Interface Spec.
Note also that a second sample program is also posted on the course web page.

In the program, words such as Left, Right, Center, Dialog, Panel, etc., are similar to reserved words or keywords in a programming language. On the other hand, File, Open, New, Print, Zoom, etc., are part of the specification of the user interface. Comma, colon, semi-colon, parentheses, square brackets, and angle brackets are used to distinguish portions of the program. A graphical user interface has four major parts, relating to the definition of the screen Main Interface followed by binding begin-end blocks that associate different user interface components (e.g., dialog boxes, pull-right menus, and browsers) with the particular buttons, functions, and panel items of the interface. Briefly, let's review each section.

In the Main Interface section, the basic structure of the interface as shown for the example in Figure 2 must be provided. Tool and screen banner are self-explanatory. The Panel Items definition specifies the different items that occur (e.g., File, Environ, Shading, etc.). For each item, a mouse button is assigned that is used to activate the item (e.g., (File Left)). The remainder of each definition contains the pull-down menu options for that item. The Graphic Functions include the available drawing options. Each function is bound to a specific mouse button for its activation. Buttons are defined in a similar fashion.

The next section of the specification is to Bind Buttons. Each button may be bound to a dialog box, browser, or a pull-down menu. For dialog boxes, the type of the box, the mouse button that is used to select within the box, and the message string that is displayed must be provided, separated by commas and enclosed in angle brackets. The third section binds the graphical function keys in a similar fashion. The fourth section is used to bind the different panel items. In the example, recall that File indicates the name for the items that occur on its pull-down menu. For each of the items (Open and New in this case), a binding to a user interface component may occur. Bindings for File, Open and Shading, Gray have been shown. Nesting occurs in the definition at this point, since it may be necessary to associate another component (e.g., dialog or browser) with the items (e.g., Drive and Directory) as we have shown. When defining a browser, the type of browser must be given and coupled with the name, the mouse button used to activate the elevator bar, and a list of words to display in the browser window. For a select browser, a mouse button must be specified to indicate how these words are selected. This is not necessary for a view browser as is shown in the final browser section.

I want to warn you in advance that there is no guarantee that what I've given you is complete. I may have neglected to demonstrate a combination of user interface component or forgot an important case. There also may be syntax errors that need correction. In summary, you as a compiler designer may need to slightly reformulate portions of the language to achieve consistency and to successfully write your grammar!

To get you started in developing your CFG, I've provided you with a portion of the grammar. Lower-case strings of letters and digits represent non-terminals, while upper-case strings and double-quoted strings represent tokens (terminals). Tokens such as Tool, Banner, Graphic, etc., are the keywords or reserved words in this language. Note that opt is an abbreviation for optional. This grammar segment is also posted on the web page for CSE244 as a text file.

```plaintext
graphical_interface -->  "Graphics" "Interface" "Spec" ";" interface_sections
"End" "Interface" "Spec" ";"

interface_sections -->  main_interface bind_buttons bind_functs
bind_panel_items bind_browsers
```
main_interface -> "Begin" "Main" "Interface" ";"
  opt_headers opt_graphicsfuncts button_panel
"End" "Main" "Interface" ";"

opt_headers -> opt_tool_banner opt_screen_banner opt_menu_panel
opt_tool_banner -> "Tool" "Banner" "is:" string ";" | empty
opt_screen_banner -> "Screen" "Banner" "is:" string ";" | empty
opt_menu_panel -> "Panel" "Items" "are:" menu_item_list ";" | empty
opt_graphicfuncts -> "Graphic" "Functions" "are:" function_list ";" | empty
button_panel -> "Buttons" "are:" opt_button_list "Quit" ";"
function_list -> opt_rect opt_line opt_circle opt_poly opt_text
  opt_rect -> "(" "Rectangle" mouse_action ")" ";" | empty
  opt_line -> "(" "Line" mouse_action ")" ";" | empty
  opt_circle -> "(" "Circle" mouse_action ")" ";" | empty
  opt_poly -> "(" "Poly" mouse_action ")" ";" | empty
  opt_text -> "(" "Text" mouse_action ")" | empty
opt_button_list -> opt_button_list ";" button_spec | ";" button_spec
button_spec -> "(" word mouse_action ");"
menu_item_list -> menu_item_list ";" menu_item | menu_item
menu_item -> "(" word mouse_action "); pull_down_menu ");"
pull_down_menu -> pull_down_menu pull_down_item
  pull_down_item
bind_buttons -> "Begin" "Bind" "Buttons" ";" button_list
"End" "Bind" "Buttons" ";"

button_list -> /* You take it from here!!! */
mouse_action -> "Left" | "Right" | "Center"
empty -> epsilon
word -> "sequence of characters without blanks"
string -> "sequence of characters - may include blanks"

Good luck and please notify me if you find errors or problems!