Corey Fischels

April Kroll

Matt Lockard

Blake Mackey

Joshua Marlow

Requirements Document

for

Graphical IDE

Dr. Bruce Mabis

Fall 2014

CS 478

Project Requirements

Our group has been tasked to create an Integrated Development Environment (IDE) for an Introductory Programming Language. The language will be a graphical language. The development language, operating system, and target language (see also: Testing) are our choice within reason.

* **Environment**

The environment will be similar to modern IDEs (such as Visual Studio, jGrasp, Eclipse, to name a few). It will include a main window for program development with relevant menu items/buttons across the top of the screen. Along the right side of the main development window will be a list of currently known objects (modules, programs, etc.) This list will be kept as a directory structure. These windows can be resized by the user.

The environment will include additional pop out windows for actions like saving a program, executing a program, etc. The language definition will also dictate new pop out windows and when they are required.

* **User Interaction**

The user will accomplish the task of developing a program not by writing in the general sense, but instead via a set of templates. A user will indicate their template choice by choosing the template through a hot key (assigned keyboard shortcut). E.g. an assignment statement template is opened by “Alt-A”.

Every template will have text to display as well as fields for user supplied information. I.e., the statement will be syntactically correct with spacing for additional information that is supplied by the user.

In the event of an assignment, the user must supply the expression to calculate and the variable to receive the new value. For example: the assignment may be displayed to the user as:

Calculate [] and save the result as []

The user will click on the appropriate box, which will open a window to allow the user to supply the necessary information. In this example, the first [] requires an expression and the second [] requires a variable name.

Users will be able to minimize/maximize pop out windows to make more additions and/or necessary changes. Windows will also have the ability to be recursive; an IF statement can contain and IF statement, which can contain an additional IF statement, etc.

The available templates and elements within each template are defined within the language currently in use.

* **Languages Defined**

A language will be defined for the user. Language definitions will be in a file in a known directory. Templates will consist of an appropriate hot key, name, and display template. For example:

A, assignment, Calculate [] and save the result as []

Other elements such as conditions and expressions, etc. will most likely be stored in Backus-Naur Form notation.

* **Testing**

The user can test their program by choosing the appropriate menu item (Test).

The user's program will then be converted to an actual programming language (developer’s choice), compiled, and executed. The input/output for the user program will be managed by the IDE.

* **File I/O**

Our IDE will read in a language definition, and the user will interact with said IDE to create, test, load, save, etc. their programs.

The IDE will not read in .txt files. The user will be able to save what they have created on permanent storage. The user will be able to save and return to the file the next time the user opens the IDE (recent programs).

The file output will be in a format readable by code editors, with file format extension name YOLO.

* **Language Files**

The language files will describe different language syntax. The lower levels of language will include simple (I/O only); additional languages will be more and more complex and include additional syntax as necessary.

Each file will consist of language elements – templates, additional definitions – and rules for converting the language into an actual programming language for execution.

Each file will represent different levels of the language. A language file may have nothing but input and output files so that the user can’t do anything else. The next level could have I/O & IF statements.

Our language will support the following syntactic elements:

INPUT, OUTPUT, ASSIGNMENT, IF, WHILE, ELSE, REPEAT, FOR, UNTIL, SWITCH, DO, NUMERIC, STRING

* Variable declaration will be handled through ASSIGNMENT

- NUMERIC and STRING data types

* Decision making, program control, and looping structures:

- IF, WHILE, ELSE, REPEAT, FOR, UNTIL, SWITCH, DO

When an object is created, it will become one of a kind. Classes, polymorphism, and dynamic binding will not be explored or used as they are outside the scope of an introductory programming course.

Semicolons will not be used. Block structure will not be an issue. The IDE is taking away complexity and syntactic elements so that it is easier to understand.

**In summary, the language defines what functionality / statement types it will have and if named method/function/subroutine can be or need to be included.**

* **Arithmetic & Logic Capabilities**

Declaration of variables will require the user to set a limit, set a variable name, and set what the variable can be equal to.

Operator precedence will follow PEMDAS: Parentheses, Exponents, Multiplication, Division, Addition, and Subtraction.

**Example:**

User types XYZ. XYZ must be declared. The user declares it as NUMERIC. Each successive use will mean it is the NUMERIC data type.

**Example of arithmetic routine:**

*A square subroutine/procedure/method, for example, X = Y \* Y*

ASSIGNMENT X

ASSIGNMENT Y = 2

X = Y \* Y

OUTPUT X

* **Data Types**

**Numeric** data type – all operands will be converted to the float data type.

**String** data type – will consist of one or more characters of alphanumeric text.

The environment will not handle arrays or other complex data structures.

* **User Files**

The user may save/load their creations anywhere available but the current directory is displayed for the user to simply click their choice.

The IDE will have functionality in place to prevent the user from accidentally closing their program without saving or opening a new program without saving (E.g.: ‘Do you want to exit without saving? Yes No Cancel’ and the like.) The user will be able to create a program and save it in a saved directory which displays a list of the user’s programs.

* **Visual design and overall aesthetic**

The environment will mainly focus on a blank window that the user utilizes to create the program in. The user will name their program upon creation (the 'New' command will open a blank window and allow them to name the program). The IDE will set up generic default names if the user does not input a name (e.g. Program1, Program2, etc.) A cursor will be positioned in this window which allows the user to traverse through their code and code elements.

The software is meant to be useable in an intro programming course with little knowledge about coding. It will be able to accept input and produce output in a simple and understandable way for the end user. The visual environment will be kept simple and intuitive for the user.

**Example:**

**I**f [ ] then [ ].

The user will have the ability to fill in the boxes.

We will treat each GUI window as a separate module. The names of programs/objects/modules we save each time are complete, self-contained modules.

The GUI will be collapsible - it will be designed to collapse after the user has input their values. Each line will collapse into a smaller line that only includes details.

User-facing programming and GUI complexity will be limited to general introductory programming keywords (IF THEN ELSE SWITCH WHILE DO WHILE, etc.)

* **Error Handling**

User errors will be at a minimum; the IDE will force the user’s input to be converted into a more proper syntactic structure. Logic errors and runtime errors will not be handled, but the software will report potential warnings and errors as best as possible, and if need be, terminate the user’s running program and report final errors and exceptions.

Infinite loops should not be an issue; they are not detected in regular languages – by the time you get to loops in introductory programs.

Unreachable code should also not be an issue. If the condition can never be false then the ELSE portion can never be reached.

The user will simply be introduced to potential errors and how they interact. A generic error message box will display to alert the user that their program encountered one or more irrecoverable errors.

* **Documentation (Help, Support, Naming Conventions)**

The environment will be intuitive; the user must know enough to start inserting program elements. The IDE will provide a few example programs to help the user get started.

The user must know enough to start inserting program elements (e.g.: Create a statement to print ‘hello world’ as the classic programming assignment. It may tell you if it handles error types [syntactic errors and the like].)

Beyond simple help information, documentation, and tooltips, the group expects a textbook / class / instructor / workshop to explain the environment in greater detail. The environment will have statement types with some simple context-sensitive messages. Tutorials (starter programs) and handholding (tooltips, help buttons, and pop ups) will be used to facilitate program creation. These simple conventions will be in place to easily guide the user through the process.

**Additional requirements:**

* **Space constraints**

The maximum allotted environment file length will be 200 lines long.

* **Compiler/Parser constraints**

The compiler and the parser may both be open source.

The compiled program will load in a console window.

* **Software Implementation constraints**

The software may be submitted to the client as either source code or as an executable.