CS1632 Deliverable 6:

Befunge Testing Strategy

Authors: Raj Patel, Chris Good

Quality of Befunge IDE

With the recent surge in the usage of the Befunge programming language, the market is ready for a integrated development environment to handle the language. We at C-137 plan on fulfilling this need with our Befunge IDE. To ensure that our startup can become the industry leader in Befunge IDEs, we need to design our IDE to have a high level of quality and to make it comparable to existing IDEs for other programming languages. With these clear goals in mind, here is the quality report following the Red-Yellow-Green protocol for the existing iteration of the Befunge IDE:

Color	**Subsystem/Subset of Functionality**	**Description**
Yellow	System Launch	IDE requires the compiling of java files and then the running of java class files on command line. Launchable, but not user friendly.
Red	User Interface	IDE is not at all user friendly, with no toolbar and no line numbers. Only one perspective, no file/project explorer, or other common IDE features
Red	File I/O	Absolutely no capability to save befunge programs or open existing befunge programs, makes IDE entirely pointless.

Yellow	Debugger	Should have separate debug perspective with various options. No ability to pause execution, step through execution, or stop execution. However, there is output and the stack is shown.
Green	System Performance	IDE takes some time to run programs, but that is inherent to Befunge's stack based design.
Green	System Administration	Code base needs to be cleaned up a bit, some functions are not used. Otherwise code and system maintenance is fine.

Areas of Concern and Improvement

Aside from the quality issues touched upon in the previous section, there are few areas of concern regarding the Befunge IDE as it currently exists. The only areas of concern revolve around several areas of improvement that could be made to the IDE to make it far more desirable to consumers. The following enhancement report reflects the most significant of the enhancements to be made:

SUMMARY: Line Numbers Enhancement

DESCRIPTION: Addition of line numbers in code text editor section of IDE.

CURRENT BEHAVIOR: Text editor is a blank text field.

ENHANCED BEHAVIOR: Text editor has numbers next to each new line in the text field.

SUMMARY: Debug Toolbar Enhancement

DESCRIPTION: Inclusion of a dedicated debug toolbar with buttons to start, step through, continuously run, and stop the program.

CURRENT BEHAVIOR: IDE has a run button to run the program.

ENHANCED BEHAVIOR: IDE will have a dedicated toolbar in the top section of the GUI that has buttons to run the program, stop/pause the program, and step through each instruction.

SUMMARY: Undo/Redo Feature Enhancement

DESCRIPTION: Include capability for user to undo and then redo actions in the IDE, such as deletions.

CURRENT BEHAVIOR: All actions inside the IDE are final.

ENHANCED BEHAVIOR: There is a toolbar or file menu at the top of the IDE GUI that has buttons from which user can undo their actions in case they make a mistake in the code or delete all of their code, and redo actions which they undo.

SUMMARY: Find Feature Enhancement

DESCRIPTION: Addition of feature in IDE to search through code by use of a ‘Find’ dialogue box. Feature will be able to search for strings with specified parameters like “wrap search”, “case sensitive”, or “whole word only”.

CURRENT BEHAVIOR: Users must search for patterns and strings manually.

ENHANCED BEHAVIOR: Users can programmatically find strings and patterns with the find feature using a separate find dialogue window.

SUMMARY: Real-Time Static Code Analysis Enhancement

DESCRIPTION: Addition of feature to IDE that statically analyzes code in real time to warn

users of potential performance/style issues and indicate whether there will be an error in program execution. **CURRENT BEHAVIOR**: IDE gives no indication of program quality or potential issues, user just runs code as is and observes result. **ENHANCED BEHAVIOR**: IDE gives indications, such as underlining or symbols next to the line, warning the user that the code has some kind of style or potential execution issue.
SUMMARY: IDE File I/O Enhancement **DESCRIPTION**: IDE is given feature to open and save Befunge files or files related to the Befunge language by way of a file menu or toolbar. **CURRENT BEHAVIOR**: User must enter in code completely manually, and this code goes away once the IDE is closed. **ENHANCED BEHAVIOR**: User can press button in menu/toolbar at the top of the IDE GUI to open a previously created Befunge file. Similarly, user can then save any file created or altered in the Befunge IDE into their file system by way of another button.

Testing Strategy

The first kind of tests we should implement are User Interface (UI) tests. These tests will simply check if the system is user-friendly and program functions are properly executed. This includes the system launch; is it relatively easy to open? Tests for system launch would be done manually. Smoke testing the IDE would test the primary functions by running a simple “Hello, World” program (`64+"!dlroW ,olleH">:#,_@`), which can be done automatedly. There should not be too much time spend on these tests; they can all be carried out by one person.

Next to be performed are service tests. These are necessary to ensure that components (i.e. the debugger) work correctly, as well as to test performance. Property-based testing would be focused on the compiler and debugger, specifically testing that these components appropriately handle type errors, incorrect use of library APIs, errors in logic, etc. Errors in logic is a crucial defect category that should be tested for.

Befunge-93 is a language that is, by design, extremely difficult to compile. This is because of two main features of the language: it is self-modifying (the p instruction can write new instructions during compilation), and it is multidimensional (meaning the same instruction can be executed in four different contexts). As such, these parameters should be tested in combinatorial manner. This would need to be done manually by at least one tester. This test could also utilize VisualVM to test speed of each method used in the IDE program. Using a profiler in VisualVM can detect hot spots (long run times) in the programs that signify areas of code that may need cleaning up. Performance can also be tested in a combinatorial manner, using parameters such as different operating systems and amounts of RAM. The main point of an IDE is to write code, execute them, and see the results, so these tests are where the most time and effort should be invested.

Finally, the unit tests will need to be conducted. These will simply test to see if both results and stack display correct values. It will also check if the GUI elements individually function correctly. These tests can be automated, and thus do not require a physical tester. Unit tests will be minimal due to the limited functionality of the IDE, and therefore should require a relatively short amount of time.

Testing this Befunge-93 will require a relatively small management team. This team will consist of us, the Directors of Quality, the IDE's development team, a marketing team to promote the product, and a team of at least 2 testers. These testers should have least one year

of experience coding with the Befunge language, and should highly experienced (at least 3 years) in Java and C; the IDE is written in Java and the Befunge compiler bf2c compiles each instruction to a snippet of C code. Testing will primarily involve performance testing, with focus on UI and service tests (compiler). We will need at least one person to conduct tests on the compiler, the most crucial component. We would need only one other person to write test programs and automated unit and system tests. Specific functions (running, stack tracing, output) can be automatedly tested using unit tests, as mentioned earlier.

We will require outside help in certain areas of the testing process. We will bring in at least one person who has experience in compiler design, as he may be needed as a consultant for the property-based tests. It may also be useful to bring in someone with a strong enough technical background to know the basic functions of your typical IDE; this person would only be used for black-box testing.

Exploratory testing revealed that there is currently very limited functionality. There are several ways this IDE can be improved, as listed in the enhancements above. The IDE should implement features such as line numbers, the ability to open and save befunge files, undo/redo, the ability to step through execution one instruction at a time, searching, etc.

Works Cited

Pressey, Chris. "Befunge-93.markdown." Befunge-93.markdown | Cat's Eye Technologies. Cat's Eye Technologies, Aug. 2012. Web. 16 Apr. 2017

https://esolangs.org/wiki/Befunge. "Befunge." Befunge - Esolang. N.p., n.d. Web. 16 Apr. 2017.