CS4120/4121/5120/5121—Spring 2019
Programming Assignment 7
Object-Oriented Features
Due: Wednesday, May 15, 4:30PM

This assignment requires to you add object-oriented features, completing your compiler. The new, extended language is called Xi++, which adds classes and inheritance to Xi. The differences between Xi++ and Xi are described in Xi++ Language Specification.

The due date for this assignment is a hard deadline. Note that the project is due at 4:30PM rather than at midnight. We recommend aiming to get the project done at least a couple of days ahead of time.

0 Changes

- None yet; watch this space.

1 Instructions

1.1 Grading

Solutions will be graded on documentation and design, completeness of the implementation, correctness, and style. 10% of the score is allocated to whether bugs in past assignments have been fixed.

1.2 Partners

You will work in a group of 3–4 students for this assignment. This should be the same group as in the last assignment. If not, please discuss with the course staff.

Remember that the course staff is happy to help with problems you run into. For help, read all Piazza posts and ask questions (that have not already been addressed), attend office hours, or meet with any course staff member either at the prearranged office hour time or at a mutually satisfactory time you arrange.

1.3 Package names

Please ensure that all Java code you submit is contained within a package (or similar, for other languages) whose name contains the NetID of at least one of your group members. Subpackages under this package are allowed and strongly encouraged. They can be named however you would like.

2 Design overview document

We expect your group to submit an overview document. The Overview Document Specification outlines our expectations.

3 Building on previous programming assignments

As before, you are building upon your work from Programming Assignment 6. The protocol is the same as in prior assignments: you are required to develop and implement tests that expose any problems with your implementation, and then fix the problems. For PA7, the correctness and completeness of your work for previous assignments will count for much more than it has earlier—it will count about as much as your work on the new features for PA7. If you had serious problems with the correctness of your PA6 submission, it is prudent to focus on ensuring that your compiler works correctly for Xi (without optimizations turned on) before tackling the new object-oriented features.

4 Version control

As in the last assignment, you must submit file `pa7.log` that lists the commit history from your group since your last submission.

5 QtXi: A GUI library for Xi++

To help you do interesting things with your Xi++ compilers, we have decided to provide you with a basic GUI library. This way you can interact better with your programs, and perhaps draw some nice graphics. In particular, we provide an Xi++ interface to a subset of the cross-platform Qt library to Xi++ programs.

The source code for the library is posted on the course website. Additionally, the VM has been updated with the library provided and prebuilt in the `QtXi` directory, along with the runtime library from PA5 provided and prebuilt in the `runtime` directory.

Under `QtXi/examples`, you'll find several examples of Xi++ code that use this library, along with corresponding assembly code for each Xi++ example that you can use to get started right away and as guidance when adding Xi++ to your compiler.

To link with this library, use `linkqt.sh` under `QtXi`, which is similar to `linkxi.sh` under `runtime` but also links in the QtXi library and its dependencies.

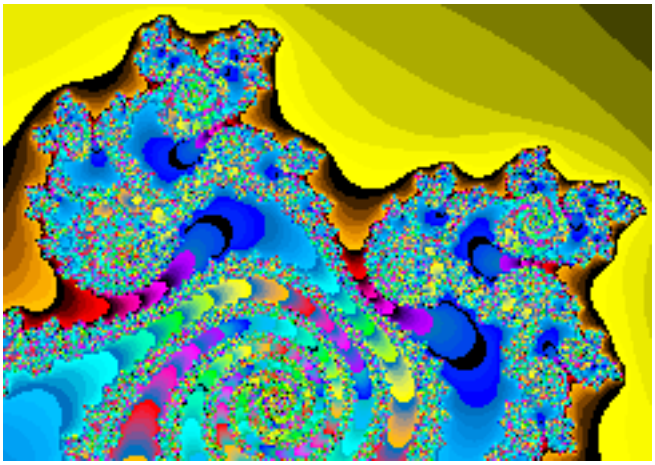
To run programs linked against this library, you will need an X server running on your host machine. If you are running Linux on your host machine, you are likely already running one. On macOS, the standard choice is XQuartz. On Windows, try Xming. `vagrant ssh` is preconfigured to enable X forwarding over SSH, so X programs executed via such an SSH session should automatically open on the host X server. A good way to test this is to invoke `xcalc` in an SSH session on the VM. Once `xcalc` is working, you should be able to link and run one of the example programs under `QtXi/examples`.

This library has been tested using our compiler, and was used successfully in the last iteration of the course, but bugs are always possible! The staff will appreciate any feedback.

6 Example programs

Some example programs are available for testing your compiler:

- **animate** and **animate-fancy**: Bouncing ball demos that leverage various Xi++ and QtXi features.
- **layouts**, **onewidget**, and **widgetevents**: Some other examples of interacting with QtXi.
- **mandelbrot**: A graphical Mandelbrot set explorer using QtXi.



- Additionally, we plan to add some Xi++ test cases to the released `xth` tests soon. Keep an eye on Piazza for this announcement.

7 Language extension

We expect you to design, implement, **document**, and test some new, small feature for Xi++.

Some examples of features you might add are the following, roughly in order of difficulty.

- Allow constants to be defined in interface and source files, integrated with constant folding (relatively easy).
- Support more powerful multiple-assignment syntax, with multiple expressions appearing on the right-hand side (relatively easy).
- Java-style `break` and `continue` that can jump to named labels.
- Make Xi type-safe by detecting uninitialized variables.
- Multiple inheritance with sparse dispatch vectors (relatively challenging).

But you don't have to implement one of these choices—feel free to be creative! It doesn't have to be a big feature, and smaller project groups should feel comfortable in implementing smaller features. You will get a bit more credit for attempting more complex features, but you will also get credit for building a rock-solid implementation of your feature and for documenting it clearly. So don't bite off more than necessary.

It is especially important that your new feature be covered well in your overview document so that we can give you credit for your work. Like the other components of the assignment, your document should at least include a strong specification for the feature, provide some examples of its use, discuss the design and implementation, and indicate where to find the relevant source code, along with any other information you think is pertinent. The Overview Document Specification is a great resource for working on this.

Your extended language should be backward-compatible with the Xi++ spec so that correct programs written according to that spec still work. Furthermore, keep in mind that the Xi++ spec is itself backward-compatible with the Xi spec, so your compiler should also handle Xi programs successfully.

8 Command-line interface

The command-line syntax is largely as defined in the previous assignment. Because Xi++ is backward-compatible with Xi, your `xic` does not need any new command line options or features to differentiate between them. However, your compiler should also support a `-noextension` option to turn off your language extension and just support Xi++.

9 Build script

Your build script `xic-build` from previous programming assignments should remain available. The expected behaviors of the build script are as defined in the previous assignment. Problems within the build script from previous submissions should be fixed.

10 Test harness

`xth` has been updated to contain test cases for this assignment and to support testing language extension.

To update `xth`, run the `update` script in the `xth` directory on the VM.

A general form for the `xth` command-line invocation is as follows:

```
xth [options] <test-script>
```

For the full list of currently available options, invoke `xth`.

The best way to run `xth` with the provided test cases is from the home directory of the VM, using the following form of command:

```
xth -compilerpath <xicpath> -testpath <tp> -workpath <wp> <xthScript>
```

where

- `<xicpath>` is the path to the directory containing your build script and command-line interface.
- `<tp>` is of the form `xth/tests/pa#/`, where `#` is the programming assignment number.

- <wp> is preferably a fresh, nonexistent directory such as `shared/xthout`.
- <xthScript> is of the form `xth/tests/pa#/xthScript`, where # is the programming assignment number.

An `xth` test script specifies a number of test cases to run. Directory `xth/tests/pa7` will contain a sample test script (`xthScript`), along with several test cases. `xthScript` also lists the syntax of an `xth` test script.

`xth` was used successfully in the last iteration of the course, but bugs are always possible. Please report errors, request additional features, or give feedback on Piazza.

11 Best Compiler Competition

The compilers from various groups will be compared in the 2019 CS 4121/5121 Compiler Bakeoff! The winning compiler project, based on the quality and correctness of generated code, will gain bragging rights, receive handsome wood plaques, and be immortalized on the course web page.

Functional and benchmark test cases

For fun and good karma, you are encouraged to submit programs that you think are good functional tests or good performance benchmarks. We encourage each group to submit up to five functional test cases and up to three benchmark test cases.

Each test case must be a valid Xi++ source file. A compiler `c` passes a test `t` if and only if

- `c` successfully compiles `t` into an assembly file `a`,
- assembling and linking `a` against the standard Xi library results in a runnable program `o`, and
- when executed, `o` terminates with exit code 0 **within 3 seconds**; i.e., it terminates normally, and not as a result of an assertion failing or an array-out-of-bounds violation.

All test cases must

- be UTF-8 encoded files, using the Unix standard of LF as a line separator (also known informally as `\n`)
- be valid Xi++ programs, according to the standard specifications (i.e., the Xi++ language specification),
- not read input,
- contain at most 20 lines of code, excluding comments, and
- contain no line longer than 80 characters.

These test cases will also be run against the compilers of other groups, and groups will receive good karma for generating the fastest code for submitted test cases, or for submitting test cases that expose bugs in other compilers. An ideal functionality test case will break exactly half the other compiler projects.

12 Submission

You should submit these items on CMS:

- `overview.txt/pdf`: Your overview document for the assignment. This file should contain your names, your NetIDs, all known issues you have with your implementation, and the names of anyone you have discussed the homework with. It should also include descriptions of any extensions you implemented.
- A `zip` file containing these items:
  - *Source code*: You should include everything required to compile and run the project. *We require that* `xic` *and* `xic-build` *are at the root of the* `zip` *file.*
    If you use a lexer generator, please include the lexer input file, e.g., `*.flex`. Please include your parser generator input file, e.g., `*.cup`.
    Your `xic-build` should use these files to generate source code, and you should not submit the corresponding generated source code files (e.g. `*.java`). Do not submit compiled versions of your own code (submitting precompiled libraries is OK).
  - *Tests*: You should include all your test cases and test code that you used to test your program. Be sure to mention where these files are in your overview document. Do not submit instructor tests or xth.
  - *Libraries*: Your build process must not download anything from the internet. If your code depends on any third-party libraries, they must be included in the submission.
    Include precompiled libraries (e.g. JAR files) when feasible, especially for large libraries. For smaller libraries, such as the release code from PA2 and PA4, it often makes sense to include the source code directly, but be sure to make clear what is library code, e.g. by package name. Do not make global environment changes in your xic-build script.

  Do not include any derived, IDE, or SCM-related files or directories such as `.class`, `.jar` `.classpath`, `.project`, `.git`, and `.gitignore`, unless they are precompiled versions of third party libraries.
  It is strongly encouraged that you use the `zip` CLI tool on a *nix platform, such as the course VM. Do not use Archive Utility or Finder on macOS as they include extraneous dotfiles, and do not use a Windows tool which does not maintain the executable bit of your `xic` and `xic-build`.
- `pa7.log`: A dump of your commit log since your last submission from the version control system of your choice.

13 Advice

It will be tempting to walk sequentially through each of the phases of the compiler, upgrading each to the new languages features. That strategy will probably not make full use of your team's implementation bandwidth. It will probably be better to plan out a strategy in which different group members can work on different compiler phases simultaneously. For example, you don't need to have a lexer or parser working in order to hand-craft a few ASTs that can be used as test cases for later compiler phases.

Also, do not forget to create negative test cases such as source files with type errors, parse errors,

etc. to ensure your compiler does not accept bad source programs.