Appendices

1 References

- Aho, Alfred, et al. *Compilers: Principles, Techniques, and Tools*. Pearson, 1986
- Pivak, Ruslan. *Let's Build A Simple Interpreter*, `https://ruslanspivak.com/lsbasi-part1/`

2 Appendix A

2.1 List of complaints regarding using pseudocode I gathered from my friends

- *"I hate having to check the code myself, it's a waste of time."*
- *"The online tool we use is helpful, but the lack of error detection annoys me to the point I ignore the tasks because I don't have the time to check every bit of code."*
- *"Pseudocode is easy, it takes a bit of practice and you can find bugs quite fast on your own, you can live without a dedicated tool."*

2.2 Transcript of conversation with my friend Jan who was most interested in my idea (translated to English)

- Igor Krzywda (me): So I have this idea to build interpreter for the pseudocode, what do you think?
- Jan Gumulak (JG): Don't we already have the tool for this, the one on the website?
- me: Yeah, I know, but I want my version to detect errors so that we don't have to waste time looking for them manually. Would you use something like this?
- JG: Of course, I hate having to look for them, especially when during lesson everyone has some different error and we have to discuss every one before we move on. It takes ages!
- me: Ok, then, when I have something I'll send you link to my repo.
- JG: Sure, I'll check it out.

2.3 Transcript of the first meeting with my advisor - my computer science teacher

- Igor Krzywda (me): Hello, I talked with others about my idea to make an interpreter for pseudocode which would detect errors in the code. Everyone seemed to be positive about the idea, so I did some deeper research and got general outline of what I'm going to do.
- Przemysław Pobiedziński (PP): Okay, that sounds good. What is the general outline?
- me: From what I read, the intepretation goes something like this: the source code in text needs to be divided into tokens, which are some identifier and a piece of data. Then all tokens need to be put into a tree so that in the end I can traverse it and do operations based on what is in any given node.
- PP: Seems correct, do you have resources to learn details from? Because you described the idea very generally.
- me: Yes, there are a lot of resources online, so I think I will be good on that front. Oh, I even managed to borrow the dragon book on compilers, it is supposed to be a really good textbook.

- PP: Sounds good. What do you want to write it in?
- me: I want to make it in plain C.
- PP: Are you sure? You are already taking on rather complex project, if you use C, you will have to write your own data structures and you will have to manage memory yourself. It will distract you from the main problem and will slow you down.
- me: I know, but I want to learn as much as possible, so I thought that writing in more low-level language would be a good idea.
- PP: I understand you, but you have to keep in mind that you don't have unlimited time so you take a big risk doing that. Why don't you use C++, it is a middle ground - you will have the access to programming on a low level of abstraction, but if you feel that it is getting too hard, you can use built-in containers. Not to mention that C++ handles strings, which I think in your case would be an immense benefit.
- me: Hmm... I think that you may be right. I will try to start writing in C, but if I feel that it is too hard for me, I will switch to C++.
- PP: Okay, do you have anything else you want to consult with me?
- me: No I don't think I have. Thank you for your time.
- PP: Sure.

3 Appendix B

Below is a list of tests done to test success criteria, source code is in directory `product/ibpci/examples/tests`. The tests have also been done in the video.

- `data_types_demo.ib` - checking that all data structures, as well as I/O
- `bubble_sort.ib` - testing loops and conditionals
- `binary_search.ib` - testing recursion
- `fizzbuzz.ib` - testing expressions in conditions
- `error_demos/*.ib` - testing error detection, where filename describes expected error

4 Appendix C

4.1 Client feedback - general thoughts

- *"I like it. I intentionally put some errors and it detected them, even gave the line number!"*
- *"Everything worked great, but I wish you made a binary, it took me a lot of time to set everything up, though I feel that it was a good investment considering that we will be doing a lot of pseudocode"*
- *"It took a bit of figuring out with using command prompt, but I used it and it was really nice to see errors pop up. Think about making a GUI version, because not a lot of people will use it on Windows otherwise."*
- *"Man, I tried but didn't have the time to set everything up to compile the project. Make a Windows binary and I'll surely try it"*
- *"It worked quite well, but when I tried to compute a factorial recursively it crashed completely, something doesn't work with that"*

4.2 Client feedback - conversation with Jan

- Igor Krzywda (me): So what do you think?
- Jan Gumulak (JG): I like it, I haven't yet written code that crashed it, so I guess that's not bad.
- me: So I guess you didn't tinker a lot with recursion yet. What about errors, are they useful?
- JG: Oh, they are great, especially syntax ones. Run-time errors sometimes give strange descriptions, one time I got comparison between null and string.
- me: Hmm... I'll look into that, thank you. Have you tried the flags?
- JG: Yes, but I don't think that they are a nice quirk. The one displaying tokens is interesting, same with the tree, but if someone was to learn from that, you need to make some documentation. The stack flag, on the other hand, was quite useful, I was tracking the variables in the functions. You could make it a bit cleaner, though, it is not very readable.
- me: I'll work on that, thanks.
- JG: No problem.

4.3 Final meeting with advisor - transcript

- Igor Krzywda (me): Hello, I sent you the source of the project and success criteria. Could you try it and compare it with success criteria?
- Przemysław Pobiedziński (PP): Sure, I will try it and get back to you.
- me: Okay, thank you.
- PP: I tried the test programs you included and everything works like in the film, so the most of the success criteria you set check out. However, there is no cheat-sheet you gave in the list. Also, I tried some recursive algorithms, and some of them worked, but when I tried returning an expression containing function call, I got an error, so you need to work on that. Over all I think that everything in terms of goal you set yourself is fine.
- me: Thank you, do you have any extra suggestions or problems you came across outside the success criteria?
- PP: Yes, I think that you shuld make a graphical interface for others to be able to use more easily. The program is good, but it is not very user-friendly, if you made a GUI it would be a great tool to use during lessons, but at this moment it is too hard for someone with no experence to use, not to mention compiling it. Another thing I would add are generic collections, notice that in the exercises you need to use a collection, not a stack or queue.
- me: Thank you for your feedback, I will keep it in mind when I will be playing with the program.