Hello and welcome to this video presentation! In my CPS 2000 course, I was tasked with creating a compiler for the PixArDis company, that could take PixArLang code and generate PixIR code, which could then be run on the PAD2000 VM Simulator. I'm happy to report that the project was a success! The simple pattern currently being displayed on the screen was once a PixArLang program that was translated into PixIR code using the constructed compiler.

(Spiral (Program 4))

Additionally, construction of such Compiler was portioned through the following sections:

1. Task 1 – Creation of a Table-driven Lexer which is responsible for tokenising the PixArLang Input code.

2. Task 2 - Hand-crafting an LL(k) parser which is responsible to ensure that the passed code is syntactically correct

3. Task 3 – Generating XML representative code of the passed input code

4. Task 4 – Initiating a Semantic Analysis Pass, to ensure that the input code is semantically correct

5. Task 5 – Generating PixIR representative code of the passed input code which will be fed to the PAD2000 simulator.

To demonstrate the capabilities of the constructed compiler, let's take a look at a few example programs.

When we pass this program through our compiler, it runs smoothly without any errors, successfully completing the parsing stage and generating the corresponding XML and PixIR code.

(Fibonacci (Program 5))

To test the error-checking capabilities of our compiler, let's modify the program by changing a variable name and observe if the compiler detects the semantic error.

(startx->start)

When we introduced a change to the variable name, the compiler detected and presented a semantic error, which confirms its error-checking capabilities.

Now let us try to include a special symbol which is not in the PixArLang language and check whether a proper error is issued.

(startx->start%)

Success, the compiler correctly labelled this token as invalid, demonstrating its ability to identify errors in the input code.

Now, let's test the comment functionality. Let's add a valid comment and see if it works.

(/\*test\*/)

Success! The comment worked perfectly. But what if we remove the ending tag for this comment?

(/\*test\*)

Nice, the compiler correctly identifies the open comment as an invalid comment.

Next, let's test for syntax errors. Let's remove a few commands and see which type of error emerges.

(remove =0 (startX))

Success, And the compiler catches a variable declaration error.

let's also test for type mismatch errors shall we. We will assign a float to an integer variable and observe the result.

(assign a float(5.5) to startX)

We can observe that the XML pass was generated, but the PixIR code was not generated as semantically, an integer value cannot be assigned a float type.

Next, let us remove all the code and check whether the constructed compiler handles well an empty file.

(Remove all the code)

As you can see, the compiler was able to handle the empty file and did not raise any errors.

So, overall, we've seen that the constructed compiler is able to handle a variety of input code and is capable of catching and correctly identifying different errors in the code. This is an important feature of any compiler, and I am pleased to see that my implementation meets these requirements.

For the remaining portion of this video, I will showcase the ability of my constructed compiler by running it with various valid code snippets. As a viewer, you can sit back, relax, and watch the beautiful patterns generated on the PAD2000 display enfold.

Let's dive in and see what our compiler can do!

(test program)

Thank you for your time and attention!