Radium Language

Radium is a low-level programming language built to be simplistic and secure. Inspired by the functionality of Rust and C and the simplicity of Go, Python, and many Markdown features, Radium is meant for developers of any level, with little to no learning curve.

Introduction:

To start, create a new file with the extension .rad. This is a Radium language file that could be compiled using our native CLI or another 3rd party compiler that supports Radium.

After creating the file, specify the packages you want in your project. For this example, we'll install the basic Radium package.

```
package radium
```

Once you have specified your packages, you can begin writing your code. You should put your functions after specifying your packages, as most compilers read code from top to bottom, and errors could be caused if the functions are not compiled before being stated. Let's define our main function.

```
package radium

func main(print)
```

We are creating the primary function. Inside our main function is the named "print" value, which acts as a variable only available in our function. Once our function is specified, we must program precisely what we want it to do. Let's make it print what the value print contains.

```
package radium

func main(print)
      print(print)
```

We just added the print statement to our main function, which will print the value of our function variable, print. To finalize and close off our function, we need to add **- -** to the end, which tells our compiler that is all we want our function to do. Let's add it.

```
package radium

func main(print)
      print(print)
--
```

The function is specified, and the packages are installed, but now we must execute our function. To do that, we specify this at the end of our program.

```
package radium

func main(print)
      print(print)
--

main("Hello World!")
```

This is printing the words "Hello World!" using our function. We specify words by putting them inside two quotation marks, which is called a string. We put them between quotation marks because the main function of specifying words inside our statements is to check variables. By specifying that these are words, we are overriding the main function of our statements, which is to check variables, not using what is inside the quotation marks.

Now, when executing this with our native CLI by using radium run (file) or a third-party compiler, it should print the words "Hello World!" But that's a bit boring, so let's move on to creating something less boring.

Inputs:

We're going to make our program relay user inputs. To start, make sure you still have your .rad file. It should contain this.

```
package radium

func main(print)
      print(print)
--

main("Hello World!")
```

To make our program capture inputs, we specify the input and async package and then add a while statement and a variable collecting input after our print code in our main function.

```
package radium
package input
package async

func main(print)
      print(print)
      while true
            relay >> await(input())
--

main("Hello World!")
```

This creates a loop where it sets the global variable relay to the input, which the program waits for instead of forever setting the input without change.>> means equal in Radium, while means not equal. After collecting the input, we need to add a print statement.

```
package radium
package input
package async

func main(print)
      print(print)
      while true
            relay >> await(input())
            print(relay)
--

main("Hello World!")
```

This should, in theory, print "Hello World!" and start a loop that collects and prints your input. To test it, execute **radium run (file)** or some other command from a third-party compiler. Still, this program is a bit bland and needs a little pizzazz.

System Commands:

One of Radium's critical points is its low-level structure, so we will add statements that interact with your system. We need to integrate the package “system” to do this.

```
package radium
package input
package async
package system

func main(print)
      print(print)
      while true
            relay >> await(input())
            print(relay)
--

main(“Hello World!”)
```

After adding the system package, we'll add statements that add the input into a directory. We'll first need to check if the directory exists; if it doesn't, create it. If it does exist, move on. We'll then add creating text files in our while true loop.

```
package radium
package input
package async
package system

func main(print)
      number >> 0
      print(print)
      if readFile(“/directory”) >> (“null”)
            mkdir(“/directory”)
      else
            # Placeholder comment
      while true
            relay >> await(input())
            print(relay)
            number >> number + 1
            writeFile(“/directory/input” ~ number ~ “.txt”, relay)
--

main(“Hello World!”)
```

The added statements set a new global function (number) to 0 and then check if the directory “directory” exists. If it doesn’t, it creates it. If it already exists, we add a placeholder comment so the else question has a statement cause if it didn’t, the script would be invalid. We then add a statement that changes our global number variable by 1 for each input to give each .txt file in the writeFile statement a specific number. Now that we’ve spruced this program up, we could execute our final program with **radium run (file)** or a third-party compiler again. To finally add the cherry on top, we could compile the .rad file with nac compile x86_64 bin (file), compiling our program into an x86_64 binary file. You could change the processor from x86_64 to your processor to ensure it works on your computer or specific processor.

Great job! You just created your first Radium program.

Radium. Releasing soon.

(this is a preview of the radium documentation :p)