Milestones

Our team is

Xiru Zhu
Tianzi Yang
Hongji Chen

For this milestone, the work was divided as such
Xiru Zhu/Tianzi Yang, we worked on the scanner and parser.
We also wrote all the test cases and go programs.
All members participated in debugging the compiler.
Hongji Chen wrote the AST and pretty_print. Xiru Zhu originally
wrote an AST and incomplete pretty_print but that was scrapped in favor of the current one.

Our future plan is to have Hongji Chen write the weeding part
while Xiru Zhu and Tianzi Yang write up the type checker.
All members will work on converting the code to hopefully if
we have enough time to NASM. If not then we would probably convert it into c or c++.

The Compiler is implemented in C which we are most familiar with.
As such bison and flex are utilized as the most familiar tools.

Implementation Design

So in terms of general implemented, there a few things we need to weed out
With function calls and cast, there is a need for weeding in certain cases of casts.
The parser interprets the cast as a function call

instead and there is no way to check this inside the parser.

Example
e.g type myInt int
func foo(a int){
    return myInt(a)
}

Boolean true and false needs to be added as part of weeding/type check portion
As of now, they are ID as far as the parser is concerned.

In addition, for append expr 'append' '(' ID ',' expr ')'
We changed it to 'append' '(' primary_expr ',' expr ')' to allow for more extensive grammar.
I got annoyed when trying to implement a stack using current language.
This does add an extra weeding phase where a cast primary_expr must be
checked and given an error when attempted here.

Number Implementation
Integer are considered 32 bits
Floats are considered 64 bits

We also have some issues with number size.
The problem was representing very large floats and integers.
With current implementation, for integers we first check if such
number can be represented as an int. If it is not the case then
we attempt to field it as a float64. If that fails then we will
immediately return a compile error from the scanner itself. We think
this is a fair solution as we do not plan to deal with infinite number

representation. Now one of the possible issues with this system is that
integers with > ~16 significant digits will be rounded given the limitation
of 64 bits float. A Compiler Warning will be gi ven in such case with the line number.

For example
3223000000000000000000000000
Will be regarded to a float value and no warnin g would be printed
31232023034349334949349349499
Will be regarded as a float value but due to ro unding errors, a WARNING will be printed
3222222222220430939330494344444444444444444 4444444444444444444444444444444434343334
Will be unable to be represented as a float64 s o a COMPILE ERROR will be outputed.

Finally, all strings will be stored as RAW data . Therefore we do
not parse it at all for now and UNICODE values like
println("Hello, ä¸–ç•Œ") are in fact allowed an d will be printed by pretty_print.
Whether we will treat it as such with the compl ete compiler is yet to be determined.